# Wireless Sensor Networks: Structure and Algorithms

# Wireless Sensor Networks: Structure and Algorithms

**Draadloze Sensornetwerken:**
**Structuur en Algoritmen**
(met een samenvatting in het Nederlands)

Proefschrift

ter verkrijging van de graad van doctor
aan de Universiteit Utrecht op gezag van de
rector magnificus, prof. dr. G. J. van der Zwaan,
ingevolge het besluit van het college voor promoties

in het openbaar te verdedigen op
woensdag 3 december 2014 des ochtends te 10.30 uur

door

Thomas Christian van Dijk

geboren op 9 januari 1984 te Amsterdam

**Promotor:** Prof. dr. J. van Leeuwen

# Contents

# 1 Introduction

Over the course of the last decade, wireless technology has taken the world by storm. It might at this point seem like that was an inevitability, but it was not always so clear. For example, Bob Metcalfe—who is rightly famous for his contributions to networking, particularly his efforts in the design of Ethernet—put it like this in 1993:

> *"Wireless mobile computers will eventually be as common as today's pipeless mobile bathrooms. Portapotties are found on planes and boats, at construction sites, rock concerts, and other places where it is very inconvenient to run pipes. But bathrooms are still predominantly plumbed. For more or less the same reasons, computers will stay wired. (...) So let's just wire up our homes and stay there."*
>
> — Bob Metcalfe [Met93].

And here we are, twenty years later. That may not be a particularly *short* time, but wireless computing has well and truly arrived. Many people carry a smartphone connected to mobile internet; many local area networks are now wireless. For many purposes it is no longer necessary to wire up our homes. Being required to stay there seems old-fashioned.

Wireless technology is of course older than its widespread consumer adoption — indeed, on a computer-science scale of time, it is *much* older. The ALOHAnet system [Abr70] from the early 1970s, for example, was an influential achievement, providing a radio-based network between the Hawaiian islands. In this case, the specifics of a situation demanded a non-standard solution; today, wireless communication might be considered the standard assumption. Such a wireless world poses new questions for computer science and brings new perspectives for existing concepts. In this introduction we shall touch upon several such aspects and how the content of this thesis addresses them.

Some of the work in this thesis is motivated by a specific application of wireless communication: sensor networks. As a general term, *sensor networks* encompass many things; specific examples will follow, but in generic terms the properties that come to mind are as follows. A sensor network consists of a large collection of small devices, each equipped with sensors and a wireless transceiver, carrying out some data-gathering or surveillance task. These devices are typically called sensor nodes.

What sets these sensor networks apart from wireless networks in general is mainly their hardware and the manner of their deployment. Firstly, the hardware in a sensor network is typically considered to be of low cost and low reliability. Secondly, the deployment of a sensor network might not be carefully designed or executed, possibly because the environment is hard to reach or hard to service. These two aspects put a focus on fault tolerance since we may be working with devices that were designed to be disposable: better to handle failures in the network than to make individual devices more reliable. Depending on the deployment and the hardware, it may be impossible or impractical to have replaceable or rechargeable batteries. This puts an additional focus on energy efficiency: a device might be permanently lost when it runs out of energy.

Much of networking theory is based on graphs: they are the *de facto* standard model for wired communication networks. Graphs have, in a sense, proven to be the 'right' model. The properties of wireless transmissions, on the other hand, have turned out harder to capture in graphical models. In wireless networking, there are certainly interesting and useful graph-based results to be had. (We provide some in Part II of this thesis.) Yet it is still very much up for debate what the 'right' model for wireless networking is, if there is such a thing. Common among many of the new models being proposed is a strong focus on the physical properties of radio transmission. In Part I of this thesis we focus on two such physical models: a new model for localisation and new results in the well-known *Signal to Interference and Noise Ratio* model.

**Localisation.** One of the areas that has seen a shift to more physically-motivated models is *localisation*, a problem that is particularly interesting in the context of sensor networks: where am I? The physical location of a sensor node is often important as this is where the measurements were taken. It is greatly advantageous if the nodes can figure out, by themselves, where they are: then there is no tedious initial configuration and the network is robust against misplaced or moving nodes.

A typical approach for the localisation problem comes from the field of graph realisation. The basis for this approach is node-to-node distance measurement. Such measurements can be based, for example, on the received signal strength of a known-power transmission. Another option is to measure the time difference between transmission and receipt of messages. This results in (approximate) distances between certain nodes being known. These data can be combined in a graph with specified edge lengths. The abstract problem of drawing a set of points such that a given set of point pairs has given distances is known as *graph realisation*. This problem is $\mathcal{NP}$-hard in many settings and variants [Sax79]. Still, the problem is well studied, for example in work on sensor network localisation [DKQW10].

An alternative approach would be to use an off-the-shelf *global positioning system* (GPS). This certainly works well on smartphones. But, as we will argue concretely in the main text, GPS is often not suitable for deployment in sensor network applications. Furthermore, edge-length measurements are, in practice, often not available at the precision required for effective graph realisation. For these reasons, there is increasing interest in *range-free* localisation. This is localisation without explicit measurement of point-to-point distances. After localisation has succeeded, point-to-point distances can of course be inferred, but a range-free system does not take direct distance measurements as input.

In Chapter 3 we propose a novel range-free localisation approach. Our localisation is performed in relation to some base stations. These are assumed to have access to a permanent power source and a powerful radio transmitter. The system is then designed to be as simple and cheap as possible for sensor nodes: they are after all the most constrained part of the system.

The base stations provide an ongoing stream of transmissions that the sensor nodes can tune into when they need to localise. We analyse how well our system performs with a probabilistic transmission schedule. Additionally we design deterministic schedules with worst-case performance bounds.

**Scheduling wireless transmissions.** A normal network cable connects two endpoints; an interesting aspect of wireless communication is that it features a medium shared by many devices. One of the consequences is that a single transmission may be detected and successfully interpreted by multiple receivers: broadcasting comes for free with the medium. On the other hand, simultaneous transmissions might interfere, to the effect that neither message can be understood. This greatly complicates the problem of coordinating communication. In this way, wireless technology forces us to revisit basic modelling assumptions.

This many-to-many nature of the medium notwithstanding, graphs are a powerful tool for the analysis of wireless networks. (A tool that we gladly use in Part II of this thesis, where we are not necessarily concerned with broadcasting or interference.) A broadcast from a node in a communication graph can be modeled by letting a messages arrive at all neighbouring nodes instead of just at a single recipient. Interference can then be modeled by saying a *collision* occurs if multiple messages arrive at a node simultaneously: then none of these messages are received. Many results have been proven in this model. Often one considers graphs with structural properties that are reasonable for wireless networks, such as disc graphs, bounded doubling dimension [GKL03] or bounded independence [KNMW05].

In addition, there is an increasing output of research concerning models that are not based on graphs. An important such model is the *signal-to-interference-plus-noise ratio* (*SINR*) model. It has also been called—perhaps a little presumptuously—*the* physical model. It is, in any case, *a* physical model: physically motivated, physically reasonable and explicitly nongraphical.

There are variants and parameters, but a typical version goes like this. Consider transceivers located on the 2D plane. The model then posits that the power of transmitted signal diminishes as $1/d^2$, where $d$ is the distance from the transmitter. In particular, for a sender at point $s$ and a receiver at point $r$ the strength of the arriving signal equals $1/|r-s|^2$. This signal reaches all receivers: all transmissions are broadcast. To determine whether a message is correctly received, we look at the ratio of its signal strength versus the total strength all others signals (which interfere). A message is correctly received if and only if, *at the receiver*, it is at least as strong as the sum of all other signal strengths. We review the exact model in the preliminaries (Section 2.6).

In this *SINR* model, receivers are influenced most by nearby transmitters. On the other hand, the signals from many faraway transmitters can build up to cause significant interference over long distances. This global nature of interference is an interesting aspect of the model; one that is missing from graph-based models.

One of the reasons for the interest in the *SINR* model is the pioneering work of Gupta and Kumar [GK00] concerning the (stochastic) capacity of wireless networks in this model. Later it has become the dominant nongraphical model in algorithmic work following the investigation of transmission scheduling by Halldórsson et al. [GWHW09]. As opposed to earlier channel-capacity results, they look at worst-case networks. This way we are faced with a computational problem in the classical sense: given a network and a set of communication requests, devise a schedule that successfully transmits everything as quickly as possible.

Their initial paper proved $\mathcal{NP}$-hardness of this 'wireless scheduling' problem. Further results include ongoing research to give distributed approximation algorithms [HM11]. We argue that, in addition to that, it remains interesting to look at exact solutions—even though $\mathcal{NP}$-hardness dooms this to infeasibility for large instances. Being able to actually calculate optimal solutions provides valuable insight into their structural properties.

In Chapters 4 and 5 we study a problem that is intimately related to the scheduling problem. We are given the location of wireless transmitters and receivers. Along with this, we get a set of transmission requests: which transmitters have messages for which receivers. We then look to find a maximum *link independent set*, that is, a maximum-size set of transmissions that simultaneously succeed. This is the most requests we can immediately fulfill. The relation to scheduling all requests is that, at any time in a schedule, any simultaneous transmissions must be such a *link independent* set.

The problem of finding a link independent set of maximum cardinality is $\mathcal{NP}$-complete. We design a branching algorithm and analyse its moderately-exponential runtime. We implement the algorithm and demonstrate that it runs well. Using this implementation we then experimentally investigate the properties of random geometric instances. We additionally prove some of these properties. In particular, we prove that very large link-independent sets are unlikely—for a certain value of 'very large'—yet that rather large link-independent sets exist with high probability.

**Robust routing.** An issue that is important in wireless networking, and particularly in sensor networks, is fault tolerance: communication is not reliable. A particular wireless link might be less reliable than a cable. Also, mobile devices might go out of range, causing links to disappear—this can be a reasonable scenario where a cable becoming unplugged might be less so.

One reason that a wireless device might fail is that it runs out of battery power. This is particularly common in sensor networks, where devices are small and cheap. It can reasonably be assumed that not all nodes run out of battery power at the same time. Then we want the remaining part of the network to continue functioning. Deployment in harsh physical conditions might also lead to a significant hardware failure rate. Things like these make fault tolerance an important issue in wireless networks and sensor networks in particular.

Many of the problems studied in this thesis are computationally hard; likely too hard to solve within a real sensor network. As we argued for the link independent set problem before, it is still interesting to do the exact computations offline for analytical purposes. However, deploying the optimal solutions arrived at in this way seems like a dangerous move when failure tolerance is required. What use is a carefully crafted plan if it is no longer valid by the time it gets executed?

This leads us to look at *robust recoverability*, a concept from operations research that has recently seen development. In this framework one defines a 'simple' *recovery procedure* to deal with faults. Simple is here a relative concept, but we particularly want to run recovery within the network. The central idea of robust recoverability is then to take the behaviour of this recovery procedure into account during planning: we already know how the recovery procedure will behave and we can base our plan on this.

In Chapter 6 we study a very basic problem: finding a path between two given nodes in a graph. We will use this path to route packets in the network. The faults we consider are complete node failures. Perhaps a node has run out of battery power, perhaps it was fatally damaged; in any case, we can no longer use it. This is where the recovery procedure comes in. It consists of assigning, beforehand, a *backup node* for every node in the network. In case of failure, the backup node steps in as a replacement. (Perhaps the backup node was, up until that point, in sleep mode in order to save battery; perhaps the backup node will now serve double duty.) By assigning backups beforehand, this replacement can be handled within the network in an ad hoc fashion.

The computational problem is then to plan the path and its backup nodes. We formalise this problem and resolve the resulting complexity questions. Some variants are polynomial-time solvable and some are $\mathcal{NP}$-complete; we give algorithms for each. We also analyse the variation where we are given the path and have to select only the backup assignment. Again, some variants are hard and some are easy; again, we give algorithms for all of them.

**Energy-efficient data gathering.** The resources studied by computer science have primarily been time and space (memory). The exponentially-increasing transistor count provided by Moore's Law has given us increasingly fast processors, but also an increase in energy consumption [Max13]. On the scale of individual machines this has led to engineering problems in terms of heat dissipation. On an industrial scale, it has sparked interest in 'green computing' [Kur08]. But even without a concern for environmental footprint, wireless devices powered by a battery make energy consumption a very practical issue.

The energy limitations of battery-powered operation are especially tangible in the field of *sensor networks*. Here we consider small, simple, sometimes even disposable devices equipped with some computing power and a wireless transceiver. Coupled with sensors for physical quantities such a device is called a *sensor node*.

An example application would be sensor nodes scattered among the crops on a farm, measuring the amount of rainfall, sunlight, soil moisture et cetera. This can support control decisions for such things as irrigation and the application of fertiliser [WZW06]. As a different, more specific example, a network including sensors for strain, vibration and temperature has been embedded in the concrete of the Hollandse Brug, a bridge at Muiderberg in the Netherlands. The network is being used to monitor the structural health of the bridge [KBK$^+$10, VKV$^+$11].

The battery of sensor nodes is typically considered nonreplaceable. This is most clear in the example where the nodes are embedded in concrete.[1] Once deployed, a sensor node performs its task until it breaks down. This provides a direct link between between energy consumption and the operational lifetime of a sensor network.

As a final topic in this thesis we consider energy consumption. This has already influenced design decisions in Chapter 3, but in Chapters 7 and 8 we look explicitly at energy budgets. We study a task that is often central in sensor networks: gathering the data from the sensors in a base station in the network. How much data can we gather before exhausting the network? We model this as a graph problem and, for its relation to classical network flow, call it *energy-constrained flow*.

[1]In the case of the bridge, one might consider wired power and networking. The advantage of a wireless approach is in the ease of deployment and the low impact on the overall engineering of the bridge.

We show that the problem of maximising the amount of data gathered is strongly $\mathcal{NP}$-complete and even $\mathcal{APX}$-hard. When restricted to geometric networks, with physically reasonable energy costs, the problem remains hard. For graphs of bounded treewidth, we give pseudopolynomial-time algorithms.

Then we look to find good solutions despite this hardness. We develop heuristic algorithms based on linear programming and column generation. Experiments with an implementation of these algorithms demonstrate their effectiveness. Rounding of the linear program also gives efficient approximation algorithms, which works particularly well for $st$-planar graphs.

In dealing with these topics, we will have touched on many aspects of wireless sensor networks, their structure and the related algorithms. Before the thesis proper begins, let us briefly draw attention to two divisions that it contains. The first is readily apparent from the table of contents: a Part I about physical models and a Part II about graphical models. This is a technical divide and merely a result of organising the material.

The second divide is methodological in nature and a divide that, where possible, we have tried to bridge. On the one hand, we have theoretical, worst-case results about the runtime of algorithms and the structural properties of wireless networks. On the other hand, we have experimental results based on implementations of our algorithms and on simulations. These two approaches are sometimes seen as being at odds with each other—indeed, as the saying goes: *"In theory, theory and practice are the same. In practice, they are not."*[2] In recognising this difference, we find that in fact theoretical and experimental research complement each other to give a more comprehensive view of the subject.

As an example in this thesis, we can point to Chapters 4 and 5 about the LINK INDEPENDENT SET problem. Even though Chapter 4 is mainly concerned with the formal correctness and worst-case runtime of an algorithm, the design of the algorithm was very much informed by experiments in the early stages of the research. Then, in Chapter 5 we complement the worst-case result with runtime measurements on an actual implementation of the algorithm. With this implementation we observe some structural properties of wireless networks and, moving to theory again, continue to prove some of the observed behaviour. As another example, the concept of a *regular* schedule (Chapter 3) and the corresponding theorems were inspired by the experimental results in Table 3.2. In these ways, there is a back and forth between theoretical and experimental results.

It is important to not be satisfied too easily with experimental appearances: models and theories are what computer science is built on, for good reason. Yet a theorem might not tell the whole story, and pure theory might not be how we arrive at a result (cf. [Tic98]).

[2]Variously attributed to Albert Einstein, Jan van de Snepscheut, Yogi Berra, and others.

# 2 Preliminaries

In this chapter we provide a baseline for the concepts and notation as used in this thesis. We review some 'textbook' definitions and properties and give some auxiliary facts that will be used later on.

## 2.1. Sets, graphs and asymptotics

### Sets

*Sets* are denoted by capital letters ($S$), sometimes in calligraphic font ($\mathcal{S}$). We use the following notation for *set comprehension*, the operation of building the set of all elements $x$ from a certain domain $S$ that satisfy predicate $P$: $\{ x \in S \mid P(x) \}$.

For integer $k$, we write $S^k$ to mean the $k$-times-repeated Cartesian product of $S$ with itself. In particular, $S^2 = S \times S$ is the set of ordered pairs of elements of $S$. We write $2^S$ for the power set of $S$, that is, the set of all subsets of $S$.

A *permutation* of a set is an ordering of its elements. The *reversal* of a permutation is the permutation given by the reverse ordering. A *rotation* of a permutation is a cyclic shift of the ordering.

### Graphs

A *graph* $G = (V, E)$ consists of a set $V$ of *vertices* and a set $E$ of *edges*. Vertices are also called *nodes*; we make no distinction. An *edge* in a *undirected graph* is an unordered pair of vertices. In a *directed graph*, the pair is ordered and is often called an *arc* instead. In either case—ordered or not—we write the pair as $(v, w)$.

A *path* in a graph is an ordered list of vertices such that an edge (or arc) exists between all successive vertices on the path. We write paths using square brackets, like $[p_1, p_2, \ldots, p_k]$. The *length* of a path is the length of this list of vertices. A path is called *simple* if the vertices in it are unique. Alternatively, a path can be considered as a set of edges (or arcs).

A graph may be drawn in the 2D plane, with each node corresponding to a point and each edge $(v, w)$ corresponding to a curve connecting the points for $v$ and $w$. An embedding of a graph is a drawing such that none of the edges cross. A graph for which such an embedding exists is called a *planar graph*. A graph with two designated nodes $s$ and $t$ is called $st$-*planar* if and only if it has a planar embedding with $s$ and $t$ on the same face. Every $st$-planar graph has a planar embedding in which $s$ and $t$ both lie on the outer face. A graph is $st$-planar if and only if the graph remains planar after adding the edge $(s, t)$.

Let $G$ be a graph with edge weights and let $G'$ be a spanning subgraph of $G$, that is: $G'$ has the same vertex set and a subset of the edges. Then distances between pairs of vertices may be higher in $G'$ than in $G$. The largest factor by which distances increase is called the *stretch factor* (or: *dilation factor*) of $G'$ with respect to $G$. A spanning subgraph with low stretch factor is called a *spanner*. If $G$ is a complete graph, with its vertices corresponding to points in the plane, and the edge weights are Euclidean distances, then it is known that the Delaunay triangulation of the point set is a planar spanner with stretch factor bounded by a constant less than 1.998 [Xia13].

### Asymptotics

We use the typical assortment of asymptotic notations (see any textbook, for example [CLRS09]). In addition we will sometimes use *big-Oh-star* notation, writing $\mathcal{O}^*(\, f(n)\,)$ to mean $\mathcal{O}(\, f(n) \cdot p(n)\,)$ for some polynomial $p(n)$. This notation suppresses polynomial factors when we are not interested in them, as the $\mathcal{O}(\,\cdot\,)$ notation suppresses constant factors.

Consider for example a runtime of the form $c^n \cdot p(n)$, for some $c > 1$. Any increase in $c$ dominates *all* polynomial factors: it is bounded by $\mathcal{O}(\, d^n\,)$ for all $d > c$. For this reason, *Oh-star* notation can be reasonable when bounding exponential runtimes. Similarly, we write $\Theta^*(\,\cdot\,)$ and $o^*(\,\cdot\,)$.

## 2.2. Probability

We denote the probability of an event $E$ by $\mathbb{P}[E]$. We write $\mathbb{P}[E \mid F]$ for the probability of $E$ conditioned on event $F$. The expected value of a random variable $X$ is written as $\mathbb{E}[X]$; its variance as $\mathbb{V}\mathrm{ar}[X]$. Variance is related to expected values by the following equality, sometimes expressed as *"expected square minus square expected:"*

$$\mathbb{V}\mathrm{ar}[X] \;=\; \mathbb{E}[X^2] - \mathbb{E}[X]^2.$$

By *linearity of expectation* we mean the well-known property that

$$\mathbb{E}[X + Y] \;=\; \mathbb{E}[X] + \mathbb{E}[Y].$$

Recall that X and Y need not be independent for this equality to hold.

A set of random variables that are mutually independent and are drawn from the same distribution is called *independent and identically distributed*. The abbreviation i.i.d. is used for this.

The minimum value of a set of random variables is called its first *order statistic*. Note that this order statistic is itself a random variable. More generally, the $k^{th}$ order statistic is defined as the $k^{th}$ smallest value among a set of random variables. The following are two basic results on order statistics (see for example [DN05]).

**Proposition 2.1 (Uniform order statistic).** *Consider $n$ independent random variables, each variable taken uniformly at random from $[0, 1]$. Let $X_{\min}$ be their minimum. Then $\mathbb{E}[X_{\min}] = \frac{1}{n+1}$.*

**Proposition 2.2 (General order statistic).** *Consider $n$ independent random variables, each variable with the same probability density function $f(x)$. Let $f_{\min}$ be the probability density function of their minimum. Then*

$$f_{\min}(x) \;=\; n \cdot f(x) \cdot \left( \int_x^\infty f(y)\,dy \right)^{n-1} .$$

*Proof sketch.* Let the minimum of the $n$ variables equal $x$. Then some variable has value $x$ and the $n-1$ other variables have value at least $x$. There are $n$ possibilities for which variable attains the minimum. □

An event $E$ is said to occur *almost surely* if $\mathbb{P}[E] = 1$. We also use this concept asymptotically to mean that a probability converges to 1 as some indicated variable goes to infinity.

Lastly, we will use some *Chernoff bounds* in Chapters 5 and 8. These are exponentially vanishing tail bounds on sums of independent variables. In particular, we use the following bound.

**Proposition 2.3 (Multiplicative Chernoff bound).** *Let $X$ be the sum of finitely many independent random variables taking values in $\{0, 1\}$ and let $\mu = \mathbb{E}[X]$. Then for any $\delta > 0$,*

$$\mathbb{P}[X > (1+\delta)\mu] \;<\; \left( \frac{e^{\delta}}{(1+\delta)^{(1+\delta)}} \right)^{\mu} .$$

For all further notation from probability theory used in this thesis we refer to standard texts, for example [MU05].

## 2.3. Stochastic point processes

In Chapter 5 we will analyse properties of the Link Independent Set problem on random geometric instances. We base our models of random instances on well-known point processes. Here we quickly review some of the pertinent definitions, specialised to reasonable subsets of $\mathbb{R}^2$—this is all we will need.

First we consider binomial point processes. In these processes, the number of points is fixed. This can be convenient, for example when we want to talk about the runtime of an algorithm on instances generated using such a process: we know exactly how many points we will get.

**Definition 2.4 (Binomial point process).** A point process on region $A$ is called *binomial with* $n$ *points* if and only if it consists of $n$ points that are i.i.d. over $A$. In particular, it is called *uniform* if and only if each point is distributed uniformly at random in $A$.

In some cases we look at Poisson point processes instead. These processes, too, are a common model for random point sets, used as a model for networks already in the early '60s [Gil61]. While Poisson point processes do not have a fixed number of points, they have an independence property that can make them easier to handle.

Recall the Poisson distribution with parameter $\lambda$: the limit of a binomial distribution with $n$ trials and success probability $p$, as $n$ goes to infinity and $n \cdot p = \lambda$ is kept fixed. This distribution has expected value $\lambda$.

**Definition 2.5 (Poisson point process).** A point process on region $A$ is called *Poisson with density* $\lambda$ if and only if

- for all bounded $B \subseteq A$, the number of points in $B$ is Poisson distributed with parameter $\lambda \cdot \|B\|$, where $\|\cdot\|$ is the Lebesgue measure (think: 'area'),
- for any collection of disjoint subsets $B_i \subseteq A$, the events in the different subsets are mutually independent.

Note that this definition directly has the following consequence.

**Proposition 2.6.** *The number of points in a Poisson point process with density* $\lambda$ *on a region of (finite) measure* $a$ *has expected value* $\lambda \cdot a$.

This proposition allows us to probably get approximately $n$ points by setting the density $\lambda = n/a$. As we will see in Chapter 5, this density $n/a$ is a meaningful parameter for the Link Independent Set problem in both the binomial and the Poisson model.

## 2.4. Treewidth

In Chapter 7 we use the concept of *treewidth*. Here we give the required definitions; for a proper introduction to this rich subject, see for example the overview by Bodlaender [Bod93].

**Definition 2.7 (Tree decomposition).** A tree decomposition of a graph $G = (V, E)$ is a pair $D = (X, T)$ where $T = (P, F)$ is a tree and $X = \{X_p \mid p \in P\}$ is a family of subsets of $V$, one for each node of $T$, such that

- $\bigcup_{p \in P} X_p = V$,
- for every edge $(i, j) \in E$, there exists a $p \in P$ with both $i \in X_p$ and $j \in X_p$,
- for all $p, q, r \in P$: if $q$ is on the path from $p$ to $r$ in $T$, then $X_p \cap X_r \subseteq X_q$.

The elements of $X$ are called *bags*.

**Definition 2.8 (Treewidth).** The width of a tree decomposition

$$(\ \{X_p \mid p \in P\},\ T = (P, F)\ )$$

is $\max_{p \in P} |X_p| - 1$. The treewidth of a graph $G$ is the minimum width over all possible tree decompositions of $G$.

**Definition 2.9 (Graph minor).** An undirected graph $G = (V, E)$ is a *minor* of a graph $H = (W, F)$ if and only if $G$ can be obtained from $H$ by a series of vertex deletions, edge deletions and edge contractions. Here an edge contraction is the operation that removes two adjacent vertices $v$ and $w$, and replaces them by a new vertex that is adjacent to all vertices that were adjacent to $v$ or $w$.

It is well known that the treewidth of a graph does not increase when taking minors. We will furthermore use the following result on graphs with small treewidth.

**Lemma 2.10 (Ramachandramurthi [Ram97]).** *If $G$ is a simple, undirected graph of treewidth at most $k$ that is not a clique, then $G$ has two non-adjacent vertices of degree at most $k$.*

**Corollary 2.11.** *Let $G$ be a simple, undirected graph of treewidth at most two, with at least three vertices. Let $s$ and $t$ be any two adjacent vertices. Then there is a vertex of degree at most two that is neither $s$ nor $t$.*

*Proof.* If $G$ is a clique, then it has exactly three vertices, and the result trivially holds. Otherwise, let $i$ and $j$ be the two non-adjacent vertices of degree at most two, as indicated by Lemma 2.10. Since $(s, t)$ is an edge of $G$ and $(i, j)$ isn't, at least one of $\{i, j\}$ is not contained in $\{s, t\}$. □

In this thesis, when referring to the treewidth of a directed graph (which is conventionally left undefined) we shall mean the treewidth of the underlying undirected graph, that is, the graph obtained by dropping the direction of the arcs.

## 2.5. An exponential bound on certain binomials

In Chapter 4 we need the following known bound, for which we briefly include a proof here.

Given a set $S$ of size $n$ and a constant $c \leqslant \frac{1}{2}$, we want a bound for the number of subsets of $S$ that have size at most $c \cdot n$. The exact number of subsets involved is

$$\sum_{i=0}^{\lfloor cn \rfloor} \binom{n}{i}.$$

Since $\binom{n}{i}$ is increasing in $i$ when $i < n/2$, the largest term is $\binom{n}{\lfloor cn \rfloor}$. We sum over at most $n/2$ terms, so the whole sum is at most $n/2$ times the largest term. This factor can be dropped to obtain a bound of $\mathcal{O}^*( \binom{n}{\lfloor cn \rfloor} )$.

For comparison with bounds of the form $b^n$ we will use the following lemma.

**Lemma 2.12.** *Let* $c \in [0, \frac{1}{2}]$. *Then* $\binom{n}{\lfloor cn \rfloor}$ *is* $\mathcal{O}^*( b^n )$, *where* $b = \frac{1}{c^c(1-c)^{1-c}}$.

*Proof.* Let $k = \lfloor cn \rfloor$. It follows that $cn - 1 < k \leqslant cn$, hence:

$$c - \frac{1}{n} < \frac{k}{n} \leqslant c \tag{2.1}$$

Then:

$$\binom{n}{\lfloor cn \rfloor} = \binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{2.2}$$

Recall Stirling's approximation for the factorial:

$$n! = \sqrt{2\pi n}\, e^{-n}\, n^n\, (1 + \mathcal{O}( n^{-1} )) \tag{2.3}$$

Since we will not be interested in polynomial terms, we immediately simplify this to $\Theta^*( e^{-n}n^n )$. Then:

$$\begin{aligned} \frac{n!}{k!(n-k)!} &\overset{\text{poly}}{\approx} \frac{e^{-n}n^n}{e^{-k}k^k\, e^{-(n-k)}(n-k)^{n-k}} && (2.4)\\ &= \frac{n^n}{k^k(n-k)^{n-k}} && (2.5)\\ &= \frac{1}{\left(\frac{k}{n}\right)^k \left(1-\frac{k}{n}\right)^{n-k}} && (2.6)\\ &= \left( \frac{1}{\left(\frac{k}{n}\right)^{\frac{k}{n}} \left(1-\frac{k}{n}\right)^{1-\frac{k}{n}}} \right)^n && (2.7) \end{aligned}$$

Since $1 - \frac{k}{n} \geqslant 1 - c$, we have:

$$\frac{1}{\left(1-\frac{k}{n}\right)^{1-\frac{k}{n}}} \leqslant \frac{1}{(1-c)^{1-c}} \tag{2.8}$$

Similarly $\frac{k}{n} > c - \frac{1}{n}$, which gives:

$$\left(\frac{k}{n}\right)^{\frac{k}{n}} > \left(c - \frac{1}{n}\right)^{c-\frac{1}{n}} = c^{c-\frac{1}{n}} \cdot \left(1 - \frac{1}{cn}\right)^{c-\frac{1}{n}} = c^c \cdot \left(\frac{1}{c}\right)^{\frac{1}{n}} \cdot \left(1 - \frac{1}{cn}\right)^{c-\frac{1}{n}} \tag{2.9}$$

Applying the inequalities (2.8) and (2.9) to Equation (2.7), we get the following, where we let $b = \frac{1}{c^c(1-c)^{1-c}}$:

$$\binom{n}{\lfloor cn \rfloor} \leqslant \left(\frac{1}{c^c\,(1-c)^{1-c}}\right)^n \cdot \left(\frac{1}{\left(\frac{1}{c}\right)^{\frac{1}{n}}\,\left(1-\frac{1}{cn}\right)^{c-\frac{1}{n}}}\right)^n \tag{2.10}$$

$$= b^n \cdot \frac{c}{\left(1-\frac{1}{cn}\right)^{cn-1}} \tag{2.11}$$

Observing that in the limit, as $n$ goes to infinity, $\left(1 - \frac{1}{cn}\right)^{cn} \approx e^{-1}$, we can simplify this to:

$$\binom{n}{\lfloor cn \rfloor} \leqslant b^n \cdot \frac{c}{\left(1-\frac{1}{cn}\right)^{cn-1}} \approx b^n \cdot ce \cdot \left(1 - \frac{1}{cn}\right) \tag{2.12}$$

This bound is $\mathcal{O}^*(\, b^n \,)$. □

## 2.6. The physical model for wireless transmissions

In Chapters 4 and 5 we will work with *the physical model* for wireless transmissions. For notation related to the physical model (also called: the *SINR* model) we follow, for example, Halldórsson et al. [HW09] and Avin et al. [AEK+12].

We consider $n$ requests for communication $\ell_1 \ldots \ell_n$. These are called *links*. Each of these represents a unit demand: the requests are for exactly one transmission for each link. As is typically the case in this model, we are mainly concerned with the geometric case where links are defined from points in the Euclidean plane: a link (that is: a transmission request) comes from a sender at point $s_i$ and is intended for a receiver at point $r_i$. Let $S$ be the (multi)set of senders $s_i$ and $R$ be the (multi)set of receivers $r_i$. Note that $|S| = |R| = n$, since each of the $n$ links has exactly one sender and one receiver; if a single physical device occurs as a sender or receiver in multiple requests, its position is included in $S$ (or $R$, respectively) as multiple occurrences of the same point.

For points $p$ and $q$, let $d(p, q)$ be the Euclidean distance between them. We now define an (asymmetric) distance between links $i$ and $j$: $d_{ij} = d(s_i, r_j)$, that is, the distance from sender $i$ to receiver $j$. For $i = j$, this is called the *length* of the link: $\ell_i = d_{ii}$.

In the geometric *SINR* model, the points and distances determine received signal strength (RSS) according to the commonly used *path-loss radio propagation* model. This model states that a signal becomes weaker over distance as follows: the strength of a signal that has travelled distance $d$ from its transmitter is proportional to $d^{-\alpha}$ for some $\alpha \geqslant 2$, with a constant to model transmitters of different strength. This $\alpha$ is called the *path loss exponent* and we take the usual choice of $\alpha = 2$ when we need a concrete value. Note that the signal strength is then inversely proportional to the surface of a sphere in three dimensions. To some degree, this spreading out of the signal over a larger surface area is the physical explanation of the loss of signal strength. Depending on the physical circumstances, higher values of $\alpha$ may be more accurate in practice because of various other loss factors.

Instead of getting the signal strengths from a geometric model, the *abstract SINR* model considers all received signal strength values as arbitrary. This is of practical use, because received signal strength can be measured in a deployed network and then the actual values can be used [RMR+06]. We do not consider inbetween models, for example based on more accurate models of path loss or based on metric-like *decay spaces* [BH14, GÁB+14].

We can now consider the *gain matrix* (or: *path loss matrix*) of a set of links: an $n \times n$ matrix whose entries give the amount of signal from sender $i$ that arrives at receiver $j$. Note that if $i = j$, then this is the received signal strength of the requested transmission for that link; otherwise it is the interference at receiver $j$ caused by the transmission from sender $i$.

**Definition 2.13 (Gain matrix).** A *gain matrix* $M$ is an $n \times n$ matrix holding the amount of *signal* $g_{ii}$ received for link $i$ on the diagonal, and the amounts of *interference* $g_{ij}$ in the off-diagonal entries. For $i \neq j$, the element $M[i, j]$ is the interference at receiver $i$, caused by sender $j$. An element on the diagonal is called a *signal*; an element off the diagonal is called an *interference*.

**Definition 2.14 (Geometric gain matrix).** A gain matrix is called geometric if all entries equal $d_{ij}^{-\alpha}$.

A more general notion of a geometric gain matrix allows the senders to transmit with varying power. Then the entries in the gain matrix are of the form $P_i/d_{ij}^{\alpha}$, where $P_i$ is the power of sender $i$. In Chapters 4 and 5 we only consider equal-power transmissions, effectively setting all $P_i = 1$.

Consider now a set of simultaneous transmissions. Let $X \subseteq S$ be the set of senders involved in the transmissions, all transmitting with equal power, and let $N$ be an amount of background noise. The *signal-to-interference-plus-noise (SINR)* function gives, for a sender $s_i \in X$ and a point $p$ in the plane, the quality of the *signal* from $s_i$ compared the *interference* by the simultaneous transmissions from other senders and the background *noise*. This function is defined as the ratio between 'good' signal and 'bad' signal:

$$\mathit{SINR}_X(s_i, p) = \frac{d(s_i, p)^{-\alpha}}{N + \sum_{j \in X, j \neq i} d(s_j, p)^{-\alpha}}. \tag{2.13}$$

This is the quality of the signal from $s_i$ as received at point $p$, given simultaneous transmissions from all senders in $X$. When $X$ is clear from context, as is usually the case, we will omit the subscript.

A receiver at location $p$ is defined to successfully hear a transmission by sender $i$ if and only if $\mathit{SINR}(s_i, p) \geqslant \beta$. This value $\beta$ is called the *reception threshold*. Certainly $\beta > 0$, but we specifically consider the case $\beta > 1$: in order to be received correctly, a signal must be stronger than the interference and noise.

The set of points in the plane at which a sender can be heard is called its *reception zone*. In this thesis we follow the definition by Avin et al [AEK+12]. The case $p = s_i$ adds some technical tedium to the definition (signal strength $0^{-\alpha}$), but the concept is clear: the set of points $p$ at which Equation 2.13 has value at least $\beta$.

**Definition 2.15 (Reception zone).** The reception zone of sender $i$ is

$$\mathcal{H}_i = \{p \in \mathbb{R}^2 - X \mid \mathit{SINR}(s_i, p) \geqslant \beta\} \cup \{s_i\}.$$

Note that if $\beta > 1$, then every point is contained in at most one reception zone, since only a single sender can have the majority of the signal at a point.

**Definition 2.16 (*SINR*-condition).** For a link $i$ and a set of transmissions $X$, the *SINR-condition* of the link is

$$\mathit{SINR}_X(s_i, r_i) \geqslant \beta.$$

In this thesis we take $N = 0$, that is: we assume no background noise. This assumption is not entirely unrealistic: by increasing the transmission power of all transmitters by an equal factor, the influence of background noise can be made arbitrarily small. In the interest of energy efficiency, it could be beneficial to directly take the background noise into account, but we will not pursue this.

We are now ready to define the key object of study in Chapters 4 and 5: link independent sets.

**Definition 2.17 (Link Independence).** A set $\mathcal{I} \subseteq S$ is called *link independent* (or: *SINR*-feasible) if simultaneous transmissions from senders in $\mathcal{I}$ are correctly received at the corresponding receivers. That is, if and only if $\mathit{SINR}_{\mathcal{I}}(s_i, r_i) \geqslant \beta$ for all $s_i \in \mathcal{I}$.

When convenient, we might also consider $\mathcal{I}$ to be a set of indices instead. This will be clear from context. In the geometric model with no background noise, this definition of link independence means that

$$\frac{d_{ii}^{-\alpha}}{\sum_{j \in \mathcal{I}, j \neq i} d_{ji}^{-\alpha}} \geqslant \beta, \quad \forall i \in \mathcal{I}. \tag{2.14}$$

This property naturally gives rise to the LINK INDEPENDENT SET problem: finding a maximum cardinality link independent set. For geometric instances we get the following problem statement.

| | **Link Independent Set (Physical model)** |
|---|---|
| *Instance:* | Set of $n$ links as pairs of points $(s_i, r_i)$. |
| | Path-loss exponent $\alpha$. |
| | Reception threshold $\beta$. |
| | Integer $k$. |
| *Question:* | Does there exist a subset consisting of at least $k$ of the links that is link independent? |

For arbitrary gain matrices, the problem statement is as follows.

| | **Link Independent Set (Abstract)** |
|---|---|
| *Instance:* | An $n \times n$ gain matrix with non-negative entries. |
| | Reception threshold $\beta$. |
| | An integer $k$. |
| *Question:* | Does there exist a subset consisting of at least $k$ of the links that is link independent? |

This problem is of interest because it is the highest number of transmission requests that can be realised simultaneously. In the physical model, the problem is $\mathcal{NP}$-hard [GWHW09]. It does admit a constant-factor approximation, even when we allow background noise and are looking for a *maximum-weight* link independent set [XTW10]. The problem has been studied under various names, based on the motivation and on variations of whether there is background noise and whether the transmission power of the senders is given as input or instead to be decided as output—for example: One-slot Scheduling [GWHW09], One-shot Scheduling [HW09], Link Capacity [GÁB+14] and Max-Connections [AD09].

We prefer the name *link independent set* because of the relation to the classic graph-theoretic concept of *independent sets*. In fact, the abstract Link Independent Set problem straightforwardly subsumes the Independent Set problem for graphs as follows.

**Lemma 2.18.** *For every instance $I_G$ of* Independent Set *on graphs, there exists an instance $I_L$ of* Link Independent Set (Abstract) *such that: the nodes in $I_G$ correspond one-to-one with links in $I_L$, and the independent sets in $I_G$ correspond one-to-one with the link independent sets in $I_L$.*

*Proof.* Let $G = (V, E)$ be a graph. Make a link for every vertex in $V$ and set the signal amount $g_{ii}$ to 1 for all $i$. Set $g_{ij}$, the interference sender $i$ causes at receiver $j$, equal to 1 if $(i, j) \in E$ and set it to 0 otherwise. Pick any reception threshold $\beta > 1$. There is now a one-to-one correspondence between link independent sets in the constructed instance and (graph) independent sets in G. Consider an independent set in G: the corresponding transmissions are link independent, because no receiver gets any interference. Conversely, consider a link independent set in the constructed instance. The signal strengths and $\beta$ are such that any interference at a receiver ruins that transmission: then this set of transmissions corresponds to an independent set in G. □

Pushing this comparison between simultaneous transmissions and independent sets further, one can consider the Scheduling problem of Goussevskaia et al. [GWHW09]: the problem of scheduling all requests (links) in the least number of timeslots possible, where the transmissions in each timeslot must be *SINR*-feasible, that is, link independent. In our terminology this could be termed Link Colouring, since colouring is covering with independent sets: cover the entire set of requests with the least number of link independent sets. Given this correspondence it is not surprising that an inclusion/exclusion algorithm in the style of Björklund and Husfeldt [BH06] can be used to do Link Colouring in $\mathcal{O}^*(2^n)$ time [HL08].

## 2.7. Mergeable heaps

Here we describe the *mergeable heap* data structure used in Chapter 7. This data structure is used for the efficient manipulation of a collection of data elements. Each element has a *key* and possibly more information stored. This collection is partitioned into disjoint sets. In the data structure the following operations are supported.

- Create a new set with one new element, with a key value and possibly other information.
- For a given set, obtain a pointer to the object that represents an element with *maximum* key value (and thus obtain this maximum, and possibly update the associated information).
- For a given set, delete the element with maximum key value.
- For two given sets, take the *disjoint union* of the two sets, that is, replace these sets by their disjoint union.

Specific implementations of mergeable heaps include the so-called binomial heaps and the Fibonacci heaps, see for example [CLRS09, Chapter 19].

**Lemma 2.19.** *There is a data structure that allows unions of sets, obtaining the element with maximum key, deleting the element with maximum key, and creating new one element sets, such that each operation takes at most $\mathcal{O}(\log d)$ time, where $d$ is the maximum size of any set that is built.*

*Proof.* In the following simple manner, we can implement this data structure, such that each operation takes at most $O(\log d)$ time, where $d$ is the maximum number of elements in a set. Each set is represented by a binary tree, with each node containing an element of the set, such that each node except the root has a parent whose key value is equal or larger, that is, key values are *non-increasing* on each path from the root to a leaf. In addition, each node stores the number of nodes in the subtree of which it is the root. A set is represented by a pointer to its root.

Finding an element with maximum key value is trivial, as the root has a maximum key value. A union can be done as follows. Suppose trees $T_1$ and $T_2$ represent what we want to take the disjoint union of. Compare the key values of the roots $r_1$, $r_2$ of $T_1$ and $T_2$ respectively. Without loss of generality, suppose that the key value of $r_1$ is at least the key value of $r_2$. If $r_1$ has at most one child, we make $r_1$ the new parent of $r_2$, and update the number of nodes in the subtree with root $r_1$ accordingly. Otherwise, let $s_1$ and $s_2$ be the two children of $r_1$. Check which of the three nodes $r_2$, $s_1$, and $s_2$ has the largest number of nodes in the subtree of which it is the root, say $q$. Now, build the tree as follows: let $r_1$ be the root. One child of $r_1$ is $q$. Then, recursively build a tree that contains all elements in the two subtrees whose root is in $\{r_1, s_1, s_2\} - \{q\}$, and then make $r_1$ the parent of the root of this tree.

Suppose the number of nodes of $T_1$ plus the number of nodes of $T_2$ is $\alpha$. Then the number of nodes of the subtree with root $q$ is at least $\alpha/3$, so the recursive step has two trees which together have at most $2\alpha/3$ nodes. Thus, the recursion depth is $O(\log d)$, and a union is done in $O(\log d)$ time.

A deletion of the maximum element can simply be done by deleting the root node, and then performing the union procedure on its children. □

Note that we cannot expect a much faster data structure since we can sort $d$ elements using $\mathcal{O}(d)$ operations on the data structure: create a set for each element, then take the union of all sets and iteratively obtain and delete the element with maximum key until no elements are left. The order in which the elements are deleted is sorted, from largest to smallest. Thus, in typical models of computation and without further assumptions on the input, $\mathcal{O}(d)$ operations on the data structure have to cost $\Omega(d \log d)$ time. This gives a lowerbound of $\Omega(\log d)$ on the worst case for a single operation.

# Part I

# Physical models

# 3 Range-free localisation using splitline schedules

In the localisation problem in wireless sensor networks, a wireless node wants to discover its physical location. Many solutions for this problem are known, with many different model assumptions and design objectives (see for example [AK09, BT05, SSGE04]). We consider the case where accurate point-to-point distance measurements are not available and put an emphasis on minimising the energy usage of the sensor nodes. This leads us to a model in which the sensor nodes passively listen in on a schedule of transmissions by some base stations (beacons). These transmissions allow a sensor node to localise itself on the line segment between two base stations, or (in higher dimensions) in a cube spanned by base stations at the vertices. The quality of a transmission schedule can be evaluated by measuring how long it takes a sensor node to learn its position to a certain precision.

First we analyse the quality of random schedules. Then we define a class of 'regular' schedules and conjecture that a $k$-regular schedule is optimal if and only if $k$ is co-prime to $n$, the length of the schedule. We computationally confirm the conjecture for $n \leqslant 14$, and prove the case $n \textbf{ mod } k \equiv k-1$. The general case remains open. Furthermore, we give a schedule with a performance that is at most a factor $4 + 4/t$ from optimal after $t$ transmissions.

## 3.1. Introduction

Localisation is an important problem in wireless sensor networks. Many applications involving monitoring, surveillance or logging of measurements rely on this problem. The (geometric) location of the sensor is important here: where was the measurement by the sensor taken or where was some event detected? Also in the case of location aware services a mobile node may need to know its location.

The objective of localisation is to relate the positions of the sensors to the positions of known fixed base stations (or: anchor nodes). Alternatively, positions can be relative only to the (unknown) location of other nodes. The former calls for a fixed infrastructure, while the latter may make sense in an ad hoc network. We will work with fixed base stations.

In the literature, localisation systems using GPS are considered unsuitable for sensor network applications [LEH04, NSB03]. We will look at the issues, as they will help us identify our design objectives. The problem with GPS is that it is expensive in several aspects. GPS hardware requires a high-powered receiver (leading to higher energy consumption) and is relatively large and expensive. Our localisation scheme works with a regular radio receiver that is likely to be available in any sensor node anyway.

### 3.1.1. Motivation for range-freeness

In localisation literature, it is often assumed that distances between nodes are available. Also the related literature on graph realisability is built on the availability of such distances. However, in practice these distances may not be available at any useful precision. This is due to problems such as the following.

- In practice, distance cannot be reliably computed from the received signal strength [GKW+02]. Signal strength is often taken to fade proportional to approximately $d^{-2}$, where $d$ is the distance between the sender and the receiver. Reality is not so clean and variation occurs in the constant factors involved as well as the exponent. This means that a system using received signal strength would probably have to be calibrated. It is unclear that it would be possible to do this automatically. And even then, effects such as signal reflections and additional fading due to obstructions (walls, pillars) make distance measurements unreliable.
- Using time-in-flight measurements and time-difference-of-arrival (TDOA) is also complicated by practical issues. They require very precise clocks and in some cases even precisely synchronised clocks. Requiring the former is unfortunate (due to the cost of good clocks), while requiring the latter is a very serious issue in sensor networks [FL06].

To circumvent these problems we will work in a weaker model than point-to-point distance measurements. Such a model is called range-free (e.g. [HHB+03]).

In our model, sensor nodes can detect the arrival *order* of messages. This is a qualitative way of using *time difference of arrival* information, requiring no clock at all. We use only a small number of localisation base stations that do have good clocks and good synchronisation. We will first assume perfect synchronisation. Later on we investigate the effect of timing errors.

## 3.1.2. Motivation for passiveness

We make a further assumption: we want to support networks spanning a large area that may be populated with sensor nodes only sparsely. This leads us to assume that direct transmissions from nodes to base stations may be expensive: we want sensor nodes to be as simple and cheap as possible and energy consumption is always an issue [BHE00]. Since a transmission over distance $d$ requires $\Omega(d^2)$ energy, transmission over large distances quickly becomes prohibitive.

If we assume a large, sparsely populated area, it may be the case that any transmission by a sensor node cannot be sustained for a long time due to battery limits. We will therefore design a *passive* localisation scheme in the sense that the sensor nodes do not have to do any transmissions in order to localise themselves. The nodes only need to listen passively to their radio and perform a small amount of local computation.

All transmissions come from the base stations. These transmissions will be designed such that a sensor node is able to infer its location after listening for a while.

## 3.1.3. Results

In our model, we start out with a sensor node whose location could be anywhere in the bounded area for which localisation is provided. (For most of the chapter this is a line segment, but in Section 3.8 this is extended to higher dimensions.) We will use transmissions from the base stations to send 'splitlines.' From these splitlines, the sensor nodes are able to infer restrictions on their actual location. Section 3.2 makes the notion of splitline concrete. The main question is then: where should the splitlines be aimed? The rest of this chapter explores this.

In Section 3.3 we analyse what happens when we send splitlines to uniformly random locations. We show that after sending $n$ splitlines for a line segments of length $\ell$, the expected bound on the location of the sensor node is $\Theta(\,\ell/n\,)$. This is, in expectation, as good as any scheme can hope to achieve in the worst case.

In Section 3.4 we briefly look at a variant where we save on the amount of communication. We still achieve an unbiased estimator for the location of the sensor node such that, conditioned on the estimate, the variance of the actual position diminishes as $\Theta(\,\ell/n\,)$.

In Section 3.5 we formalise our concept of a splitline *schedule* and propose a way to compare schedules. Then, in Section 3.6 we define a class of 'regular' schedules and conjecture that a $k$-regular schedule is optimal if and only if $k$ is co-prime to $n$, where $n$ is the length of the schedule. We computationally confirm the conjecture for $n \leqslant 14$, and prove the case $n \ \mathbf{mod}\ k \equiv -1$. In Section 3.7 we give a different set of schedules and prove that those schedules have a performance at most a factor $4 + 4/t$ from optimal after receiving $t$ splitlines.

## 3.2. Model

We will now introduce the model for our range-free localisation scheme. Most of this chapter concerns the 1-dimensional case where we do localisation on a line segment: this most cleanly demonstrates the involved techniques. At the end of the chapter we will make some remarks on how to generalise the results to higher dimensions.

Consider two high-powered base stations, **A** and **B**, and the line segment between them. The base stations have well-synchronised clocks and can broadcast messages. These messages are assumed to propagate at a uniform, constant speed $v$. The sensor nodes we consider all lie on the line segment between **A** and **B**.

Having posited fixed well-synchronised base stations, we will in turn make our sensor nodes very simple and low-powered. Our sensor nodes have no explicit clocks at all. However, they are able to determine in which order incoming messages arrive.

Let $\ell$ be the distance between **A** and **B** and consider a sensor node at arbitrary position $\mathbf{p} \in [0..\ell]$. We will now introduce our main technique. Consider both base stations sending a message at the exact same time, where the message contains the identity of the base station. These messages propagate at a uniform speed of $v$. The sensor node has no clock, but *can* determine which of the messages arrives first. If the node receives **A**'s message first, its location must be in $[0, \ell/2]$; likewise, if it receives **B**'s message first, its location must be in $[\ell/2, \ell]$. For the node, this splits the possible locations at $\ell/2$. Such messages are called *splitline messages*.

The base stations can make the split appear elsewhere by varying the relative time at which they send their messages. Let $t_\mathbf{A}$ and $t_\mathbf{B}$ be the times at which the base stations send their message. To make the split occur at location $s$, the base stations must send at times such that $t_\mathbf{A} + \frac{s}{v} = t_\mathbf{B} + \frac{\ell - s}{v}$. That is, the transmission times must be such that

$$t_\mathbf{B} - t_\mathbf{A} = \frac{2s - \ell}{v}. \tag{3.1}$$

The time difference needed for a change in the splitline location is inversely proportional to the signal propagation speed: taking the derivative for $s$ yields

$$\frac{\partial(t_\mathbf{B} - t_\mathbf{A})}{\partial s} = \frac{2}{v}. \tag{3.2}$$

Plugging in the speed of a radio signal gives a time difference of approximately 7 nanoseconds per meter; speed of sound gives approximately 6 milliseconds per meter. This gives an indication of the precision required of the clocks in the base stations.

Consider the possible collision effect between the splitline messages from **A** and **B**. In our model, a sensor node can determine the order of arrival of messages. Physically, this may be ill defined or complicated in the situation where the second message starts arriving before the first message has fully passed the receiver. It is therefore in our interest to have a *small* splitline message. We can get by with only one bit of information: whether the message comes from **A** or from **B**. But then the sensor node only knows which base station it hears first, not where the split location was aimed (that is, which value for $s$ was used in Equation 3.1).

In some situations, it may be possible to arrange for sensor nodes to already know where the splitline is aimed. More generally, after two actual well-timed single-bit messages (for which timing is crucial) have been sent and received, one of the base stations can send a *location message* which tells the sensor node where the previous splitline was aimed. Note that if every location message is encoded independently, for example in order to not require sensor nodes to keep state, then these messages will need to have size $\Omega(\log|P|)$ where $P$ is the set of possible splitline locations.

Now we look at the effect of timing errors. The base stations use Equation 3.1 to determine times $t_\mathbf{A}$ and $t_\mathbf{B}$. Suppose that due to timing errors (possibly negative) they actually send at times $t_\mathbf{A} + e_\mathbf{A}$ and $t_\mathbf{B} + e_\mathbf{B}$. We plug this back into Equation (3.1), write the resulting location of the splitline as $s + e_s$ and separate the error terms. This gives

$$s + e_s = \frac{\ell + t_\mathbf{B} - t_\mathbf{A}}{2} \cdot v + \frac{e_B - e_A}{2} \cdot v. \tag{3.3}$$

Writing the absolute difference in transmission time errors as $\Delta = |e_B - e_A|$ we get

$$|e_s| = \frac{\Delta}{2} v. \tag{3.4}$$

Here we see that only the difference between the errors matters and that the location error is proportional to signal speed. Again plugging in some numbers, radio signal gives a location error of approximately 15 centimeters per nanosecond difference between the transmission errors, and speed of sound gives a location error of approximately 17 centimeters per millisecond difference. This gives an indication of the synchronisation required of the clocks in the base stations.

Next, we will construct localisation schemes in this model.

## 3.3. Random splitlines

Our first localisation scheme is as follows. At every time step, the base stations coordinate to pick a uniformly random location $s$ in the interval of reals $[0, \ell]$.

Then each station sends a splitline message such that together they aim the splitline at $s$. Then a base station sends a location message to indicate where the splitline was aimed. This is repeated indefinitely. Whenever a sensor node wants to determine its location, it starts listening to the splitline and location messages. It keeps listening until it is satisfied with the precision with which it has determined its position.

We will now analyse the quality of this scheme. Our measure of quality will be the size of the uncertainty interval of a node, that is, the interval of possible locations for a node, compatible with the received splitlines. In the worst case, when all splitlines are aimed at the boundary of the area, the sensor node never learns anything about its location. However, in expectation, the scheme will turn out to work quite well.

Consider a sensor node at position $x$ that has heard a set $S$ of splitlines, with $x \notin S$. We define the *uncertainty* of the node as the sum of the distance to the closest splitline to the left and the closest splitline to the right. This is the length of exactly the interval in which the sensor node could lie, consistent with the received splitlines.

**Definition 3.1 (Uncertainty of a set of splitlines).** For a set $S$ of splitlines and a point $x \notin S$, the *uncertainty* is

$$u(x, S) = \min\{x - s \mid s < x, s \in S\} + \min\{s - x \mid s > x, s \in S\}.$$

Now consider a sensor node at arbitrary position $x$ and a desired upperbound $c$ on the uncertainty (that is: a guarantee of *localisation precision*). The probability that this precision has been achieved after hearing $t$ splitlines approaches $1$ exponentially in $t$.

**Theorem 3.2.** *Consider a position $x$ with $0 < x < \ell$, a localisation precision bound $c > 0$, and a set $S$ of independent, uniformly random splitlines. Then as $|S|$ goes to infinity, $\mathbb{P}[u(x, S) \geqslant c]$ is $\mathcal{O}(d^{|S|})$ for some constant $d < 1$.*

*Proof.* We will upperbound $u(x, S)$ and then show that this bound is larger than $c$ with exponentially vanishing probability. Partition the splitlines in $S$ arbitrarily into pairs (if $|S|$ is odd, ignore a splitline). The probability that a pair of random splitlines, by itself, gives an uncertainty at most $c$ is non-zero. This probability does not depend on $|S|$ and is independent for all the pairs. By amplification with $\lfloor \frac{|S|}{2} \rfloor$ pairs, the probability of failing to achieve the bound vanishes exponentially in $|S|$. $\square$

Next, we define $e(n)$ as the expected uncertainty after hearing $n$ splitlines, with the expectation taken over the position of the node and the location of the splitlines. We assume the sensor node knows it lies between the base stations, so it effectively gets the splitlines '$0$' and '$\ell$' for free.

**Definition 3.3 (Expected uncertainty).** For a uniformly random position $\mathcal{X}$ and a set of $n$ independent, uniformly random splitlines $\mathcal{S}_n$, the expected uncertainty is

$$e(n) \ = \ \mathbb{E}[\ u(\mathcal{X}, \mathcal{S}_n \cup \{0, \ell\})\ ].$$

**Theorem 3.4.** *The expected uncertainty after hearing $n$ splitlines is*

$$e(n) = \frac{2\ell}{n+2}.$$

*Proof.* For convenience, we will scale from $[0, \ell]$ to $[0, 1]$: let $f(n) = e(n)/\ell$. We will derive $f(n)$ and this will give us $e(n)$.

Consider a node at position $x$ that hears $i$ splitlines to its left and $n - i$ splitlines to its right. The expected uncertainty is the sum of the expected uncertainty on the left and on the right. To the left is an area of size $x$ that contains $i$ splitlines. The expected distance to the closest one is $\frac{x}{i+1}$ as it is the first order statistic of $i$ independent, uniformly random variables in $[0, x]$. Similarly, the expected distance to the closest splitline on the right is $\frac{1-x}{n-i+1}$.

The probability that the above case actually occurs is $\binom{n}{i} x^i (1-x)^{n-i}$: of the $n$ nodes, $i$ must actually fall to the left of $x$, and $n - i$ must fall to the right.

We now sum over all possibilities for $i$. Call this sum $g(n, x)$.

$$g(n,x) = \sum_{i=0}^{n} \binom{n}{i} x^i (1-x)^{n-i} \left( \frac{x}{i+1} + \frac{1-x}{n-i+1} \right) \tag{3.5}$$

This is the expected uncertainty for a node at position $x$. We want the expected value for $x$ uniformly from $[0, 1]$, so we integrate $g(n, x)$ over the interval $[0, 1]$:

$$f(n) = \int_0^1 g(n,x)\,dx. \tag{3.6}$$

Standard calculation then shows that $f(n) = \frac{2}{n+2}$. By scaling back we get $e(n) = \frac{2\ell}{n+2}$.

□

From Equation 3.5 we get the expected uncertainty for a node at position $x$. We will now make several observations about this dependence on $x$. Evaluating this at the boundary of the area, that is $x = 0$ or $x = 1$, we get

$$g(n,0) = g(n,1) = \frac{1}{n+1} \tag{3.7}$$

This is easy to see from Equation 3.5: for $x = 0$ the summand equals zero except when $i = 0$, and for $x = 1$ when $i = n$. More directly, observe that this case corresponds to the first order statistic of $n$ independent uniform random variables in $[0, 1]$: on one side the node has no uncertainty since it lies on the boundary, on the other side there must be $n$ splitlines. This order statistic indeed

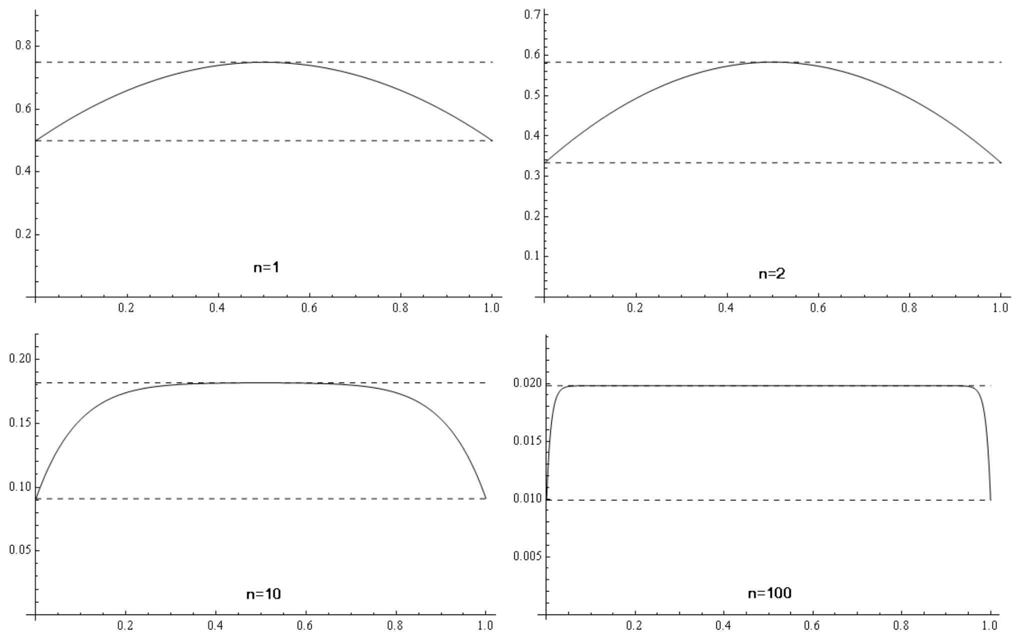


**Figure 3.1.** Plot of $g(x,n)$ for various values of $n$, with minimum (attained at $x = 0$ and $x = 1$) and maximum (attained at $x = \frac{1}{2}$) indicated. Beware the very different scale on the $y$-axis.

has value $\frac{1}{n+1}$. Note that this expected uncertainty is lower than when $x$ is uniformly random.

Evaluating Equation 3.5 for $x = \frac{1}{2}$ gives

$$g(n, \frac{1}{2}) = \frac{2 - 2^{-n}}{n+1}. \tag{3.8}$$

Comparing Equations (3.7) and (3.8) shows that the expected uncertainty is lower at the boundary of the area than it is in the middle. This is reasonable, since a node in the middle needs two nearby splitlines to get a good bound, whereas for a node close to the boundary a single nearby splitline suffices. The ratio between these high and low expected values approaches 2. For high $n$, the beneficial effect of being near the boundary does get confined to a smaller area near $x = 0$ and $x = 1$. See Figure 3.1, which plots the dependence on $x$ for various $n$. Figure 3.2 shows the dependence on $n$ for various $x$.

## 3.4. Random splitlines without location messages

We now look at a different way of estimating positions using random splitlines. In this section, we will *not* use location messages. An advantage of this is that it requires less communication: location messages require $\mathcal{O}(\log p)$ bits per round when localising up to $1/p$ precision. Also, without location messages we may

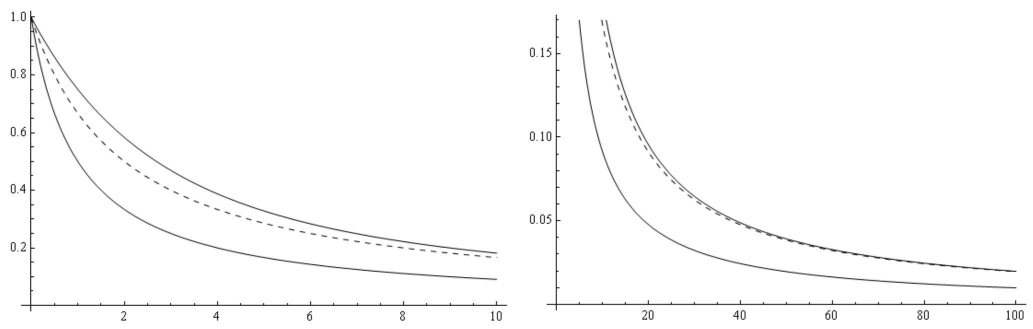


**Figure 3.2.**
Dashed line: expected uncertainty for uniformly random position, $f(n) = \frac{2}{n+2}$.
Top solid line: expected uncertainty at middle, $g(\frac{1}{2}, n) = \frac{2-2^{-n}}{n+1}$.
Bottom solid line: expected uncertainty at boundary, $g(0, n) = \frac{1}{n+1}$.
Left image: close-up for small $n$. Right image: larger $n$.

be able to send splitline messages at a higher rate since we don't have to pause for the location messages in-between.

| | |
|---|---|
| $\mathcal{P}$ | Position of the sensor node. Uniformly random on $(0, 1)$. |
| $\mathcal{S}$ | Positions of the splitlines. Set of $n$ independent random variables uniformly in $[0, 1]$, independent of $\mathcal{P}$. |
| $\mathcal{K}$ | Number of splitlines left of the sensor node. $\lvert\{s \in \mathcal{S} \mid s < \mathcal{P}\}\rvert$ |
| $\mathcal{L}$ | Position of closest splitline to the left of the sensor node, or 0 if there is none. $max\{s \in \mathcal{S} \cup \{0\} \mid s < \mathcal{P}\}$ |
| $\mathcal{R}$ | Position of closest splitline to the right of the sensor node, or 1 if there is none. $min\{s \in \mathcal{S} \cup \{1\} \mid s > \mathcal{P}\}$ |

**Table 3.1.** Random variables used in Section 3.4.

In the analysis we use the random variables defined in Table 3.1. Using this notation, the purpose of the location messages is to be able to observe $\mathcal{L}$ and $\mathcal{R}$: the sensor node knows where the splitlines were aimed. This gives hard bounds on $\mathcal{P}$ and in the previous section we derived the expected uncertainty, which in the new notation is $\mathbb{E}[\mathcal{R} - \mathcal{L}]$. If we do not send location messages, then $\mathcal{L}$ and $\mathcal{R}$ cannot be observed. Yet on the basis of the splitline messages alone, a sensor node can still observe $\mathcal{K}$: for every splitline it sees whether it falls to the left or

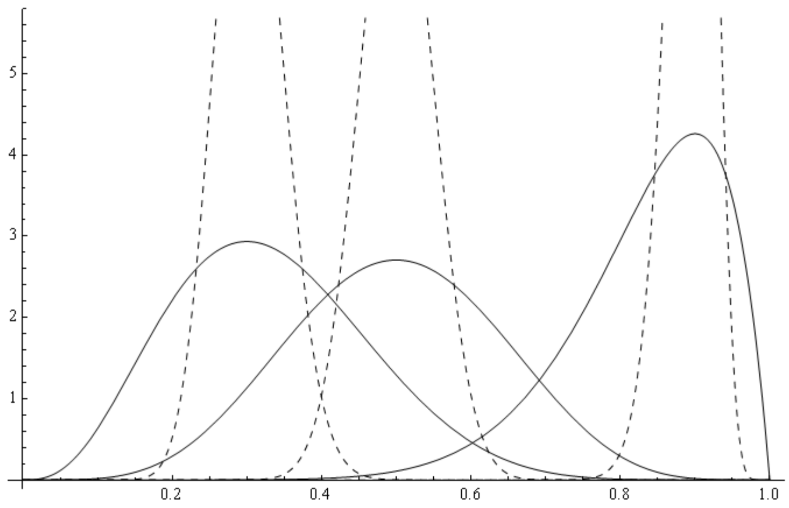


**Figure 3.3.** Plot of probability density function $f(p)$ for $\hat{p} \in \{0.3, 0.5, 0.9\}$. The solid line is for $n = 10$, dashed line is $n = 100$.

to the right.

In Section 3.4.1 we give an estimator for $\mathcal{P}$, based on observing only $\mathcal{K}$, and we analyse the quality of this estimator. Then in Section 3.4.2 we compare this to a similar estimator that does observe $\mathcal{L}$ and $\mathcal{R}$. This allows us to comment on the usefulness of location messages.

### 3.4.1. Estimator without location messages

The splitline locations are independent and uniformly random in $[0, 1]$. Then conditioned on an observation of $\mathcal{P}$, we have that $\mathcal{K}$ is binomially distributed with success probability $p$. Furthermore it is then well known that, observing $\mathcal{K} = k$, the maximum likelihood estimate for $\mathcal{P}$ is $\hat{p} = k/n$ and that this estimator is unbiased.

To evaluate the quality of this estimator, consider the probability distribution over the true position of the sensor, given estimate $\hat{p}$. Then $k$ splitlines must fall left of $p$ and $n - k$ must fall to the right. For $0 \leqslant p \leqslant 1$, this gives the following probability density function $f$ (where $\hat{p}$ and $n$ remain free variables):

$$\mathbb{P}[\, \mathcal{P} = p \mid \mathcal{K}/n = \hat{p} \,] \overset{\text{def}}{=} f(p) = \frac{p^{\hat{p}n}(1-p)^{(1-\hat{p})n}}{\int_0^1 r^{\hat{p}n}(1-r)^{(1-\hat{p})n}\, dr}. \tag{3.9}$$

Note that the denominator is just a normalisation term. This probability density function is illustrated for several $\hat{p}$ and $n$ in Figure 3.3.

**Lemma 3.5.** *The expected true position of the sensor node, conditioned on estimate $\hat{p}$, is*

$$\mathbb{E}[\, \mathcal{P} \mid \mathcal{K}/n = \hat{p} \,] = \frac{n\hat{p} + 1}{n + 2}. \tag{3.10}$$

*Proof.* By definition of expected value, we have the following.

$$\mathbb{E}[\ \mathcal{P} \mid \mathcal{K}/n = \hat{p}\ ] \ = \ \int_0^1 p \cdot f(p)\, dp \tag{3.11}$$

Then the lemma follows from standard calculation. □

For $\hat{p} = \frac{1}{2}$, the expectation evaluates to $\frac{1}{2}$ like we want. For other values of $\hat{p}$ the unbiasedness is asymptotic: for example, $\hat{p} = 0$ evaluates to $\frac{1}{n+2}$.

Finally, we look at the variance of actual sensor position, conditioned on estimate $\hat{p}$. As a function of $n$, the variance tells us how quickly we gain confidence in the estimate.

**Lemma 3.6.** *For constant* $\hat{p}$, $\mathbb{V}\text{ar}[\ \mathcal{P} \mid \mathcal{K}/n = \hat{p}\ ]$ *diminishes as* $\Theta(\ n^{-1}\ )$.

*Proof.* Calculate the variance as 'expected square minus square expected' and then simplify:

$$\begin{aligned}
&\mathbb{V}\text{ar}[\ \mathcal{P} \mid \mathcal{K}/n = \hat{p}\ ] \\
&= \left(\int_0^1 p^2 \cdot f(p)\ dp\right) - \left(\int_0^1 p \cdot f(p)\ dp\right)^2 \\
&= \frac{\hat{p}(1-\hat{p})n^2 + n + 1}{n^3 + 7n^2 + 16n + 12}.
\end{aligned} \tag{3.12}$$

The lemma follows, as with constant $\hat{p}$ we have $\Theta(\ n^2\ )/\Theta(\ n^3\ )$, which is $\Theta(\ n^{-1}\ )$. □

## 3.4.2. Estimator with location messages

For comparison we now look at a different estimator: this one based on observing $\mathcal{L}$ and $\mathcal{R}$ in addition to $\mathcal{K}$. This will allow us to evaluate the value of location messages. Our estimate is $\hat{p} = \frac{\mathcal{L}+\mathcal{R}}{2}$. Again, we look at the probability density over the true position of the sensor given the estimate. Call the observed event $\mathcal{O}$.

$$\mathcal{O} \overset{\text{def}}{=} [\hat{p} = \tfrac{\mathcal{L}+\mathcal{R}}{2} \wedge \mathcal{K} = k] \ = \ [\mathcal{L} + \mathcal{R} = 2\hat{p} \wedge \mathcal{K} = k] \tag{3.13}$$

**Theorem 3.7.** *The probability density of true sensor position* $\mathbb{P}[\mathcal{P} = p \mid \mathcal{O}]$

$$= \ c \cdot \sum_{k=0}^{n} \left(\text{Binom}(k; n, p) \int_0^p \text{Beta}(\frac{\ell}{p}; k, 1)\ \text{Beta}(\frac{2\hat{p} - \ell - p}{1-p}; 1, n-k)\, d\ell\right)$$

*with* $c$ *some normalisation constant.*

*Proof.* We make a derivation starting off with the probability density we are interested in.

$$\mathbb{P}[\mathcal{P} = p \mid \mathcal{O}] \tag{3.14}$$

Applying Bayes' theorem, this is equal to:

$$\frac{\mathbb{P}[\mathcal{O} \mid \mathcal{P} = p]\ \mathbb{P}[\mathcal{P} = p]}{\mathbb{P}[\mathcal{O}]} \tag{3.15}$$

Now a lot of cancellations occur. Notice that $\mathbb{P}[\mathcal{P} = p] = 1$ over the appropriate support ($\mathcal{P}$ is uniform on $(0, 1)$). This is equal to:

$$\frac{\mathbb{P}[\mathcal{O} \mid \mathcal{P} = p]}{\mathbb{P}[\mathcal{O}]} \tag{3.16}$$

Next we condition the denominator on $\mathcal{P}$ to arrive at (3.17):

$$\frac{\mathbb{P}[\mathcal{O} \mid \mathcal{P} = p]}{\int_0^1 \mathbb{P}[\mathcal{P} = p]\ \mathbb{P}[\mathcal{O} \mid \mathcal{P} = p]\, dp} \tag{3.17}$$

Again, we use $\mathbb{P}[\mathcal{P} = p] = 1$ to get (3.18). Notice that the denominator just acts as a normalisation term. We obtain:

$$\frac{\mathbb{P}[\mathcal{O} \mid \mathcal{P} = p]}{\int_0^1 \mathbb{P}[\mathcal{O} \mid \mathcal{P} = p]\, dp} \tag{3.18}$$

We continue with $c$ denoting this normalisation constant, to arrive at:

$$c \cdot \mathbb{P}[\mathcal{O} \mid \mathcal{P} = p] \tag{3.19}$$

We expand the definition of $\mathcal{O}$ and condition on $\mathcal{K}$:

$$c \cdot \mathbb{P}[\mathcal{L} + \mathcal{R} = 2\hat{p} \wedge \mathcal{K} = k \mid \mathcal{P} = p] \tag{3.20}$$

$$= c \cdot \sum_{k=0}^{n} \Big( \mathbb{P}[\mathcal{K} = k \mid \mathcal{P} = p]\ \mathbb{P}[\mathcal{L} + \mathcal{R} = 2\hat{p} \mid \mathcal{P} = p \wedge \mathcal{K} = k] \Big) \tag{3.21}$$

To keep the formula manageable, let $\mathcal{O}_2 = [\mathcal{P} = p \wedge \mathcal{K} = k]$, leading to:

$$c \cdot \sum_{k=0}^{n} \Big( \mathbb{P}[\mathcal{K} = k \mid \mathcal{P} = p]\ \mathbb{P}[\mathcal{L} + \mathcal{R} = 2\hat{p} \mid \mathcal{O}_2] \Big) \tag{3.22}$$

In (3.23) we have conditioned on $\mathcal{L}$. It suffices to integrate only up to $p$: since by definition $\mathcal{L}$ cannot be greater than $\mathcal{P}$, the integrand is 0 when $\ell > p$. This gives:

$$c \cdot \sum_{k=0}^{n} \Big( \mathbb{P}[\mathcal{K} = k \mid \mathcal{P} = p] \int_0^p \mathbb{P}[\mathcal{L} = \ell \mid \mathcal{O}_2]\ \mathbb{P}[\ell + \mathcal{R} = 2\hat{p} \mid \mathcal{L} = \ell \wedge \mathcal{O}_2]\, d\ell \Big) \tag{3.23}$$

We rearrange terms and drop $\mathcal{L} = \ell$ from the conditions of the rightmost probability density. This is valid because $\mathcal{L}$ and $\mathcal{R}$ are independent when conditioned on $\mathcal{P}$ and $\mathcal{K}$. This leads to:

$$c \cdot \sum_{k=0}^{n} \left( \mathbb{P}[\mathcal{K} = k \mid \mathcal{P} = p] \int_0^p \mathbb{P}[\mathcal{L} = \ell \mid \mathcal{O}_2] \, \mathbb{P}[\mathcal{R} = 2\hat{p} - \ell \mid \mathcal{O}_2] \, d\ell \right) \tag{3.24}$$

Now we are in a position to recognise some distributions. As we mentioned earlier, $\mathcal{K}$ conditioned on $\mathcal{P}$ is binomially distributed. Recall that $\mathcal{O}_2 = [\mathcal{P} = p \wedge \mathcal{K} = k]$. Conditioned on this, $\mathcal{L}$ is beta distributed: $\mathcal{L}$ is the largest of $k$ points uniformly in $[0, p]$. Furthermore, $\mathcal{R}$ is the smallest of $n - k$ points uniformly in $[p, 1]$. Then $\mathcal{R} - p$ is the smallest of $n - k$ points uniformly in $[0, 1 - p]$ and also beta distributed. This finishes the proof:

$$c \cdot \sum_{k=0}^{n} \left( \text{Binom}(k; n, p) \int_0^p \text{Beta}(\frac{\ell}{p}; k, 1) \, \text{Beta}(\frac{2\hat{p} - \ell - p}{1 - p}; 1, n - k) \, d\ell \right). \tag{3.25}$$

□

The probability density in Theorem 3.7 (that is, actual position conditioned on estimated position) is illustrated for $n = 4$ as the solid line in Figure 3.4, which was plotted by numerically evaluating Equation (3.7). To show the value of the location messages, we also plot the corresponding probability density for the estimator without location messages (from Section 3.4.1). We argued before that omitting location messages might allow for a higher transmission rate. Therefore we also show the probability density where the latter estimator gets more splitlines (since those can be sent at a higher rate). We plot this for speedup factors 1, 4 and 16. It can be observed that with a speedup of factor 4, the distributions look somewhat similar.

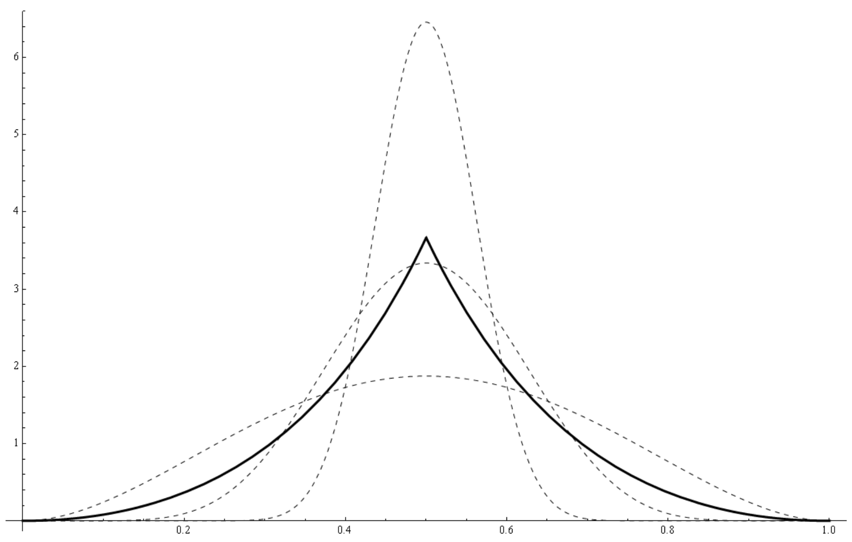
We can notice something further in Figure 3.4. With location messages and $\hat{p} = \frac{1}{4}$ we see that the support of the actual position is only $(0, \frac{1}{2})$. This is because observing $\hat{p} = \frac{\mathcal{L}+\mathcal{R}}{2}$ gives a hard bound on $p$: since $\mathcal{L} > 0$ and $\mathcal{R} > p$, we have that $p < 2\hat{p}$.

## 3.5. Splitline schedules

Sending random splitlines does not give worst case guarantees, even though it works well on expectation. We will now design deterministic schedules of splitlines which also 'typically' work well, but for which, in addition, we can give good worst case guarantees.

Our aim is to localise a sensor node to a precision of 1 on a line segment of length $\ell + 1$. We choose to only aim splitlines at integer positions. This gives location messages a size of $\mathcal{O}(\log \ell)$.

**Definition 3.8 (Splitline schedule).** A splitline schedule for a line segment of length $\ell = n + 1$ is a permutation of the integers $1 \dots n$, that is, a permutation



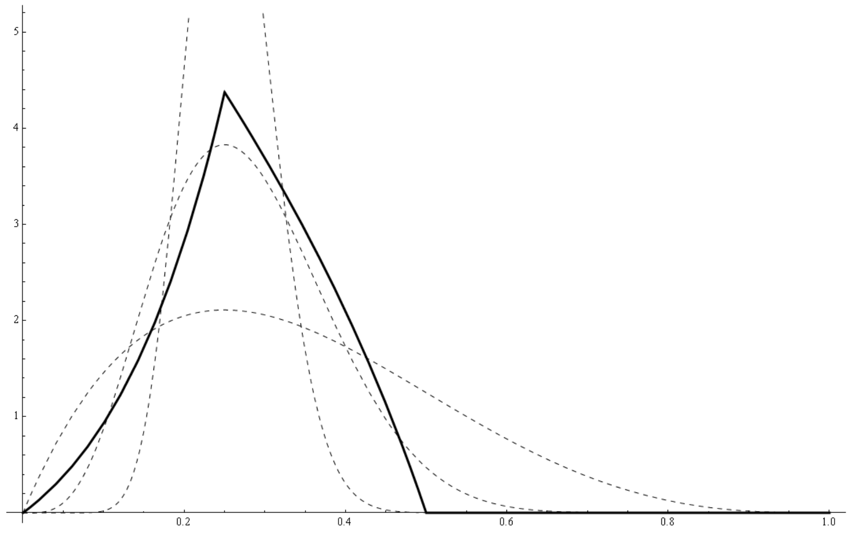


**Figure 3.4.** This figure is for $n = 4$; the top figure with $\hat{p} = 1/2$ and the bottom figure $\hat{p} = 1/4$. The solid line is for the estimator with location messages and gives the probability density $\mathbb{P}[\mathcal{P}|\mathcal{L} + \mathcal{R} = 2\hat{p}]$. The dashed lines are for the estimator without location messages and give the probability density $\mathbb{P}[\mathcal{P}|\mathcal{K}/n = \hat{p}]$ with a speedup of 1, 4 and 16.

of the integer positions on the interior of the line segment. We write schedules using round brackets.

The base stations will aim splitlines at all integer positions on the line segment in the order indicated by the schedule. This schedule is repeated over and over. The time it takes for the schedule to complete ($n$ rounds of splitline/location messages) may be long. We therefore want to allow for sensor nodes to start listening at any point in the schedule.

We will now formalise a measure of quality for splitline schedules.

**Definition 3.9 (Uncertainty of a schedule).** Consider a sensor node at real position $p$. It starts listening to the schedule at offset $t_0$ and then hears $t$ splitlines. The uncertainty $u(p, t_0, t)$ is the size of the interval of possible locations for the node, as constrained by the observed splitlines.

To analyse the quality of a certain schedule, define the worst case behavior.

**Definition 3.10 (Worst case uncertainty).** The worst case uncertainty is

$$w(t) = \max_{p,t_0} u(p, t_0, t).$$

That is, $w(t)$ is the uncertainty after hearing $t$ splitlines, the worst case taken over all possibilities for where the node is, and when it starts listening. We will now make some initial observations.

The worst case uncertainty of a schedule is unaffected by rotation or reversal (of the underlying permutation). We will therefore no longer consider all permutations on $n$ elements, but only the relevant equivalence classes.

**Definition 3.11 ($\mathcal{P}_n$).** Consider the equivalence relation where two permutation are equivalent if they are equal up to rotation and reversal. For the permutations on $n$ elements, call the resulting set of equivalence classes $\mathcal{P}_n$.

Note that $w(t)$ is non-increasing. The value of $w(t)$ depends on the schedule at only a limited number of times. Before having heard any splitlines, the node does not know where it is. Therefore, any schedule has $w(0) = \ell$. Next, notice that always $w(1) = \ell - 1$: due to taking the worst case over starting time $t_0$ and position $p$, the first splitline heard can be at position 1 while $p > 1$.

For $t \geqslant n$ we have that $w(t) = 1$: having heard all distinct splitlines, the node knows its location to precision 1 and never learns more. Furthermore, $w(n-1) = 2$: only one splitline remains unheard and, taking the worst case over position, the node can lie in the remaining interval of size 2.

From the preceding observations we have that $w(t)$ depends on the schedule only for $t \in [2 \ldots n-2]$. For those values of $t$, we also have $2 \leqslant w(t) \leqslant \ell - 2$. In this way, the quality of a schedule from $\mathcal{P}_n$ is expressed as an $(n-3)$-tuple of integers $< n$. We call this the schedule's *uncertainty tuple*. We write uncertainty tuples using square brackets.

The value of $w(t)$ is always high at early $t$. Conversely, having $w(t) = 1$ requires high $t$. In fact, there is the following lower bound.

**Lemma 3.12.** *For any splitline schedule,* $w(t) \geqslant \frac{\ell}{t+1}$.

*Proof.* The definition of $w(t) = \max_{p,t_0} u(p, t_0, t)$ includes taking the worst case of starting time $t_0$. For this bound we only consider a schedule from a specific starting point: we fix $t_0 = 0$. Trivially $\max_{p,t_0} u(p, t_0, t) \geqslant \max_p u(p, 0, t)$. The best possible schedule, for $t_0 = 0$ and *this* value of $t$, evenly spaces the splitlines along the line segment. This gives a value of $\frac{\ell}{t+1}$. Now we have a lower bound for $w(t)$: even for fixed $t_0$, and specifically for this $t$, no schedule can do better. □

Consider the schedule given by the identity permutation. This is a very bad schedule: it has $w(t) = \ell - t$, for $t < \ell$. (Consider $t_0 = 0$ and $p = n$.) There are schedules that do much better, and in fact come close to the lowerbound from Lemma 3.12. But how exactly do we compare the quality of schedules? Consider the following two schedules for $\ell = 8$ (that is, $n = 7$):

$$(1, 4, 7, 3, 6, 2, 5) \text{ and } (1, 3, 5, 7, 2, 4, 6).$$

These schedules give rise to the following two uncertainty tuples. It is unclear which one should be considered to have higher quality.

$$[4, 3, 3, 2] \overset{?}{\leqslant} [5, 3, 2, 2]. \tag{3.26}$$

Notice that in position 1 the left tuple is strictly better. However, in position 3 the right tuple is strictly better. This leads us to prefer either one or the other depending on the time; neither is strictly better than the other. Accordingly, we define the following partial order, in which the above tuples are incomparable.

**Definition 3.13 ($\leqslant$ on tuples).** For tuples $A$ and $B$, let $A \leqslant B$ if and only if $A[i] \leqslant B[i]$ for all $i$.

**Definition 3.14 (Optimal schedule).** A schedule is *optimal* (or: minimal) if, over all of $\mathcal{P}_n$, its uncertainty tuple is minimal according to $\leqslant$.

There is not necessarily a single least element in this order; for example, the two schedules given above are (all) the optimal schedules for $n = 7$.

By exhaustive search we have determined that the number of minimal solutions for small $n$ is 1, 1, 1, 1, 1, 1, 2, 1, 5, 5, 5, 7, 5, 20. These are listed in Table 3.2

.

## 3.6. Regular schedules

We will now look at an interesting class of splitline schedules.

**Definition 3.15 (Regular schedule).** A schedule is called $k$-*regular* if and only if its equivalence class in $\mathcal{P}_n$ contains the schedule $A$ with $A[i] = (ik \ \mathbf{mod}\ n) + 1$.

**Proposition 3.16.** *There exist $k$-regular schedules of length $n$ if and only if $k$ and $n$ are relatively prime; that is, if and only if* $\gcd(k, n) = 1$.

Note that, because of equivalence under reversal and rotation, a schedule is $k$-regular if and only if it is $(n-k)$-regular. We conjecture that these schedules are

| $n$ | Schedule (hexadecimal) | $w(t)$ for $t = 1 \ldots n$ | Comments |
|---|---|---|---|
| 1 | 1 | 1 | 1-regular |
| 2 | 1 2 | 2 1 | 1-regular |
| 3 | 1 2 3 | 3 2 1 | 1-regular |
| 4 | 1 2 3 4 | 4 3 2 1 | 1-regular |
| 5 | 1 3 5 2 4 | 5 3 2 2 1 | 2-regular ⋆ |
| 6 | 1 3 5 2 6 4 | 6 4 3 2 2 1 | |
| 7 | 1 3 5 7 2 4 6 | 7 5 3 2 2 2 1 | 2-regular ⋆ |
| | 1 4 7 3 6 2 5 | 7 4 3 3 2 2 1 | 3-regular |
| 8 | 1 4 7 2 5 8 3 6 | 8 5 3 3 2 2 2 1 | 3-regular ⋆ |
| 9 | 1 3 5 7 9 2 4 6 8 | 9 7 5 3 2 2 2 2 1 | 2-regular ⋆ |
| | 1 4 7 3 6 9 2 5 8 | 9 7 4 3 3 3 2 2 1 | |
| | 1 4 6 8 2 5 9 3 7 | 9 6 4 4 3 2 2 2 1 | |
| | 1 4 6 9 3 8 5 2 7 | 9 6 5 3 3 3 2 2 1 | |
| | 1 5 9 4 8 3 7 2 6 | 9 5 4 4 3 3 2 2 1 | 4-regular |
| 10 | 1 3 8 6 A 4 2 7 9 5 | 10 8 6 4 3 2 2 2 2 1 | |
| | 1 6 4 9 2 7 5 A 3 8 | 10 7 5 5 3 2 2 2 2 1 | |
| | 1 4 7 A 3 6 9 2 5 8 | 10 7 4 3 3 3 2 2 2 1 | 3-regular |
| | 1 5 8 2 6 A 4 9 3 7 | 10 6 5 5 3 3 3 2 2 1 | |
| | 1 5 9 4 A 6 2 8 3 7 | 10 6 5 4 4 3 3 2 2 1 | |
| 11 | 1 3 5 7 9 B 2 4 6 8 A | 11 9 7 5 3 2 2 2 2 2 1 | 2-regular ⋆ |
| | 1 4 7 A 2 5 8 B 3 6 9 | 11 8 5 3 3 3 2 2 2 2 1 | 3-regular ⋆ |
| | 1 5 9 2 6 A 3 7 B 4 8 | 11 7 4 4 3 3 3 2 2 2 1 | 4-regular ⋆ |
| | 1 6 B 4 9 2 7 5 A 3 8 | 11 7 5 5 3 3 2 2 2 2 1 | |
| | 1 6 B 5 A 4 9 3 8 2 7 | 11 6 5 5 4 4 3 3 2 2 1 | 5-regular |
| 12 | 1 4 A 8 2 6 C 9 3 5 B 7 | 12 9 6 4 4 4 3 2 2 2 2 1 | |
| | 1 4 B 8 5 2 9 C 6 3 A 7 | 12 9 7 4 3 3 3 3 2 2 2 1 | |
| | 1 5 9 4 8 C 3 7 B 2 6 A | 12 9 5 4 4 4 3 3 3 2 2 1 | |
| | 1 5 9 3 C 6 A 2 8 4 B 7 | 12 9 6 4 4 3 3 3 2 2 2 1 | |
| | 1 5 9 4 C 8 2 6 A 3 B 7 | 12 8 7 4 4 3 3 3 3 2 2 1 | |
| | 1 7 3 B 8 4 C 6 A 2 5 9 | 12 8 6 4 4 4 3 3 2 2 2 1 | |
| | 1 6 B 4 9 2 7 C 5 A 3 8 | 12 7 5 5 3 3 2 2 2 2 2 1 | 5-regular |
| 13 | 1 3 5 7 9 B D 2 4 6 8 A C | 13 11 9 7 5 3 2 2 2 2 2 2 1 | 2-regular ⋆ |
| | 1 4 7 A D 3 6 9 C 2 5 8 B | 13 10 7 4 3 3 3 3 2 2 2 2 1 | 3-regular |
| | 1 5 9 D 4 8 C 3 7 B 2 6 A | 13 9 5 4 4 4 3 3 3 2 2 2 1 | 4-regular |
| | 1 6 B 3 8 D 5 A 2 7 C 4 9 | 13 8 5 5 3 3 3 2 2 2 2 2 1 | 5-regular |
| | 1 7 D 6 C 5 B 4 A 3 9 2 8 | 13 7 6 6 5 5 4 4 3 3 2 2 1 | 6-regular |
| 14 | 1 4 7 A D 2 5 8 B E 3 6 9 C | 14 11 8 5 3 3 3 3 2 2 2 2 2 1 | 3-regular ⋆ |
| | 1 4 9 E 7 2 C 5 A 8 3 D 6 B | 14 11 7 5 5 5 3 3 2 2 2 2 2 1 | |
| | 1 5 3 A C 8 E 6 2 4 B 9 D 7 | 14 11 10 8 6 4 3 2 2 2 2 2 2 1 | |
| | 1 5 9 D 2 6 A E 3 7 B 4 8 C | 14 11 7 4 4 4 3 3 3 3 2 2 2 1 | |
| | 1 5 A E 8 4 C 2 9 6 D 3 B 7 | 14 10 8 5 4 4 4 3 3 2 2 2 2 1 | |
| | 1 5 A E 4 8 C 2 6 9 D 3 7 B | 14 10 6 5 5 5 3 3 3 3 2 2 2 1 | |
| | 1 6 3 C 9 E 5 8 2 B 4 D 7 A | 14 9 9 7 5 3 3 3 3 3 2 2 2 1 | |
| | 1 6 9 E 4 C 7 2 A 5 D 8 3 B | 14 10 8 5 5 4 3 3 2 2 2 2 2 1 | |
| | 1 6 B 4 9 E 2 7 C 5 A 3 8 D | 14 12 7 5 5 3 3 3 2 2 2 2 2 1 | |
| | 1 6 B 4 D 8 2 A 5 E 7 C 3 9 | 14 9 7 6 5 4 3 3 2 2 2 2 2 1 | |
| | 1 6 B 4 E 9 2 7 C 5 A 3 D 8 | 14 10 7 5 5 3 3 3 3 2 2 2 2 1 | |
| | 1 6 B 2 7 C 3 8 D 4 9 E 5 A | 14 9 5 5 4 4 4 3 3 3 2 2 2 1 | 5-regular ⋆ |
| | 1 7 4 C A 2 6 E 8 B 3 5 D 9 | 14 10 8 6 4 4 4 3 2 2 2 2 2 1 | |
| | 1 7 B 4 D 6 A 2 8 E 5 C 3 9 | 14 9 7 7 4 4 3 3 3 2 2 2 2 1 | |
| | 1 7 C 4 A 2 8 E 6 D 5 B 3 9 | 14 8 7 7 5 5 3 3 3 3 3 2 2 1 | |
| | 1 7 D 4 A 6 C 2 8 E 5 B 3 9 | 14 10 6 6 4 4 3 3 3 2 2 2 2 1 | |
| | 1 7 D 4 A 2 8 E 5 B 6 C 3 9 | 14 9 6 6 5 5 3 3 3 3 3 2 2 1 | |
| | 1 7 D 5 B 4 C 6 E 8 2 A 3 9 | 14 8 7 6 6 5 5 4 4 3 3 2 2 1 | |
| | 1 8 5 C 2 9 6 D 3 A 7 E 4 B | 14 10 7 7 4 3 3 3 3 2 2 2 2 1 | |
| | 1 8 6 D 4 B 2 9 7 E 5 C 3 A | 14 9 7 7 5 5 3 2 2 2 2 2 2 1 | |

**Table 3.2.** Table of all optimal schedules for $n \leqslant 14$, computed by exhaustive search. Precisely the regular schedules predicted by Conjecture 1 are present. The cases covered by Theorem 3.28 are marked with $\star$.

optimal (with the exception of $k = 1$, which, as we noted before, is a very bad schedule).

**Conjecture 1.** *All $k$-regular schedules are optimal except for $k = 1$.*

**Lemma 3.17.** *All $k$-regular schedules with $n \leqslant 14$ are optimal except for $k = 1$.*

*Proof.* Check Table 3.2, which we have computed by exhaustive search. □

We will now prove the conjecture for case where $n \textbf{ mod } k \equiv -1$. We will start by introducing the method of analysis.

**Definition 3.18 (Chunk).** Let $A = (a_0, a_1, \dots, a_{n-1})$ be a schedule. For $0 \leqslant t \leqslant n$, a $t$-*chunk* of $A$ is an unordered tuple of $t$ consecutive elements from $A$, where the first element is considered consecutive to the $n^{th}$. We write chunks using angle brackets. That is, a $t$-chunk is $\langle a_{i+1}, a_{i+2}, \dots, a_{i+t}\rangle$, for some $i$, where indices are taken mod $n$.

*Example 3.19.* The 3-chunks of $(1,3,5,2,4)$ are $\langle 1,3,5\rangle$, $\langle 3,5,2\rangle$, $\langle 5,2,4\rangle$, $\langle 2,4,1\rangle$ and $\langle 4,1,3\rangle$.

If $1 < t < n$, there are $n$ distinct $t$-chunks for a given $A \in \mathcal{P}_n$. Define $\mathrm{chop}(A,t)$ as the function that maps a schedule $A$ to the set of its $t$-chunks. Given the set $\mathrm{chop}(A,t)$, we can *reconstruct* $A \in \mathcal{P}_n$ up to rotation and reflection. (As noted before, we do not care about rotation and reflection, as these give equivalent schedules.)

*Example 3.20.* If the order in the chunks were preserved, reconstruction would be easy. For example, the 3-chunks $\langle 1,4,5\rangle$, $\langle 4,5,2\rangle$, $\langle 5,2,3\rangle$, $\langle 2,3,1\rangle$ and $\langle 3,1,4\rangle$ reconstruct uniquely to the (equivalence class of) permutation $(1,4,5,2,3)$.

*Example 3.21.* In general, the order in chunks is not preserved. Still reconstruction is possible. For example, the 3-chunks $\langle 1,2,3\rangle$, $\langle 1,3,4\rangle$, $\langle 1,4,5\rangle$, $\langle 2,3,5\rangle$ and $\langle 2,4,5\rangle$ reconstruct uniquely to the schedule $(1,4,5,2,3)$.

To see this, consider the chunks that contain a 1. There must be $t = 3$ such chunks. Among the other numbers in those chunks are: twice a 3 and twice a 4. In the schedule, those must be the neighbours of 1. In particular, on the side of the 3, there must next be 2: there is a chunk $\langle 1,2,3\rangle$. Likewise on the side of the 4, there must be 5: there is a chunk $\langle 1,4,5\rangle$. This process can be repeated until the entire schedule has been reconstructed.

Let $C$ be a set of $n$ $t$-chunks. $C$ may or may not correspond to a schedule. Still it is possible to do reconstruction in the following sense: given $C$, produce the unique $A \in \mathcal{P}_n$ satisfying $\mathrm{chop}(A,t) = C$ if it exists, and No otherwise.

We now give some necessary conditions for $C$ to reconstruct to a schedule.

**Lemma 3.22.** *Let $C = \mathrm{chop}(A,t)$, for some $t$ with $0 < t < n$. Then, for every integer $1 \leqslant i \leqslant n$, exactly $t$ tuples in $C$ contain the number $i$. Also, no two chunks in $C$ are equal.*

*Proof.* Consider the schedule $A$: a permutation of $1..n$. A chunk is a tuple of $t$ consecutive elements in $A$. Therefore, every element in $A$ is contained in exactly $t$ different chunks. All these chunks are different even if the order is disregarded. □

**Definition 3.23 (Gap).** The *gap* of a chunk $c$ as the biggest interval left in $[0, n+1]$ when it is split at the locations in $c$.

*Example 3.24.* With $n = 7$, the gap of $\langle 2, 4\rangle$ is 4 because of the interval 4..8. With $n = 7$, the gap of $\langle 1, 4, 6\rangle$ is 3 because of the gap 1..4.

Now we consider the relation between values $w(t)$ of a schedule and its chunks. Applying the definition of $w$ and $\mathrm{chop}$ directly gives that, for any schedule $A$ and $1 \leqslant t \leqslant n$, we have that all chunks in $\mathrm{chop}(A, t)$ have gap at most $w(t)$. More specifically, $w(t)$ equals the maximum gap of any chunk in $\mathrm{chop}(A, t)$. This leads an observation that is crucial in the following proofs.

**Lemma 3.25.** *A schedule cannot contain $t$-chunks with gap larger than $w(t)$.*

*Proof.* Suppose to the contrary that the schedule has a $t$-chunk of gap $g > w(t)$. Then there exists a starting time $t_0^*$ such that exactly these $t$ splitlines are heard first. Since the chunk has gap $g$, there exists a $p^*$ such that $u(p^*, t_0^*, t) = g$. But now $u(p^*, t_0^*, t) > w(t)$, which contradicts $w(t) = \max_{p,t_0} u(p, t_0, t)$. □

We are now ready to prove some cases of Conjecture 1. We start with a simple example and then extend it.

**Theorem 3.26.** *If $n$ is odd, then a 2-regular schedule exists and any such schedule is optimal.*

*Proof.* First of all, $n$ is relatively prime to 2 if and only if $n$ is odd, therefore the schedule exists.

To show that the schedule is optimal, consider the time $t^* = \lceil n/2 \rceil$. We will show that the 2-regular schedule is the unique schedule with $w(t^*) = 2$. Then it must be optimal, since this shows that all other schedules have $w(t^*) > 2$ and are therefore not smaller in the partial order $\leqslant$.

Suppose $A$ is a schedule with $w(t^*) = 2$ and let $C = \mathrm{chop}(A, t^*)$. Recall that $|C| = n$. We want $w(t^*) = 2$, so by Lemma 3.25 all chunks in $C$ must have gap $\leqslant 2$. By Lemma 3.22, the number of chunks in $C$ containing a 1 must be exactly $t^*$. There are only $t^*$ possible chunks satisfying both these conditions: the chunk must cover the interval $[0..n+1]$ using the number 1 and $t^* - 1$ more integers, while leaving a gap of at most 2. This can only be achieved by chunks consisting of steps of 2, except for one step of size 1. This can be done in exactly $t^*$ ways, showing that in fact all these chunks must be in $C$.

Identical reasoning, for the number $n$ instead of 1, also gives $t^*$ chunks that must also be in $C$. Note that only $t^* - 1$ of these are new, because the chunk $\langle 1, 3, 5, \ldots, n-4, n-2, n\rangle$ was already counted in the previous paragraph.

This gives a total of $2t^* - 1$ distinct chunks that must be in $C$. By definition of $t^*$ we have $2t^* - 1 = n = |C|$. Therefore, if there indeed exists a schedule $A$ as supposed at the start of the proof, it must reconstruct from exactly these chunks. Doing this reconstruction gives that $A$ is 2-regular. □

*Example 3.27.* We illustrate the proof for $n = 7$. In this case $t^* = 4$. Consider all possible 4-chunks of gap $\leqslant 2$ that contain the number 1 or $n$:

$$1: \ \langle 1,2,4,6\rangle \ \langle 1,3,4,6\rangle \ \langle 1,3,5,6\rangle \ \langle 1,3,5,7\rangle$$

$$n: \ \langle 1,3,5,7\rangle \ \langle 2,3,5,7\rangle \ \langle 2,4,5,7\rangle \ \langle 2,4,6,7\rangle$$

The 7 distinct chunks among these reconstruct to the 2-regular schedule.

We now prove a more general case of Conjecture 1.

**Theorem 3.28.** *If $n \bmod k \equiv -1$, then a $k$-regular schedule exists and is optimal.*

*Proof.* This time, consider $t^* = \lceil n/k \rceil$. We will show that the $k$-regular schedule is the unique schedule with $w(t^*) = k$ and no schedule has lower $w(t^*)$. This will show the regular schedule is optimal.

Again consider $t^*$-chunks, now of gap at most $k$. There are $t^*$ such chunks containing a 1 and therefore we need them all. (Here we use $n \bmod k \equiv -1$.) The same holds for chunks containing an $n$.

These chunks containing a 1 and/or $n$ do not determine the entire schedule by the same argument as before: there are not enough of them. They do, however, directly fix a part of the schedule, which we can find by inspecting the chunks. For the rest of the proof, we call this partial schedule $R$. It is illustrated in Table 3.3.

Note that $R$ is $k$-regular. If $k = 3$, then $R$ is in fact a complete schedule and we are done. In the following, consider $k \geqslant 4$. Any schedule with $w(t^*) = k$ contains this subschedule $R$. Next we will show that the rest of the schedule must also be regular.

A schedule must contain all the numbers in $[1..n]$. In $R$, we have already used the numbers $ik - 1$, $ik$ and $ik + 1$ for all appropriate $i$. So the numbers available for the rest of the schedule are:

$$\underbrace{[2 \ldots k-2]}_{\text{Group 1}}, \quad \underbrace{[k+2 \ldots 2k-2]}_{\text{Group 2}}, \quad \underbrace{[2k+2 \ldots 3k-2]}_{\text{Group 3}}, \quad \ldots$$

These can be divided, as shown, into *groups* $[(i-1)k + 2 \ldots ik - 2]$, for $i \in \{1 \ldots t^*\}$.

Now consider extending $R$ to the right. There needs to be a chunk $c$ that contains the last $t^* - 1$ numbers from $R$, plus one new number. The numbers fixed in $c$ are are $k+1, 2k+1, \ldots, n-k+2$. We are aiming for $w(t^*) = k$ and therefore $c$ must have gap at most $k$. This means the added number must come from group 1: otherwise $c$ has gap $k+1$, going from 0 to $k+1$. So we must

| | |
|---|---|
| $k-1$ | |
| $2k-1$ | |
| $\vdots$ | $ik-1$ for $i \in [1..t^*-1]$ |
| $n-k$ | |
| $\mathbf{n}$ | |
| $k$ | |
| $2k$ | |
| $\vdots$ | $ik$ for $i \in [1..t^*-1]$ |
| $n-k+1$ | |
| $\mathbf{1}$ | |
| $k+1$ | |
| $2k+1$ | |
| $\vdots$ | $ik+1$ for $i \in [1..t^*-1]$ |
| $n-k+2$ | |

**Table 3.3.** The partial schedule $R$ in the proof of Theorem 3.28. The schedule is read from top to bottom in the left column; the right column classifies the indicated blocks.

extend $R$ to the right with an element from group 1, say $x$. Then we repeat the argument to show that the next number must come from group 2 in order to fix the gap $x$ to $2k+1$: it is too big to leave open since $x \leqslant k-2$. Iterating this argument shows that the $i^{\text{th}}$ number we add must come from group $(i \ \mathbf{mod}\ t^*)+1$.

We will now show that, in stepping regularly through the groups, the numbers must be taken in $k$-regular order. Beginning from $R$ again, consider the first two numbers to add, say $x_1$ from group 1 and $x_2$ from group 2. We will argue that $x_1 = 2$ and $x_2 = k+2$. Notice that these are $k$-regular steps.

First of all, if $x_1 = 2$, then $x_2 = k+2$ (since otherwise the gap $x_1$ to $x_2$ would be too big) and we are done.

For contradiction, now consider taking $x_1 \neq 2$. Then also $x_2 \neq k+2$: at some later point we need to take the 2 and then the $k+2$ must come after it (not some other number from group 2). Consider the point in the schedule where we do add the 2. There needs to be a chunk $c$ with the number 2 in addition to one number from each of the groups 2 up to $t^*$. The number from group 2 is *not* $k+2$: we cannot take it before we take the 2, because we need to take it after the 2. But then $c$ has gap larger than $k$: from 2 to this number from group 2, which is strictly larger than $k+2$. This contradicts the existence of a schedule with $w(t^*) = k$. Therefore we need $x_1 = 2$.

Repeating this argument shows that we need to start by taking the smallest number from every group. After we have done so, again we need to take the smallest number from each group. This continues until all numbers are taken.

The resulting schedule is k-regular. □

To recap the argument, notice that a schedule with $w(t^*) < k$ is impossible (since there are not enough chunks of sufficient gap containing a 1). Since a schedule with $w(t^*) = k$ is necessarily regular, this means any non-regular schedule has $w(t^*) > k$. This gives optimality of the k-regular schedule.

Theorem 3.28 claims only the case $n \textbf{ mod } k \equiv -1$. Outside of this case, the proof technically fails: there are too many feasible chunks containing a 1, and we can no longer argue that we need all of them. What's more, the k-regular schedule is, in general, not the unique schedule that attains $w(t^*) = k$. See for example $n = 9$ in Table 3.2: the 4-regular schedule is not the only schedule achieving $w(3) = 4$.

The general case of Conjecture 1 remains open.

## 3.7. Reverse-binary schedule

In the previous section a class of regular schedules was proven to be optimal (Theorem 3.28). These schedules are particularly well-suited for certain values of $t$ (that is, $t = \lceil n/k \rceil$) and do not necessarily perform well at other times. We will now design a type of splitline schedule that is good at all times, though not optimal in the sense of the previous section. To this end, we will design a schedule that achieves a good spread of splitlines, at all offsets $t_0$ and at many values for $t$.

### 3.7.1. Interleaved pre-order traversal

We will start by introducing the *interleaved pre-order traversal* of a tree.

**Definition 3.29 (Interleaved pre-order traversal).** Let $T$ be a complete ordered binary tree rooted at vertex $v$, that is, all internal nodes have out-degree 2 and all leaf nodes have equal depth. If $v$ is the only vertex in the tree, the interleaved pre-order traversal $\mathrm{Trav}(v)$ is the list containing only $v$. Otherwise $\mathrm{Trav}(v)$ is the ordered list $(v, L[1], R[1], L[2], R[2], \ldots)$ where $L = \mathrm{Trav}(\mathrm{left}(v))$ and $R = \mathrm{Trav}(\mathrm{right}(v))$.

See Figure 3.5 for an example. Consider a complete binary tree where the leaf nodes are labeled 1 through $n$ from left to right ($n$ a power of two) and where the internal nodes are left unlabeled. We will interpret the leaf labels as splitline locations and in this way any traversal of a schedule tree gives rise to a splitline schedule. See Figure 3.6 for an example. We will call this schedule the Reverse-binary schedule, for reasons that will become clear later on.

**Definition 3.30 (Reverse-binary schedule).** The *reverse-binary schedule* of length $n = 2^k$ is the interleaved pre-order traversal of a complete ordered binary tree $T$

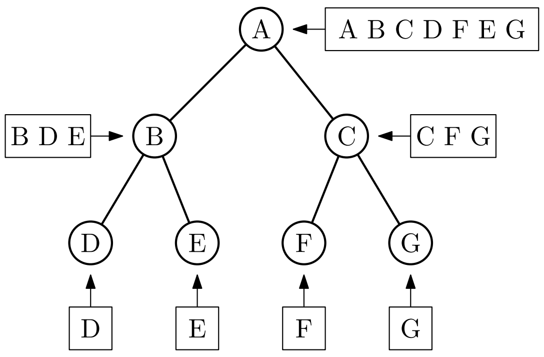


**Figure 3.5.** A labeled complete binary tree with the corresponding interleaved pre-order traversal indicated at every node.

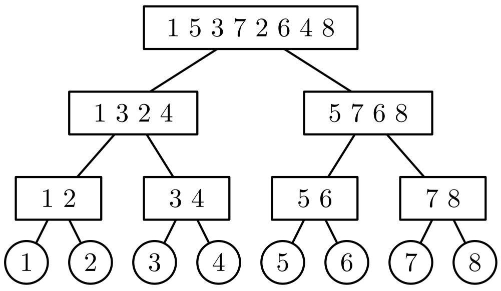


**Figure 3.6.** The interleaved pre-order traversal of a leaf-labeled tree, with the corresponding sub-traversal indicated at every internal node.

where the leaves are labeled $1$ to $n$ (from left to right) and the internal vertices are unlabeled and ignored in outputting the traversal.

The reasoning for interleaving the left and right subtraversals is that subsequent elements in the traversal then come from 'opposite' sides of the tree. In a schedule this corresponds to well-distributed splitlines. The quality as a schedule will be analysed in the next subsection, but first we consider some properties of the traversal itself.

The interleaved pre-order traversal of a tree can be done in linear time and space.[3] If the tree needs to be in memory, or the output is in the form of a list, then $\Omega(n)$ space is of course necessary. However, for the purpose of deriving a splitline schedule we are not interested in actually performing the traversal. In our setting the base stations send the splitlines one at a time and do not necessarily need to remember previous splitlines, as long as they know enough to calculate the next splitline. We will give an algorithm that enumerates the labels in a reverse-binary schedule of length $n$ with logarithmic space (for holding a number in the range $[0..n-1]$) and amortised constant delay.

[3]For example, the traversal can be done with constant delay on a pointer machine with 2 pointers of working storage per node, and constant additional amount of pointers. We will not dwell on these details here.

For convenience in the analysis, we will consider labels in the range $[0..n-1]$ instead of $[1..n]$. The acquired results hold for the reverse-binary schedule by simple translation.

**Definition 3.31 (Schedule tree).** The *schedule tree* of height $h$ is a complete ordered binary tree on $n = 2^h$ leaves, where the leaves are labeled $0$ to $n-1$, from left to right, and the internal vertices are unlabeled.

**Proposition 3.32.** *In a schedule tree, label the edges to left children with* $0$ *and the edges to right children with* $1$. *Then the path from the root to a leaf in a schedule tree gives the binary expansion of the label of the leaf.*

**Definition 3.33 ($h$-Foliage).** For a complete binary tree $T$, an $h$-*foliage* is a height-$h$ subtree of $T$ containing $2^h$ of $T$'s leafs.

Note that a foliage is itself a complete binary tree. For any $h$, and taking the labels modulo $2^h$, all $h$-foliages of a schedule tree are congruent. (This follows by considering the binary expansion of the leaf labels.)

When indicating the bits of a number, we call the least significant bit (lsb) 'bit 1', the second least significant 'bit 2,' and so on.

**Lemma 3.34.** *Let* $F$ *be an* $h$-*foliage of a schedule tree. In the interleaved pre-order traversal of* $F$, *the* $h^{th}$ *lsb alternates between* $0$ *and* $1$.

*Proof.* The tree $F$ contains exactly two $(h-1)$-foliages: let $F_L$ be the one rooted at $\mathtt{left}(\mathtt{root}(F))$ and $F_R$ the one rooted at $\mathtt{right}(\mathtt{root}(F))$. Since $\mathtt{root}(F_L)$ is a left child of a node at height $h$, all leaf labels in $F_L$ have their $h^{th}$ bit set to $0$ (Proposition 3.32). Similarly, all leaf labels in $F_R$ have their $h^{th}$ bit set to $1$. Labels in the interleaved pre-order traversal of $F$ come alternatingly from $F_L$ and $F_R$ and therefore alternate their $h^{th}$ bit. $\square$

**Lemma 3.35.** *Let* $F$ *be an* $h$-*foliage of a schedule tree and let integer* $i \leqslant h$. *In the interleaved pre-order traversal of* $F$, *the* $i^{th}$ *lsb of the labels starts at* $0$ *and then flips after every* $2^{h-i}$ *labels.*

*Proof.* By induction on $h$.

If $h = 1$, then there are $2$ leaves. First in the traversal is the left child of a node at height $1$. Therefore its label has lsb $0$. Then comes a right child, with lsb $1$, and the lemma holds.

Now suppose the lemma holds for all $h$-foliages; this implies the lemma for $h+1$ as follows. Consider a foliage $F$ of height $h+1$ and call its height-$h$ left and right subfoliages $F_L$ and $F_R$ respectively. The traversal of $F$ is the interleaving of the traversals of $F_L$ and $F_R$. First we look only at the $h$ least significant bits. The induction hypothesis holds for $F_L$ and $F_R$ and therefore their traversals agree on the $h$ least significant bits. The traversal of $F$ is then, for the $h$ least significant bits, just the same as $F_L$ and $F_R$ except that every element is doubled. Any alternating patterns in these bits then have their period doubled. This accounts for the $h$ least significant bits. As for the $(h+1)^{st}$ lsb, Lemma 3.34 gives that it alternates. Hence the lemma holds for $h+1$.

□

**Corollary 3.36.** *In the interleaved pre-order traversal of a schedule tree, the $i^{th}$ leaf label encountered equals the binary expansion of $i$ read backwards.*

*Proof.* Consider the binary expansions of the numbers in the list $[0..n-1]$. The first number is all-0. Then the least significant bit flips every time. The second bit flips every other time and, in general, the $k^{th}$ lsb flips after every $2^k$ numbers.

Let $T$ be a schedule tree of height $h$ and consider its interleaved pre-order traversal. The first number is also all-0. Then, since $T$ itself is an $h$-foliage, Lemma 3.35 applies. □

Standard results on incrementing a binary counter now directly give that the reverse-binary schedule can be generated using amortised constant delay (worst case logarithmic). The only memory used by the enumerating algorithm is the current label (which we will interpret as a splitline location). The next label can be calculated using only a small constant amount of bits working space.

### 3.7.2. Analysis

We will now analyse the quality $w(t)$ of the reverse-binary schedule.

**Theorem 3.37.** *Let $1 \leqslant t \leqslant n$. The reverse-binary schedule has $w(t) \leqslant \frac{4\ell}{t}$. If additionally $t$ is a power of two, $w(t) \leqslant \frac{2\ell}{t}$.*

*Proof.* First we prove the latter. Consider the two most significant bits of any four consecutive splitlines. By Corollary 3.36 these always take on exactly the four possibilities `00`, `01`, `10` and `11`. This means that for any four consecutive splitlines, there is one in each quarter of the line segment. Within their quarter, they might lay anywhere. Still, this gives that the gaps between them can be at most $\ell/2$. This gives $w(4) \leqslant \ell/2$. This generalises to $w(t) \leqslant 2\ell/t$, for $t$ a power of two.

To analyse non-power-of-two values for $t$, we use that $w(t)$ is non-increasing in $t$: a value of $w(t)$ for a certain $t$ upperbounds all later $w(t')$. We will handle non-power-of-two values for $t$ by taking the bound for the largest power of two smaller than $t$. Rounding down to a power of two costs at most a factor two. This gives $w(t) \leqslant \frac{4\ell}{t}$. □

Recall from Lemma 3.12 that any schedule has

$$w(t) \geqslant \ell/(t+1). \tag{3.27}$$

The reverse-binary schedule, by Theorem 3.37, has

$$w(t) \leqslant 4\ell/t. \tag{3.28}$$

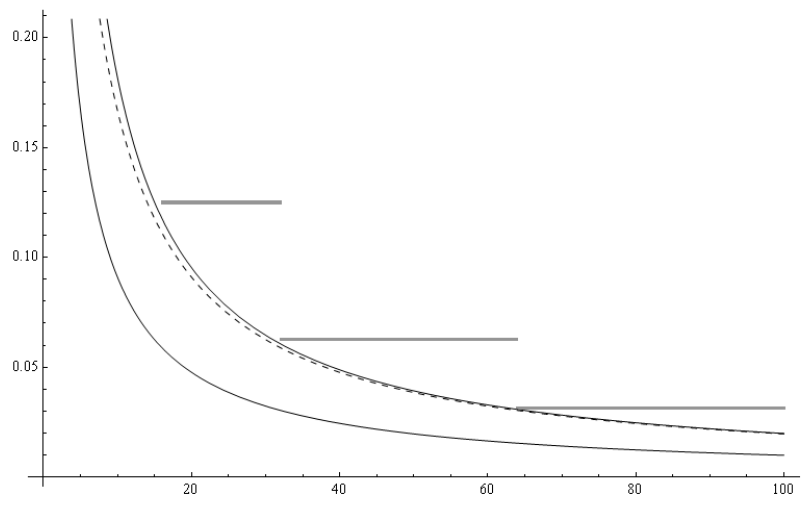


**Figure 3.7.** Curves indicate expected uncertainty for random splitlines as in Figure 3.2. Horizontal line segments indicate upper bound on worst case uncertainty for the reverse-binary schedule (Theorem 3.37).

This means that, for all t, no schedule can have a $w(t)$-value that is more than a factor

$$\frac{4\ell}{t} \Big/ \frac{\ell}{t+1} = 4 + \frac{4}{t} \tag{3.29}$$

smaller than the reverse-binary one. That is, at every time t, the reverse-binary schedule gives a worst case guarantee that is within a factor slightly-more-than-4 of the best $w(t)$ that any schedule can give for this time — even if that schedule is constructed specifically for this value of t, like the lower bound and the regular schedules were. For t a power of two, the ratio even improves to $2 + 2/t$.

**Corollary 3.38.** *Let $1 \leqslant t \leqslant n$. For t a power of two, no schedule exists that has a worst case uncertainty that is smaller than that of the reverse binary schedule divided by $2 + 2/t$. For other t, the factor is bounded by $4 + \frac{4}{t}$.*

Next, we compare the worst case uncertainty for the reverse-binary schedule (see Theorem 3.37) with the expected uncertainty for random splitlines (see Theorem 3.4). The ratio between $w(t) \leqslant \frac{4\ell}{t}$ and $e(t) = \frac{2\ell}{t+2}$ is $2 + 4/t$. For t a power of two, this improves to $1 + 2/t$. These values are plotted in Figure 3.7.

So the worst case guarantee of the reverse-binary schedule is close to the expected value for random splitlines, but these are values with different meanings. Perhaps more interesting is the probability that the random splitlines perform *worse* than the reverse-binary schedule. That is: the probability that t random splitlines give an uncertainty that is higher than the worst case of hearing t consecutive splitlines from the reverse-binary schedule. Call this probability, as function of t, $p(t)$.

Let $\mathcal{X}_t$ be a binary random variable indicating whether for a node at uniformly random location, t independently uniformly random splitlines give lower

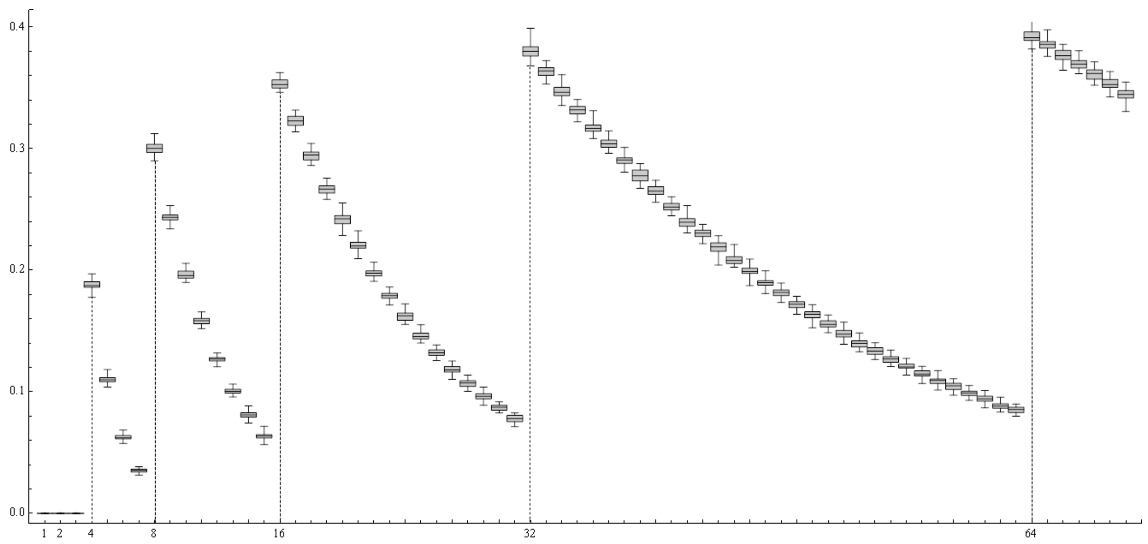


**Figure 3.8.** Estimate for $p(t)$ by 10,000 samples of $\mathcal{X}_t$; box-whisker plot of 100 trials, indicating sample minimum, quartiles and maximum.

uncertainty (0) or higher uncertainty (1) than the worst case uncertainty bound for the reverse-binary schedule. Then $\mathbb{E}[\mathcal{X}_t] = p(t)$. In this way, we have estimated $p(t)$ computationally using 10,000 samples of $\mathcal{X}_t$. We have then repeated this experiment 100 times and give a box-whisker plot (indicating sample minimum, quartiles and maximum) in Figure 3.8.

Notice that at power-of-two values for $t$, the probability $p(t)$ is high. This makes sense, since then $e(t)$ is near the reverse-binary schedule's $w(t)$. For other $t$, as the ratio between $w(t)$ and $e(t)$ increases, so the probability $p(t)$ decreases.

### 3.7.3. Communication vs Sensor memory trade-off

Generating the reverse-binary schedule is very cheap. We can make a trade-off between the amount of location messages (communication) and memory in a sensor in the following way. Instead of sending a location message after each splitline message, the base stations send a location message only once every $k$ splitline messages. The sensor node then uses $k$ bits of additional memory to remember the last $k$ splitline results ("did I hear **A** first, or **B**"). As long as it has not heard a location message yet, it does not know exactly what these bits mean for its location. (It could use the estimator based on $\mathcal{K}$ from Section 3.4.1.) As soon as it receives a location message, it can easily generate the previous splitline locations and can now interpret the earlier splitline results. Also, it can interpret any later splitlines messages. This costs $k$ extra bits of memory and a little computation for each sensor node, but every $k$ rounds, it saves $(k-1) \cdot \mathcal{O}(\log n)$ bits of communication from the base stations.

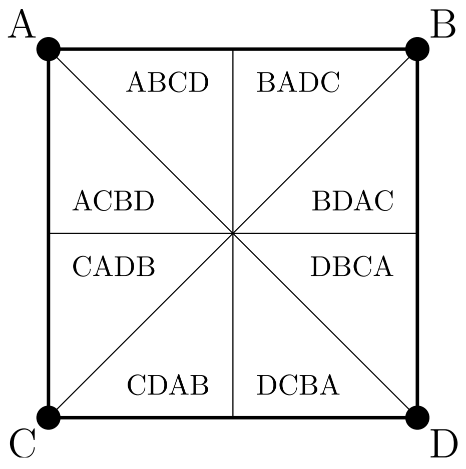


**Figure 3.9.** Information from a single round of splitline messages when localising on a rectangle.

## 3.8. Extension to higher dimensions

Our localisation schemes can easily be extended to higher dimensions. By way of illustration we go from a line segment and 2 base stations to a rectangle using 4 base stations. This generalises to a d-dimensional hypercube using $2^d$ base stations.

Again, the base stations will do carefully timed transmissions and the sensor nodes will infer bounds on their location. We put base stations on each of the four corners and there will now be 4 splitline messages per time step. If all base station transmit simultaneously the arrival order of these four messages divides the rectangle into eight areas as indicated in Figure 3.9. Notice a horizontal and a vertical splitline, and two diagonal splitlines.

We will ignore the diagonal splitlines and concentrate on using the axis-aligned splitlines. We can aim their intersection by varying the timing by the base stations. Consider the intersection between the horizontal splitline and the line segment **AB**. Equation 3.1 prescribes a time difference between the transmission times of A and B, say $\Delta_x$. This same time difference must apply to the transmission times of C and D. Similarly we get a time difference $\Delta_y$ for **CA** and **DB**. This set of differences is consistent: pick some $t_C$ and set

$$t_A = t_C + \Delta_y,$$

$$t_D = t_C + \Delta_x,$$

$$t_B = t_C + \Delta_x + \Delta_y.$$

This allows us to simultaneously aim a horizontal and a vertical splitline. (In higher dimensions, a set of d axis-perpendicular planes.)

We then construct a schedule for the 2D localisation problem by getting both the x and the y coordinate of the splitpoint from a 1D splitline schedule. Bounds on (or expectation of) the uncertainty of the 1D schedule then transfer directly to the individual dimension to which it is applied.

If the same schedule is used on both axes, all splitpoints will be aimed on the line segment **CB**. Then a location message of size $\mathcal{O}(\log \ell)$ suffices, instead of $\mathcal{O}(d \log \ell)$.

## 3.9. Concluding remarks

In this chapter we have introduced a new model for localisation in wireless sensor networks. It has the advantages of being range-free and cheap in terms of local computation and energy consumption. We give several localisation schemes in this model. For the randomised ones, we have analysed the expected performance. For the (deterministic) splitline schedules, we conjectured that a certain class of schedules is optimal and prove this for some restricted cases. The proof technique used in these restricted cases breaks down for the general case. Using a different approach, we give a set of schedules with a performance that is, at all times, within a constant factor of optimal. An improvement in this factor (either from a better schedule or from a better lower bound) would be interesting. Regarding the extension to higher dimensions, it seems that $d + 1$ base stations should suffice for localisation in a d-dimensional simplex. We have not investigated this. For high $d$, this would be a significant improvement over our hypercube approach, but for the important cases of $d \in \{1, 2, 3\}$ the difference is not large.

# 4 A moderately exponential time algorithm for Link Independent Set

An important problem in sensor networks, or wireless networking in general, is the scheduling of transmissions. This is a problem of both practical and theoretical importance. On the practical side, it is a problem that any actual system solves, one way or another. On the theoretical side, it is one of the most fundamental questions that can be studied for any model of communication. In this chapter and the next, we specifically look at the so-called "physical model:" collisions between simultaneous transmissions are determined by a signal-to-interference-plus-noise-ratio (*SINR*) condition. In this model, we specifically look at the problem of scheduling as many simultaneous transmissions as possible, given some set of requests: Link Independent Set. See Section 2.6 for a review of this problem and of the physical model.

We develop an exponential-time branching algorithm for Link Independent Set. The performance of the algorithm is dependent on the availability of certain substructures in the instance. In the next chapter, where we experimentally investigate the structural properties of link independent set instances, we will see that these substructures are common. In this chapter we show that the algorithm has worst-case runtime $\mathcal{O}(1.888^n)$ on instances in which these substructures are available.

## 4.1. Introduction

In the context of sensor networks, there are two widespread classes of models for wireless communication. Many results on wireless networks take a graphical model of connectivity and signal collision: for every pair of nodes in the network, there either is or is not a possibility to communicate, independent of the presence of other nodes. Usually a (geometric) distance threshold is used to determine this. Collisions between simultaneous transmissions are then defined in terms of the resulting graph. For example, a collision at a node $v$ may be defined to occur if more than one of $v$'s neighbours transmit at the same time. Typically these conditions consider only small graph distances. The local nature of this definition makes these models convenient for algorithms and analysis.

A problem with this model is that it deviates from reality in significant ways. On the one hand, it inherently does not model certain physically feasible sets of transmissions (as demonstrated with real hardware by Moscibroda et al. [MWW06]). In this way, it underestimates what is possible in reality. On the other hand, it neglects the combined interfering effect of many distant nodes together. This is an overestimation of what is really possible.

In order to design algorithms that better match reality, the so-called 'physical model' for wireless networks has been introduced, in which collisions are modelled by a *signal-to-interference-plus-noise ratio* (*SINR*) condition. An overview of the basic definitions and notations in this model is provided in Section 2.6.

The *SINR* model has been extensively studied in a channel capacity setting since the pioneering work of Gupta and Kumar [GK00]. At first the focus was on problems relating to probabilistic channel capacity; more recently an interest has developed for algorithmic results on worst case instances (for example [GWHW09, HW09]). This leads for example to the problem of scheduling of communication requests in this model, where the global nature of interference sets the model apart from many others. The problem of picking as many simultaneous requests as possible is called the Link Independent Set problem. Fulfilling all requests in the least number of timeslots is called Link Scheduling.

The decision version of both Link Independent Set and Link Scheduling are $\mathcal{NP}$-hard, as shown by Goussevskaia et al. [GWHW09]. As mentioned in Section 2.6, there has been significant interest in the Link Independent Set problem, though it goes by many different names, like One-slot Scheduling, One-shot Scheduling, Link Capacity and Max-Connections. Foremost among the algorithmic results are various approximation algorithms, both centralised and distributed, also for variations and special cases of the *SINR* model. See for example [RMR+06, XTW10, BH14, GÁB+14].

Hua and Lau give inclusion/exclusion algorithms for Link Scheduling that use either $\mathcal{O}^*(2^n)$ time and exponential space, or $\mathcal{O}^*(3^n)$ time and polynomial space [HL08]. Their algorithm involves solving Link Independent Set, but they do so with an $\mathcal{O}^*(2^n)$-time algorithm because that is not the bottleneck for their runtime. They mention that an improved exact algorithm for Link Independent Set would nevertheless be of interest.

In this chapter we use the following statement of the Link Independent Set problem.

| | **Link Independent Set (Abstract)** |
|---|---|
| *Instance:* | An $n \times n$ gain matrix with non-negative entries. |
| | Reception threshold $\beta$. |
| | An integer $k$. |
| *Question:* | Does there exist a subset consisting of at least $k$ of the links that is link independent? |

## Results

We develop a moderately exponential time branching algorithm for Link Independent Set (Abstract). An important aspect of the analysis of its runtime is our branching rule *Limiting Pair*. This rule tries to pick links to branch on such that it can prove an upperbound on the solution size in the subproblems; how well it does so when a suitable link is chosen, is called the *quality* of the branch.

The algorithm works on the *gain matrix* (or: *path loss matrix*) of the problem instance (definition is given in Section 2.6). With high probability, geometric instances have a gain matrix with a certain domination property. The main result of this chapter is Algorithm 1 for solving Link Independent Set (Abstract) and a proof of the following theorem. The condition it requires does not have a clean interpretation without more technical details at this point; these details follow later in the chapter, and in the next chapter we will experimentally demonstrate that this condition is likely to hold in practice.

**Theorem 4.1.** *Algorithm 1 solves* Link Independent Set (Abstract). *If Limiting Pair branches are always of quality* 0.333 *or better unless all optional rows are safe, then the algorithm has runtime* $\mathcal{O}(1.888^n)$.

We will see in the next chapter that the effective branching factor of Algorithm 1 is much lower than 1.888 in practice. Of great importance to the practical runtime of a branching algorithm is the time spent on branch selection. We provide a data structure to speed up this part of the algorithm.

**Theorem 4.2.** *Branch selection for Algorithm 1 can be done in* $\mathcal{O}(n \log n_o)$ *time using a data structure that can be built in* $\mathcal{O}(n_o^2 \log n_o)$ *time, where* $n_o$ *is the size of the original instance and* $n$ *is the size of the remaining instance being branched on. The data structure uses* $\mathcal{O}(n_o^2)$ *space.*

First we note that Link Independent Set (Abstract) has an elegant formulation as an integer linear program. This formulation can be useful for easily implementing experiments: given an off-the-shelf ILP solver, this approach is simpler to implement than our algorithm. It can also be used when experimentally investigating variants of the *SINR* model to which our algorithm might not be easily adapted.

## 4.2. Integer linear programming formulation

Here we briefly show that the LINK INDEPENDENT SET (ABSTRACT) problem has a simple formulation as an integer linear program. This formulation may be useful for computational verification and further exploration of the problem and variations thereof.

Recall the problem statement: $g_{ij}$ is the strength of the signal from sender $i$ as it arrives at receiver $j$, $\beta$ is the reception threshold, and we want a maximum cardinality set $\mathcal{I} \subseteq S$ such that

$$\frac{g_{ii}}{\sum_{j \in \mathcal{I}, j \neq i} g_{ji}} \geqslant \beta, \qquad \forall i \in \mathcal{I}. \tag{4.1}$$

We use a binary decision variable $x_i$ to indicate whether link $i$ is chosen in $\mathcal{I}$, that is, let $\mathcal{I} = \{i \mid x_i = 1\}$. Rearranging Equation (4.1) reveals that these constraints are individually linear for fixed $g_{ij}$ and $\beta$:

$$\sum_{j \neq i} g_{ji} x_j \leqslant g_{ii}/\beta, \qquad \forall\, i \in \{i \mid x_i = 1\} \tag{4.2}$$

We want to enforce this constraint only for values of $i$ where the decision variable $x_i$ takes value 1. We handle this using the following trick: add a large constant $M$ to the right-hand side of the constraint and add it to the left-hand side only if $x_i = 1$.

$$Mx_i + \sum_{j \neq i} g_{ji} x_j \leqslant M + g_{ii}/\beta, \qquad \forall i \tag{4.3}$$

Pick $M = \sum_{j \neq i} g_{ji}$ and consider what happens depending on the value of $x_i$. If $x_i = 0$, then constraint $i$ is unconditionally satisfied (by choice of $M$). If $x_i = 1$, on the other hand, then the original constraint remains. Therefore Equations (4.1), (4.2) and (4.3) are equivalent.

The result is an integer linear program with a linear number of variables and constraints. Maximising $\sum x_i$ results in a set $\mathcal{I}$ of maximum-cardinality, thereby solving LINK INDEPENDENT SET (ABSTRACT); see the full program below. We note that the constraints are all knapsack constraints: $\leqslant$-constraints with only positive coefficients. While of course still $\mathcal{NP}$-hard, this is a form that an ILP solver might be particularly efficient on in practice.

Integer Linear Program: Link Independent Set (Abstract)

Maximise

$$\sum_{i \in S} x_i \tag{4.4}$$

Subject to:

$$Mx_i + \sum_{j \in S \setminus \{i\}} g_{ji} x_j \leqslant M + g_{ii}/\beta \qquad \forall i \in S \tag{4.5}$$

$$x_i \text{ binary} \qquad \forall i \in S \tag{4.6}$$

## 4.3. Internal time lemma

In this section we introduce an observation about the runtime of branching algorithms. It is not shocking and the proof is not complicated, but as a general observation we think it is not quite 'common knowledge' and therefore believe it merits mention separate from our specific application.

Consider a branching algorithm $A$ and integers $k$ and $d$ such that $A$ solves an instance of size $n$ by recursively solving at most $k$ instances, each of integer size at most $n - d$. It solves instances of at most some constant size by special case, taking constant time each. The branching tree (or: recursion tree) of $A$ on an instance of size $n$ then has height $\mathcal{O}(\, n/d\,)$ and the number of leafs is $\mathcal{O}(\, k^{n/d}\,)$ = $\mathcal{O}(\, \sqrt[d]{k}^{\,n}\,)$.

Let $\mathcal{O}(\, f(m)\,)$ be an upperbound on the time $A$ spends in a branching node in case the instance has remaining size $m$. (Think for example of branch selection, reduction rules, and combining the recursive results.) Many exponential-time branching algorithms have the described form and the function $f$ is typically polynomial. The total runtime of $A$ is then easily seen to be $\mathcal{O}^*(\, k^{n/d}\,)$: the time spent in internal branch nodes is polynomially related to the time spent in the branch leafs. It is in fact possible to spend more time in the internal nodes asymptotically 'for free.'

**Lemma 4.3 (Internal Time).** *The runtime of* $A$ *on an instance of size* $n$ *is* $\mathcal{O}^*(\, k^{n/d} + f(n)\,)$.

*Proof.* The time spent in branch leafs is $\mathcal{O}^*(\, k^{n/d}\,)$: that is how many leafs there are, and they require constant time each. Consider the time spent in internal nodes. The time spent in the branching root is $\mathcal{O}(\, f(n)\,)$ since we are still looking at the entire instance there. For the other internal nodes, consider two cases depending on how $f(n)$ relates to $k^{n/d}$.

<u>Case</u> $f(n)$ *is* $\mathcal{O}^*(\, k^{n/d}\,)$.

The branching nodes at depth 1 together take $\mathcal{O}(\, k \cdot f(n - d)\,)$ time, which is $\mathcal{O}^*(\, k \cdot k^{\frac{n-d}{d}} \,) = \mathcal{O}^*(\, k^{n/d} \,)$. In general, this is $\mathcal{O}(\ k^i \cdot f(n - id)\ )$ for all nodes at depth $i$ together, which is $\mathcal{O}^*(\, k^i \cdot k^{\frac{n}{d} - i} \,) = \mathcal{O}^*(\, k^{n/d} \,)$.

Summing over at most $n$ levels then gives a total time of $\mathcal{O}^*(\, k^{n/d} \,)$.

Case $f(n)$ *is* $\Omega(\, k^{n/d} \,)$.
The branching nodes at depth 1 together take $\mathcal{O}(\, k \cdot f(n-d) \,)$ time. This is $\mathcal{O}(\, f(n) \,)$, since $f(n)$ itself grows faster than $k^{n/d}$. In general, this is $\mathcal{O}(\, k^i \cdot f(n - id) \,)$ for all nodes at depth $i$ together, which again is $\mathcal{O}(\, f(n) \,)$.

Summing over at most $n$ levels then gives a total time of $\mathcal{O}^*(\, f(n) \,)$

This combines to a bound of $\mathcal{O}^*(\ k^{n/d} + f(n)\ )$. □

The lemma implies that, disregarding polynomial factors, it is free to spend the "moderately-exponential" amount of $\mathcal{O}^*(\, k^{m/d} \,)$ time in *each and every* branching node, where $m$ is the size of the remaining instance in the branching node. Then the term $f(n)$ is still dominated by the term $k^{n/d}$. Note also that the $f(n)$ term occurs only once in the runtime, by itself, and that the polynomial term hidden in the $\mathcal{O}^*(\,\cdot\,)$ notation is only $\mathcal{O}(\, n \,)$.

We are now ready to develop the main result of this chapter: a moderately-exponential time branching algorithm for Link Independent Set (Abstract). We will first focus on correctness and only later discuss runtime.

## 4.4. Basic operations

The algorithm works in terms of the gain matrix. Recall that this matrix contains, for every pair of a sender $i$ and a receiver $j$, the amount of signal received at receiver $j$ if sender $i$ transmits. Our algorithm works for arbitrary gain matrices and does not use any geometric information (like, for example, position of senders and receivers).

When experimentally evaluating the runtime in Chapter 5, we will look specifically at random geometric instances. Gain matrices arising from such instances typically have properties that enable our algorithm to branch efficiently. These properties might also hold in instances that do not come from geometric instances and might arise also during branching. For now, we look at arbitrary matrices and correctness. We recall the following concept from Section 2 (Definition 2.13).

**Definition 4.4 (Gain matrix).** A *gain matrix* $M$ is an $n \times n$ matrix holding the amount of *signal* $g_{ii}$ received for link $i$ on the diagonal, and the amounts of *interference* $g_{ij}$ in the off-diagonal entries. For $i \neq j$, the element $M[i, j]$ is the interference at receiver $i$, caused by sender $j$. An element on the diagonal is called a *signal*; an element off the diagonal is called an *interference*.

We will work with reception threshold $\beta = 1$ (see Equation 2.14). That is, a successful transmission requires that the amount of signal received is at least as much as the amount of interference. (Recall that we assume that there is no background noise.) For general gain matrices this choice of $\beta$ is without loss of generality since we can scale the values on the diagonal to simulate other values of $\beta$.

**Definition 4.5 (Safe row).** A row in a gain matrix is called *safe* if and only if all its entries are non-negative and the sum of the interferences is at most the signal, that is, if the sum of the off-diagonal elements is at most the value on the diagonal. A row that is not safe is called *unsafe*.

Note that if a row is safe, then the corresponding transmission succeeds unconditionally. Its signal is so strong that it would be received successfully despite all possible interference, hence the name *safe*.

The link independent set problem can now be formulated as follows: pick a set $\mathcal{I}$ of rows and the *corresponding* columns such that in the resulting matrix all rows are safe. Note that a matrix of which all rows are safe is also called *diagonally dominant*; thus we are basically asking for a suitable submatrix that is diagonally dominant. When the diagonal has been scaled by $\beta$, the safeness condition is exactly the *SINR*-condition. This is the problem statement we will work with in this chapter.

| | **Link Independent Set (Arbitrary matrix)** |
|---|---|
| *Instance:* | An $n \times n$ matrix $M$ with non-negative entries. |
| | An integer $k$. |
| *Question:* | Does there exist a subset consisting of at least $k$ rows of $M$, with their corresponding columns, such that in the resulting submatrix all rows are safe? |

At this point we make an observation about safe rows. Should we always greedily take a safe row into a solution if we encounter one? This is not the case. As noted, a safe row is indeed safe from having its *SINR*-condition violated. However, the interference its sender causes might ruin many other rows that are themselves not safe. Consider for example the following matrix.

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 2 & 0 & 1 \end{bmatrix}$$

Notice that the first row is safe. Taking it, however, immediately violates the *SINR*-condition of the second and third row: sender one causes too much interference for the other transmissions to be successful. Meanwhile, the unique maximum link independent set consists of only the second and third rows. This means that we cannot unconditionally, greedily take safe rows in the solution.

During the run of the algorithm, we will mark the rows of the matrix with a label: *optional* or *required*. Conceptually, there are also *forbidden* rows, but we delete those from the matrix instead of labelling them. We start by marking all rows as *optional*. These rows might end up in the solution or they might not. By extension, we call interference (that is, an off-diagonal element in the gain matrix) *optional* if it corresponds to an optional row. In the analysis of the runtime, we shall measure the size of an instance by the number of *optional* rows remaining. We will now discuss two basic operations that we will use in the various branching and reduction rules: *require* and *forbid*.

The *require* operation on a row $r$ corresponds to deciding that we will definitely take $r$ in the independent set.

**Definition 4.6 (Require operation).** To *require* an optional row $r$, 'commit' all the interference that $r$ causes onto the signal amounts on other rows: for all rows $i \neq r$ subtract interference $M[i,r]$ from signal $M[i,i]$ and set $M[i,r]$ to 0. Furthermore, mark $r$ as *required*. We write $M/\text{require}(r)$ for the result of applying the *require* operation on row $r$ in $M$.

This transformation is valid: on every row $i \neq r$ we subtract $M[i,r]$ signal and also subtract $M[i,r]$ interference. If we indeed take $r$ in the independent set, this produces equivalent *SINR*-conditions. In the following example, changes have been highlighted.

$$\begin{bmatrix}5&2&3&1\\3&5&3&2\\2&2&5&3\\3&3&1&5\end{bmatrix}\begin{array}{l}\text{Optional}\\\text{Optional}\\\text{Optional}\\\text{Optional}\end{array}\xrightarrow{\text{require second row}}\begin{bmatrix}\mathbf{3}&\mathbf{0}&3&1\\3&5&3&2\\2&\mathbf{0}&\mathbf{3}&3\\3&\mathbf{0}&1&\mathbf{2}\end{bmatrix}\begin{array}{l}\text{Optional}\\\textbf{Required}\\\text{Optional}\\\text{Optional}\end{array}$$

**Observation 4.7.** *The require operation does not affect whether any row is safe or not.*

We do not remove row $r$ from the instance: unless $r$ is safe, it still represents a *SINR*-condition that we must be careful to respect. In fact, in the example above, we know after requiring the second row that certainly we cannot have both row 1 and 3 in the independent set as well: then receiver two would get 6 interference while the signal has strength only 5. Notice further in the example above that, after we require the second row, we can no longer have both the first and the fourth row: that would give 3 interference on row four while only 2 signal remains unspoken for once we decided that we will certainly include row two in the independent set.

In these ways, the require operation provides progress for the branching algorithm even though it does not reduce the size of the matrix: it reduces the number of rows marked *optional* by one. Also, signals $M[i,i]$ may be decreased and this can enable reduction rules.

The other important operation is called *forbid*. Forbidding a row $r$ corresponds to deciding that we will not take $r$ in the solution. It works as follows.

**Definition 4.8 (Forbid operation).** To *forbid* an optional row $r$, remove it and the corresponding column from the instance. We write $M/\text{forbid}(r)$ for the result of applying the *forbid* operation on row $r$ in $M$.

In contrast to the require operation, forbidding a row can make other rows safe: some off-diagonal elements (interference) are removed from $M$.

## 4.5. Branching and reduction rules

A branching algorithm consists of branching rules, reduction rules and base cases. A branching rule is a procedure that decomposes an instance into a number of *cases* (instances that are smaller, by some measure) and combines the solutions for these cases into a solution for the original instance. A branching rule is called *valid* if and only if this combined solution is an optimal solution to the original instance. A reduction rule is a branching rule with only one case: it simplifies the instance. A set of branching and reduction rules can be applied recursively to solve an instance, where some *base cases* terminate the recursion on instances that are somehow simple (for example: constant size). This is a branching algorithm. Correctness and optimality of the branching algorithm follows from the validity of its constituent rules and base cases.

We will now develop a set of valid rules and base cases. The require and forbid operations from the previous section suggest the following trivial branching rule, which branches on a single row of the matrix.

**Branching rule:** Single Row
*Pick an optional row $r$. Branch in two cases and return the larger solution.*

**Case Require row $r$.**

**Case Forbid row $r$.**

**Proposition 4.9.** *The branching rule Single Row is valid.*

Exhaustively using this rule and checking, as a base case, the branch leafs leads to an $\mathcal{O}^*(\,2^n\,)$-time algorithm. In itself this is particularly uninteresting because in that time we can just as well (and probably faster in a practical sense) enumerate all subsets of the set of rows and test each of them. However, this starting point puts us in a position to design an improved algorithm, by introducing reduction rules and more advanced branching rules.

**Definition 4.10 (Busted row).** Let $r$ be a row with $M[r,r] < 0$, that is, negative signal. Then $r$ is called *busted*.

**Reduction rule:** Busted Row
*Let $r$ be a busted row. If $r$ is optional, Forbid $r$. If $r$ is required, answer* No.

**Lemma 4.11.** *The reduction rule Busted Row is valid.*

*Proof.* If a row $r$ is busted, then its *SINR*-condition is necessarily violated by the senders corresponding to the set of currently required rows. If $r$ is *optional*, we *forbid* it because we know it cannot be taken in a valid solution along with the required rows. If $r$ is itself labelled *required*, then the current branch is not feasible and we can conclude No. □

One might notice that this reduction rule will not initially fire on any geometric instances: in a geometric instance all actual signal amounts are positive. However, *require* operations may lower signal values in the matrix and thereby enable this rule. Indeed this is what the following rule is designed to accomplish.

**Branching rule:** Busted Pair
*Let $r$ and $i$ be optional rows such that $M[r, i] > M[r, r]$. That is, the interference from $i$ at $r$ is larger than the signal at $r$. Branch in three cases and return the largest solution.*
**Case Require $i$, Forbid $r$.**

**Case Forbid $i$, Require $r$.**

**Case Forbid both $i$ and $r$.**

Notice that the case where both rows would be required is not included.

**Lemma 4.12.** *The branching rule Busted Pair is valid.*

*Proof.* Consider using the Single Row branching rule first on $i$ and then on $r$, checking for Busted Row inbetween. We argue that the Busted Pair rule has the same effect and is therefore also valid.

When the first Single Row branch requires row $i$, the signal $M[r, r]$ will drop below 0, since by the precondition for the Busted Pair rule we have that $M[r, i] > M[r, r]$. In that branch, the Busted Row rule forbids $r$. The branch where both $i$ and $r$ are required therefore does not occur. This is exactly what the Busted Pair rule does. □

The Busted Pair rule is a branch into three cases, each with a reduction in size of 2: two rows lose the label *optional*. If we could always apply this rule, the resulting runtime would be $\mathcal{O}^*(3^{n/2}) = \mathcal{O}^*(\sqrt{3}^n) \approx \mathcal{O}^*(1.73^n)$. It is unlikely, however, that this rule would always be enabled.

**Definition 4.13 (k-Limited).** A row is called $k$-limited if and only if the sum of its $k + 1$ smallest *optional* interferences is larger than its signal.

This means that committing $k$ more interferences on this row could work, possibly depending on which ones, but $k + 1$ is certainly not feasible. Notice that, by definition, if a row is safe, then it is not limited (that is, for no $k$ is the row $k$-limited): safe means that even all interferences together are not a problem, while $k$-limited means that any set of interferences above a certain size is always a problem.

**Lemma 4.14 (Limited required).** *Consider a gain matrix $M$, with $R$ the set of its required rows. Let $r \in R$ be $k$-limited and let* Opt *be the cardinality of the largest link independent set that is a superset of $R$. Then* Opt $\leqslant |R| + k$.

*Proof.* Row $r$ is marked *required*, so we have already decided that $r$ will be in the independent set. Therefore its *SINR*-condition must be satisfied. The interference from other required rows has already been subtracted from its original signal amount. Consider requiring at least $k+1$ additional rows. Then at least $k+1$ additional interferences will be subtracted. By definition of $k$-limited, row $r$ is then busted regardless of which $k+1$ additional rows are selected. Thus $R$ can be extended to a set of at most $|R|+k$ independent links, hence Opt $\leqslant |R|+k$. □

This lemma shows that it is, in some sense, advantageous to have a *required* row that is $k$-limited for a low value of $k$: it provides an upperbound on the solution size. Call such rows *limited-required*. We can use this to bound the branch tree. The next branching rule aims to introduce a limited-required row for a low value of $k$.

**Branching rule:** Limiting Pair
*Consider optional rows $r$ and $i$ where $r$ would be $k$-limited after row $i$ is required. Pick $r$ and $i$ such that $k$ is minimised. Branch in four cases and return the largest solution.*

**Case Require both $i$ and $r$. In this branch, require at most $k$ additional rows.**

**Case Require $i$, Forbid $r$.**

**Case Forbid $i$, Require $r$.**

**Case Forbid both $i$ and $r$.**

**Lemma 4.15.** *The branching rule Limiting Pair is valid.*

*Proof.* The Limiting Pair branch is an exhaustive branch on two rows. By the choice of $i$ and $r$, the row $r$ is limited-required in the "both required" branch. Then by Lemma 4.14 a link independent set can contain at most $k$ additional rows. □

If a limiting pair with low $k$ can be found, the Limiting Pair branching rule is a good rule: three options that reduce the number of *optional* rows by two, and a fourth option with a good bound on the solution size. (We will analyse how good this is in Section 4.7.) If no limiting pair with low $k$ is available, it is not a good branching rule: in the worst case there is effectively no bound, which would lead to an $\mathcal{O}^*(2^n)$ runtime. We will show in Section 5.3 that in random geometric instances there tend to be plenty of good limiting pairs. Also, we have two more rules to help in case no good limiting pairs are available.

In the following, let $R$ be the set of required rows, $S$ the set of safe rows and $n$ the number of optional rows. (Recall that the number of rows labelled *optional* is the measure of the instance, hence the variable name $n$.) Let $0 < c \leqslant \frac{1}{2}$ be a constant to be picked later.

**Reduction rule:** Big
*Let $\mathcal{I}$ be optimal among all $\mathcal{I} \supseteq R$. If $|\mathcal{I}| \geqslant (1-c)n$, then return $\mathcal{I}$ and don't continue branching.*

**Lemma 4.16.** *The reduction rule Big is valid.*

*Proof.* The rule is to return the optimum if it is big. If we continue branching, we only consider supersets of $R$ and therefore cannot find anything better than $\mathcal{I}$; then there is no need to actually do the branching. □

This rule is trivially valid, but checking whether it applies is expensive, depending on $c$. To apply it, we enumerate all sufficiently large supersets of $R$, in $\mathcal{O}^*(\binom{n}{\lfloor cn \rfloor})$ time: try all sets excluding up to $\lfloor cn \rfloor$ of the remaining $n$ optional rows. In the analysis of the runtime we will balance $c$ against several other cases.

In contrast, checking the final rule is computationally trivial while arguing its validity takes some work.

**Reduction rule:** Few Required
*Let all optional rows be safe. If $|R| < c \cdot n$, then return* No.

In order to prove validity, we first provide some bounds. Consider the size $\textsc{Opt}_R$ of the optimal solution in the current branching subtree (that is, the optimum over only supersets of $R$).

**Proposition 4.17.** *If $\textsc{Opt}_R \geqslant |R| + (1-c)n$, then the optimal solution for the current subtree will be found by the Big rule.*

This is literally the statement of the Big rule. Consequently the branching tree is cut off when this condition arises. Recall that $S$ is the set of safe rows.

**Lemma 4.18.** *If all optional rows are safe and $\textsc{Opt}_R < |S|$ then the global optimum is not in the current subtree.*

Before we prove this, note that this case may indeed arise: it can happen that we cannot add as many rows to a solution as there are safe rows. This is because of the *SINR*-conditions of the required rows. Consider the following internal branch state, where rows 1 through 4 are safe and may indeed form the global optimum but can no longer all be taken in the solution because row 5 is already required.

$$M = \begin{array}{l} \left[\begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 2 \end{array}\right] \begin{array}{l} \\ \\ \\ \\ \text{Required} \end{array} \end{array}$$

*Proof of Lemma 4.18.* Consider the currently remaining instance $M$ and the original instance $M_0$. Construct instance $M^+$ by starting from $M_0$, forbidding all rows that were forbidden to arrive at $M$, and also forbidding all rows that are *required* in $M$.

$M^+$ consists entirely of rows in $S$, which by definition are safe in $M$. Then by construction these rows are also safe in $M^+$: there is less interference. Furthermore, there are no required rows in $M^+$ so we can require all of $S$. Then $S$ is a link independent set in $M_0$ and by assumption it is larger than the optimum in the subtree of $M$. This proves the lemma. □

These two lemmata show that, while branching, there is no really big solution (the Big rule would have found it and killed the branch) and, if all *optional* rows are safe, we are not interested in small solutions (they are not globally optimal). We now combine these two observations.

**Lemma 4.19.** *If all optional rows are safe and the global optimum is in the current branching subtree, then $|R| > c \cdot n$.*

*Proof.* Note that $n \leqslant |S|$ since by assumption all *optional* rows are safe. Then combining bounds 4.17 and 4.18 gives

$$n \leqslant |S| < \textsc{Opt}_R \leqslant |R| + (1-c)n$$

Then necessarily $|R| > c \cdot n$. □

We are now ready to prove the validity of the Few Required rule.

**Lemma 4.20.** *The reduction rule Few Required is valid.*

*Proof.* By transposition of Lemma 4.19, if all optional rows are safe and $|R| \leqslant c \cdot n$, then the global optimum is not in the current branching subtree and we can stop. □

This concludes the branching and reduction rules. We now combine them into an exact algorithm for Link Independent Set (Abstract).

## 4.6. Branching algorithm

We have now seen the reduction and branching rules and combine them into a branching algorithm. The algorithm repeatedly applies an applicable rule. Details are provided in Algorithm 1, but broadly speaking the algorithm is as follows.

1. Apply Busted Row where possible.
2. Try a Busted Pair branch.
3. Else, if all *optional* rows are safe
   (a) Try to apply Big

   (b) Try to apply Few Required

   (c) If you get here, there is no good Limiting Pair. Do a Single Row branch

4. Else, do the best Limiting Pair branch.

Recall that the Limiting Pair branch gives an upperbound on possible solution size in one of its branches. This upperbound for a branching subtree is maintained during the branching process and used to prune branches where too many rows are *required*. This is important for our runtime guarantee.

Aside from this local upperbound, we can simultaneously apply a general branch-and-bound technique as follows. We are handling a maximisation problem. Therefore any feasible solution found by the algorithm during branching immediately gives a global lowerbound on the optimum. We can combine this bound with the local upperbound: if they conflict, the global optimum cannot be in this branching subtree. Also, if the number of remaining optional rows in a branching subtree is not enough to extend the set of required rows to reach the lowerbound, then the global optimum cannot be in this branching subtree. We prove no guarantee on the effectiveness of this type of branch-and-bound, but we do use it in our implementation and show in Section 5.3.2 that it is beneficial in practice.

The remainder of this chapter now proceeds as follows. In Section 4.7 we analyse the exponential runtime of Algorithm 1. Then we consider the time spent in individual branch nodes. Let $n_0$ be the size of the original instance, and $n$ the size of the remaining instance being branched on. Checking for the availability of a Busted Pair and finding the best Limiting Pair can both be done trivially in $\mathcal{O}(n^3)$ time: check all pairs of rows in linear time each. It is still fairly easy to improve this to $\mathcal{O}(n^2)$ time using $\mathcal{O}(n_0^2 \log n_0)$ preprocessing time to build a data structure. In Section 4.8 we improve branch-selection time further to $\mathcal{O}(n \log n_0)$ time, using a data structure built with $\mathcal{O}(n_0^2 \log n_0)$ preprocessing time. This approach also improves the total space overhead from $\mathcal{O}(n_0^3)$ to $\mathcal{O}(n_0^2)$: we show that it is not necessary to keep a copy of the current gain matrix in every branch node.

## 4.7. Running time

Crucial in the analysis of the runtime of Algorithm 1 is the quality of the Limiting Pair branches: how good is the bound that they provide? This is formalised as follows.

**Definition 4.21 (Quality of Limiting Pair branch).** Let $M$ be an instance with $n$ remaining optional rows. Let $(r, i)$ be a pair of optional rows such that $r$ is $k$-limited in $M/\text{require}(i)$. Then the *quality* of the Limiting Pair branch on $r$ and $i$

**Algorithm 1:** Branch( M, upperbound )

**Input**: Gain matrix M, its rows labelled Optional or Required. An upperbound on the optimum of M.

**Output**: Updates a global 'best solution so far' $M_{\text{best-yet}}$ (lines 5 and 22). Let $M_0$ be the original instance before any branching. If $M_0$ has an optimum solution consistent with M's forbidden and required rows, then an optimal solution of M is considered for $M_{\text{best-yet}}$.

```
1  if upperbound $\leqslant 0$ then forbid all rows
2  Apply BustedRow to M where possible
3  $O \leftarrow$ the set of optional rows in M
4  if $|O| = 0$ then
5      if all rows in M are safe then update the global $M_{\text{best-yet}}$ if M is larger
6  else if $|O| = 1$ then
7      $r \leftarrow$ the optional row of M
8      Branch( M/require(r), upperbound $- 1$ )
9      Branch( M/forbid(r), upperbound )
10 else ( $|O| \geqslant 2$ )
11     if M has a busted pair then
12         $(r, i) \leftarrow$ a busted pair in M
13         Branch( (M/require(i))/forbid(r), upperbound $- 1$ )
14         Branch( (M/forbid(i))/require(r), upperbound $- 1$ )
15         Branch( (M/forbid(i))/forbid(r), upperbound )
16     else
17         $S \leftarrow$ the set of safe rows in M
18         $(\ell, i) \leftarrow$ the limiting pair in M with best quality
19         if $\ell \in S$ then
20             (Comment: no good limiting pair is available.)
21             if the Big rule applies to M then
22                 Update the global $M_{\text{best-yet}}$ if the Big solution is an
                   improvement
23             else
24                 $R \leftarrow$ the set of required rows in M
25                 if $|R| < c \cdot |O|$ then
26                     Branch( M/require($\ell$), upperbound $- 1$ )
27                     Branch( M/forbid($\ell$), upperbound )
28                 end
29             end
30         else
31             newBound $\leftarrow \min\{$ upperbound $- 2$, ResultingLimit$(M, \ell, i)$ $\}$
32             Branch( (M/require(i))/require($\ell$), newBound )
33             Branch( (M/require(i))/forbid($\ell$), upperbound $- 1$ )
34             Branch( (M/forbid(i))/require($\ell$), upperbound $- 1$ )
35             Branch( (M/forbid(i))/forbid($\ell$), upperbound )
36         end
37     end
38 end
```

is defined as $\frac{k}{n-2}$. When we say that a Limiting Pair with quality $c$ is available, we mean that it has quality $c$ *or better*.

The interpretation of this notion of quality is that it gives the fraction of *optional* rows that could potentially still be Required in $(M/\text{require}(r))/\text{require}(i)$, as bounded by Lemma 4.14. Expanding the definition again, we see that a Limiting Pair branch of quality $c$ gives, in the both-required branch, an upperbound of $\lfloor c \cdot n \rfloor$ on the number of additional rows that we can require. Notice that lower $c$ means better quality.

When the algorithm applies the Limiting Pair rule, it has some quality: applying the rule results in a $k$-limited row, for some specific value of $k$. The algorithm just continues branching using the regular branching rules, using $k$ as an upperbound on the number of further rows to require. This prunes the search tree. The branching can result in no more leaf nodes than exhaustively enumerating all ways to pick at most $\lfloor c \cdot n \rfloor$ additional rows.

**Lemma 4.22 (Limited runtime).** *Let $M$ be an instance with $n$ remaining optional rows and a $(c \cdot n)$-limited required row. If $0 < c \leqslant \frac{1}{2}$ the time spent by Algorithm 1 is $\mathcal{O}^*(\, b^n \,)$, where $b = \frac{1}{c^c(1-c)^{1-c}}$.*

*Proof.* In any leaf node in the search tree, at most $\lfloor c \cdot n \rfloor$ additional rows are required. There are only $\sum_{i=0}^{\lfloor cn \rfloor} \binom{n}{i}$ ways to do this, which directly gives an upperbound on the number of actually-occurring leaf nodes. By Lemma 2.12), this satisfies the bound in the lemma. □

**Corollary 4.23.** *Let $M$ be an instance and $(r, i)$ be a limiting pair of quality $0 < c \leqslant \frac{1}{2}$. The both-required branch takes $\mathcal{O}^*(\, b^n \,)$ time, where $b = \frac{1}{c^c(1-c)^{1-c}}$*

*Proof.* Lemma 4.22 applies to $(M/\text{require}(r))/\text{require}(i)$. □

This means that the computation in the both-required branch in fact takes only $\mathcal{O}^*(\, \binom{n}{\lfloor cn \rfloor} \,)$ time. For the analysis of the total runtime of the algorithm, we will charge this time to the internal branch node where the Limiting Pair branch was taken, with the intent of using Lemma 4.3 ("Internal time"): as long as the time spent locally at each branch node is bounded by the runtime of the remaining branching, it can be added "for free" to the exponential runtime of the algorithm. To this end we will now reinterpret the Limiting Pair branch as a branch in only three options, that spends an additional $\mathcal{O}^*(\, \binom{n}{k} \,)$ time in the branch node itself.

**Branching rule:** Limiting Pair (reinterpreted)
*Let $r$ and $i$ be optional rows and let row $r$ be $k$-limited after row $i$ is Required. Pick $r$ and $i$ such that $k$ is minimised. Solve $(M/\text{require}(i))/\text{require}(r)$ in $\mathcal{O}^*(\, \binom{n}{k} \,)$ time. Branch in three cases:*

**<u>Case</u> Require $i$, Forbid $r$.**

**<u>Case</u> Forbid $i$, Require $r$.**

**<u>Case</u> Forbid both $i$ and $r$.**

**Lemma 4.24.** *The reinterpreted Limiting Pair rule is equivalent to the original Limiting Pair rule, in terms of validity and time bound.*

*Proof.* By the validity of the original branch rule (Lemma 4.15), the original branch case "Require $i$, require $r$, require at most $k$ additional rows" solves $(M/\text{require}(i))/\text{require}(r)$. It solves this case in $\mathcal{O}^*(\binom{n}{k})$ time (Corollary 4.23) and so does the reinterpreted rule. The reinterpreted rule does not change the other three branch cases. □

This reinterpreted Limiting Pair rule consists of three branches that reduce the number of *optional* rows by 2 (for a branching factor of $\sqrt{3}$) and an additional $\mathcal{O}^*(\binom{n}{k})$ time in the branch node itself.

Except for Single Row (which we will handle in a moment), all of our branching rules now have branching factor $\sqrt{3}$. We have, however, introduced $\mathcal{O}^*(\binom{n}{k})$ computation time into some internal branch nodes, for some value $k$ that depends on how the branching works out. If it is always the case that $k < 0.238n$ (that is, if there is always a Limiting Pair branch of quality 0.238 or better) then this extra work is 'free.'

**Lemma 4.25 (Lucky runtime).** *If a Limiting Pair branch of quality* 0.238 *or better is always available, then Algorithm 1 runs in $\mathcal{O}^*(\sqrt{3}^n)$ time.*

*Proof.* Disregarding time spent in internal nodes, the runtime of the algorithm is $\mathcal{O}^*(3^{n/2}) = \mathcal{O}^*(\sqrt{3}^n)$. If the quality of a Limiting Pair branch is always at most $c$, then the time spent in an internal node is $\mathcal{O}^*(b^n)$, where

$$b = \frac{1}{c^c(1-c)^{(1-c)}}.$$

If we want $b \leqslant \sqrt{3}$ and solve for $c$, we requires $c \lesssim 0.238$. If indeed $c < 0.238$, then by Lemma 4.3 the total runtime of the algorithm is $\mathcal{O}^*(\sqrt{3}^n + \sqrt{3}^n)$. □

We may not always be so lucky and therefore a final piece of analysis is needed. Recall that safe rows cannot become limited-required with quality better than the trivial 1: safe means that even all interferences together cannot be a problem, so this row is not going to give an upperbound. This means in particular that if all *optional* rows are safe, then there can be no good Limiting Pair branch. If furthermore no reduction rules apply, we have no better option that to use the Single Row branching rule. On its own, it would lead to $\Omega^*(2^n)$ runtime. Now we analyse some interplay with the Big and Few Required rules.

We look at the case where either there is a good Limiting Pair, or otherwise all *optional* rows are safe. In that situation, if there is no good Limiting Pair available, there need to be at least $c \cdot n$ required rows already: if there aren't, the Few Required rule would have been applied. Until that point we have only used the reduction rules and the Pair branching rules (Busted Pair and Limiting Pair). That means that for every *required* row in the instance, there was also a row to which the forbid operation was applied: the reduction rules only forbid rows or conclude No, and the Pair branch rules always forbid a row whenever they require a row. Then the measure of the instance has already been decreased by at least $2|R| = 2 \cdot c \cdot n$, exclusively by rules with the good branching factor $\sqrt{3}$ and reduction rules (even better).

**Lemma 4.26.** *Assume the following holds in all internal branch nodes: either it has a Limiting Pair branch of quality* $c$*, or all its optional rows are safe. Then, disregarding time spent in internal branching nodes, Algorithm 1 runs in* $\mathcal{O}^*(\, b^n \,)$ *time where*

$$b = \sqrt{3}^{\frac{2c}{1+2c}} \cdot 2^{\frac{1}{1+2c}}.$$

*Proof.* Let $M$ be the original instance and $|M|$ its size, that is, the number of *optional* rows we start out with. By assumption, Single Row branches only happen when all *optional* rows are safe and also $|R| > c \cdot n$. This combined condition does not hold for the branch root (since there $|R| = 0$); in fact, some amount of branching must have taken place before it can hold. We will give a bound $n^*$, depending on $|M|$, such that a Single Row branch can only happen when the measure of the instance has dropped below $n^*$. To bound the worst case running time of the algorithm, we say that the Single Row branch is taken each and every time the measure is below $n^*$: above $n^*$ we have branching factor $\sqrt{3}$ (necessarily) and below $n^*$ we have branching factor 2 (pessimistically). The runtime is then

$$\mathcal{O}^*(\ \sqrt{3}^{|M|-n^*} \ \cdot\ 2^{n^*}\ ).$$

Now we see how big $n^*$ can be, subject to the following two conditions. Firstly $|R| \geqslant c \cdot n$, otherwise the branch is killed by the Few Required rule. Secondly, $n^* + 2|R| \leqslant |M|$: the number of *required* rows, forbidden rows and remaining *optional* rows cannot exceed the original instance size. In the worst case no reduction rules were applied and $n^* + 2|R| = |M|$. Then solving

$$\textbf{maximise } n^* \text{ s.t.}$$
$$|R| < c \cdot n^*$$
$$n^* + 2|R| = |M|$$

gives $n^* = \frac{|M|}{1+2c}$. The resulting runtime is

$$\mathcal{O}^*(\ \sqrt{3}^{|M|-\frac{|M|}{1+2c}} \ \cdot\ 2^{\frac{|M|}{1+2c}}\ )$$
$$= \mathcal{O}^*(\ \left(\sqrt{3}^{\frac{2c}{1+2c}} \ \cdot\ 2^{\frac{1}{1+2c}}\ \right)^{|M|}\ )$$

This proves the lemma. □

**Theorem 4.27.** *Assume all internal branch nodes either have a Limiting Pair branch of quality* $c = 0.333$*, or have no unsafe optional rows. Then Algorithm 1 has runtime* $\mathcal{O}(\, 1.888^n \,)$ *by picking* $c = 0.333$ *in the Big and Few Required rules.*

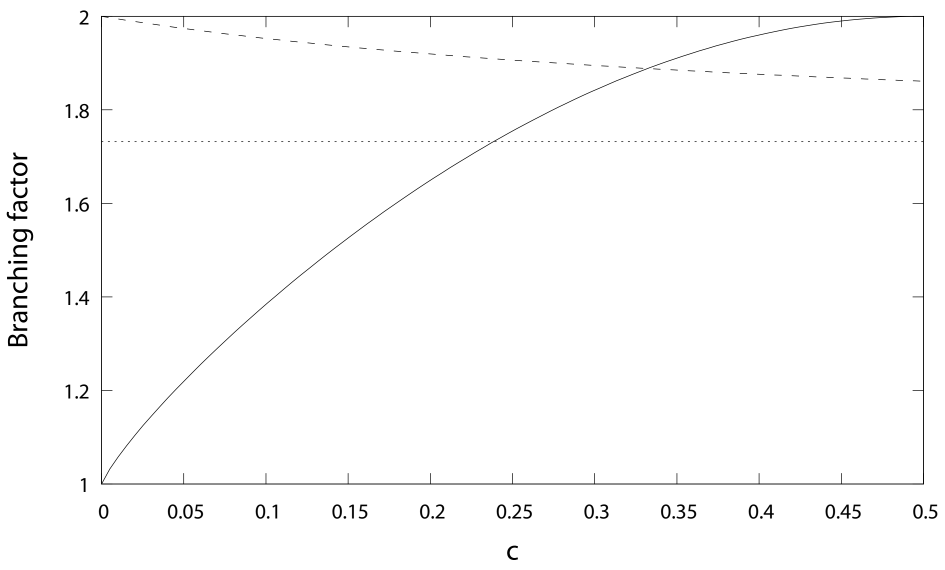


**Figure 4.1.** Picking c to balance the runtime of rule Big versus the branching factor of the algorithm. Solid line: $b_{\text{int}} = \frac{1}{c^c(1-c)^{(1-c)}}$. Dashed line: $b_{\text{branch}} = \sqrt{3}^{\frac{2c}{1+2c}} \cdot 2^{\frac{1}{1+2c}}$. Dotted line: the baseline $\sqrt{3}$.

*Proof.* We continue from the preceding lemma and now also account for the time spent in internal branch nodes. We will use the parameter c to balance this with the branching factor. By Lemma 4.24 the computation in internal branch nodes takes $\mathcal{O}^*(\binom{n}{\lfloor cn \rfloor})$ time, which is exponential with base

$$b_{\text{int}} = \frac{1}{c^c(1-c)^{(1-c)}}.$$

By Lemma 4.26 the regular branching takes exponential time with base

$$b_{\text{branch}} = \sqrt{3}^{\frac{2c}{1+2c}} \cdot 2^{\frac{1}{1+2c}}.$$

Then the internal time lemma (Lemma 4.3) shows that the total runtime is

$$\mathcal{O}^*(\, {b_{\text{int}}}^n + {b_{\text{branch}}}^n \,).$$

This bound is minimised by picking c to minimise

$$\max\{\, \frac{1}{c^c(1-c)^{(1-c)}},\ \sqrt{3}^{\frac{2c}{1+2c}} \cdot 2^{\frac{1}{1+2c}} \,\}.$$

Solving numerically gives $b_{\text{int}} = b_{\text{branch}} \approx 1.888$ by picking $c \approx 0.333$. This optimisation problem is illustrated in Figure 4.1. □

This completes the proof of Theorem 4.1.

## 4.8. Improved polynomial factors: branch selection

An important step in Algorithm 1 is the selection of the pair of rows to branch on. The pseudocode for Algorithm 1 mentions the procedures FindBustedPair and BestLimitingPair. With $\mathcal{O}(n^2 \log n)$-time preprocessing these can fairly simply be implemented to run in $\mathcal{O}(n^2)$ time and $\mathcal{O}(n^2)$ space per branch node. (Note that this means cubic space in total because of linear branching depth.)

Using the same amount of preprocessing time, we will now implement the branch selection in $\mathcal{O}(n \log n_o)$ time per branch node, where $n_o$ is the size of the original instance. Furthermore, we use only $\mathcal{O}(n^2)$ space in total instead of per branch node. First we sketch the high level procedure, then we describe a data structure to make it run fast.

First of all note that if we want to spend $o(n^2)$ time in branching nodes (disregarding, of course, the occasional expensive Big rule), then we cannot store the gain matrix and copy it for the various child nodes: just the copy itself would take $\Theta(n^2)$ time. To achieve $\mathcal{O}(n \log n_o)$ time we will work with a single data structure that we modify when going into branches and unmodify coming out of branches; the stack of modifications will take only linear space.

To implement FindBustedPair$(M)$ we check each row of the matrix. If on row $r$ there exists an interference in column $i$ that is higher than the signal, we immediately choose those rows $r$ and $i$ for a Busted Pair branch. We check this by finding the maximum interference on row $r$. The data structure will support this by a procedure MaxInterference$(r)$ running in $\mathcal{O}(\log n_o)$ time.

In the procedure BestLimitingPair$(M)$ we check for each row $r$ how well we can limit it. To do so, we find the largest interference on row $r$, subtract it from the signal and count how many other interferences we can fit in the remaining signal. For this, the data structure provides a procedure CountLimit$(r, t)$ that returns the largest $k$ such that the sum of the smallest $k$ interferences on row $r$ is at most $t$. It does so in $\mathcal{O}(\log n_o)$ time. We iterate over the rows and keep a running minimum of this value. When we are done, we know the most limiting pair to return: the row $r$ with lowest result from CountLimit and the row $i$ responsible for the highest interference on that row.

Lastly the data structure must support the require and forbid operations on rows. The Forbid operation deletes an entire row and the corresponding column. The Require operation sets all interferences in a column to 0 after subtracting the present interference from the row's signal. The data structure supports both these operations in $\mathcal{O}(\log n_o)$ time per row, for a total of $\mathcal{O}(n \log n_o)$ time. Since we are branching with only a single instance of the data structure, it also supports 'unrequire' and 'unforbid' operations in the same time.

**Theorem 4.28.** *Branch selection can be done in $\mathcal{O}(n \log n_o)$ time using a data structure that can be built $\mathcal{O}(n_o^2 \log n_o)$ time, where $n_o$ is the size of the original instance and $n$ is the size of the remaining instance being branched on. The data structure uses $\mathcal{O}(n_o^2)$ space.*

**Algorithm 2:** MaxInterference( $v$ )

```
Input: Tree node $v$.
Output: The maximum value over leafs descendant from $v$.
1  if IsLeaf( $v$ ) then return Value( $v$ )
2  else if Count( Right($v$) ) $\geqslant 1$ then
3      return MaxInterference( Right($v$) )
4  else
5      return MaxInterference( Left($v$) )
6  end
```

**Algorithm 3:** CountLimit( $v$, $t$ )

```
Input: Tree node $v$, number $t$.
Output: Maximum $k$ such that the sum of the smallest $k$ leaf values
        descendant from $v$ is smaller than $t$.
1  if IsLeaf( $v$ ) then
2      if Value( $v$ ) $< t$ then return 1
3      else return 0
4  else
5      if $t <$ Sum( Left($v$) ) then
6          return CountLimit( Left($v$), $t$ )
7      else
8          return
           Count( Left($v$) ) + CountLimit( Right($v$), $t -$ Sum(Left($v$)) )
9      end
10 end
```

*Proof.* For every row in the matrix, we store the *signal* and *interferences* independently. At initialisation, we build for every row a binary search tree of height $h = \mathcal{O}(\log n_0)$ containing the interference values in its leafs. In the internal tree nodes we store

- a Count field indicating the number of leaf descendants it has, and
- a Sum field indicating the sum of the values of its leaf descendants.

These trees (which at this point might resemble Fenwick trees [Fen94], before we add additional augmentation) can be constructed simply in $\mathcal{O}(n \log n)$ time each, for a total of $\mathcal{O}(n^2 \log n)$ time. It is a simple exercise to implement MaxInterference and CountLimit to run in $\mathcal{O}(h)$ time on these trees. There is a small twist because of deletions, which we will discuss next, but they pose no real complication. See Algorithms 2 and 3 for details.

This leaves the deletion of rows and of columns. (The Require operation is a slight variation on column deletion and we omit the details.) To delete a column we need to be able to find, in the various trees, the leaf nodes corresponding to a certain column. To support this, we build (at initialisation time) for every column a doubly linked list of the tree leaf nodes corresponding to this column. This allows us to find all the leaf nodes in the other trees belonging to the same column as a certain leaf node we want to remove.

When removing interferences from a row, that is, when removing a leaf node from a tree, we do not actually delete the node from the tree structure. This is to simplify the later 'undeletion' of the node. Instead, we mark the node as removed and in $\mathcal{O}(h)$ time update the $\mathrm{Sum}$ and $\mathrm{Count}$ fields of the appropriate internal nodes to reflect only the remaining leafs. The MaxInterference and CountLimit procedures are easily adapted to cope. (For example, don't just check whether children exist, but check whether their $\mathrm{Count} \geqslant 1$.) Undeletion is then a simple case of updating $\mathrm{Sum}$ and $\mathrm{Count}$ fields to reflect the presence of the node, again in $\mathcal{O}(h)$ time. Note that there are no arbitrary insertions, only undeletions.

The implementation advantage of these 'virtual' deletions is that we work with a structurally static tree and as such no memory management is involved after the preprocessing. This does mean that $h = n_0$ instead keeping a balanced tree of the remaining leafs with height $\mathcal{O}(\log n)$.

We have just argued that deletion of a single interference from a row runs in $\mathcal{O}(\log n_0)$ time. We traverse a doubly linked list to find the tree leafs to delete. The deletion of an entire column then takes $\mathcal{O}(m + n \log n_0)$ time, where $m$ is the number of items in the doubly linked list: $\mathcal{O}(m)$ time to traverse the list and $\mathcal{O}(\log n_0)$ time for each of the $n$ actually remaining rows. The way we delete rows will ensure that $m = n$.

To remove a row, we visit all live leafs in its tree (in $\mathcal{O}(n \log n_0)$ time[4]) and *bypass* them in the doubly linked list that they are in. When undeleting the row, we unbypass the leafs. This is correct since, by the structure of the branching algorithm, the unbypass operations will occur in reverse order of the bypass operations. (This is a procedure called 'dancing links' by Knuth [Knu00], who traces it back to Hitotumatu and Noshita [HN79].) These bypasses ensure that for all columns still in the instance, the linked lists only contain the rows that are still in the instance. □

## 4.9. Concluding remarks

In this chapter we have provided an exact algorithm for Link Independent Set (Abstract). Its branching rules are valid for arbitrary gain matrices, but designed to take advantage of the structure available in geometric instances. We have proven a worst-case bound of $\mathcal{O}^*(1.888^n)$ on the runtime of Algorithm 1, but only under an assumption on the input. In the Chapter 5 we will experi-

[4]This upperbound suffices for our purposes and is simply seen to hold. With some work it can be shown that the time taken is $\Theta(n \log(\frac{n_0}{n}))$ in the worst case.

mentally demonstrate that this assumption is reasonable for random geometric instances and that, in fact, the effective branching factor of Algorithm 1 is expected to be much better than 1.888 on such instances.

# 5 Link Independent Set: Properties of random geometric instances

In the previous chapter we have developed an exact algorithm for Link Independent Set (Abstract). In this chapter we continue our investigation of the problem. Using an implementation of Algorithm 1, and a formulation as an integer linear program, we investigate the properties of random geometric instances of the problem. To this end, we first define two classes of random instances. On these, we see that our algorithm performs well: the high-quality Limiting Pair branches that are crucial for its runtime are often available. In fact, we see that the effective branching factor of the algorithm is much better in practice than we were able to prove in the worst case.

We also investigate the size of optimal solutions as it depends on the parameters of the random instances, particularly the density of the point set. We observe some clear trends in the experimental data, and finish the chapter by proving some of these properties.

## 5.1. Introduction

We first give an overview of what we will do in this chapter. We have implemented Algorithm 1 from the previous chapter. Using this implementation we show that the algorithm performs well in practice on random geometric instances. We will show that this is due to both the availability of high-quality Limiting Pair branches and the general effect of branch and bound. More precisely, we will argue the following in Section 5.3.

*Experimental Result 5.1.* While branching on random geometric instances of Link Independent Set (Abstract), the *Limiting Pair* branches nearly always have quality $c \lesssim 0.4$ unless all *optional* rows are safe. Many *Limiting Pair* branches are much better and make up for the small number of bad branches.

*Experimental Result 5.2.* The full set of rules from Algorithm 1 significantly improves the runtime, compared to a basic version that just uses the Single Row rule. This effect is most pronounced when the density of the instance is high, that is, the instance has many links in a small area.

*Experimental Result 5.3.* Adding a general-purpose branch and bound rule to Algorithm 1 significantly improves the runtime, compared to a version without it. This effect is most pronounced when the density of the instance is low, that is, the instance has few links in a large area.

Besides wall-clock time measurements we also investigate the so-called *effective* branching factor [RN10], and obtain the following experimental result.

*Experimental Result 5.4.* On random geometric instances, Algorithm 1 has effective branching factor $b_e \lesssim 1.2$.

Furthermore we experimentally investigate the effect of the parameters of our random model (number of nodes, size of area) on the size of the solution. We show that this effect is well-behaved and substantiate this experimental claim by proving the following two probabilistic statements. The first one says that 'very large' link independent sets are unlikely to exist. Let $\varphi$ be an upperbound on the fatness of reception zones, that is, the area in which a certain transmission can be successfully received (see Definitions 2.15 and 5.8).

**Theorem 5.5.** *Consider a binomial point process instance with $n$ points on a torus of measure $a$, and let $k < n$. If*

$$k > \frac{a\varphi^2}{\pi(1 - \sqrt[n]{1-p})},$$

*then the probability that the instance has a link independent set of cardinality $k$ is at most $p$. This lowerbound on $k$ is $\Omega\left(\frac{a \cdot n}{-\ln(1-p)}\right)$.*

Secondly we show that a 'quite large' link independent set does exist with high probability.

**Theorem 5.6.** *In Poisson instances of density $n/a$ on a square of measure $a$, or a torus with square area of measure $a$, with high probability a link independent set exists of size $\Theta\left(\sqrt{\frac{n \cdot a}{\log a}}\right)$.*

## 5.2. Two models of random geometric instances

We will now define a binomial and a Poisson model of random Link Independent Set instances. These will provide insight into the combinatorial properties of the problem and the behaviour of our algorithm.

Both models distribute senders and receivers in a bounded region $A$. Let this region have measure $a$. We perform our computational experiments on rectangular regions. A complicating aspect of rectangles is their boundary: these lead to different conditions for points near the edge versus points in the middle of the region, since points near the edge have fewer nearby neighbours. We are not necessarily interested in these effects per se, so in our probabilistic analyses we will look instead at the case where the region is a rectangle that wraps around as a torus. An intuitive way to interpret this is as a region within a larger sensor network: in terms of surrounding nodes, there are no boundary effects.

We use a two-step procedure for generating a random geometric instance. First we place the senders at points $s_i$ and then we pick the locations $r_i$ for the receivers. We do this by picking $r_i$ uniformly at random from the unit disc centered at $s_i$. This provides a locality of communication that is reasonable from an application perspective. Note that as $A$ increases in size, the relative size of this unit disc decreases. In this way, the *density* of senders in $A$ is important. The proofs in the toroidal model assume that $A$ is sufficiently large in the sense that small balls indeed have quadratic area and in particular that unit discs have area $\pi$.

We pick the locations of the senders either using a uniform binomial point process or alternatively using a Poisson point process of uniform density.[5] We will call these 'the binomial model' and 'the Poisson model' respectively. In the former we set the number of points to $n$ and end up with a density of $\lambda = n/a$. In the latter we set the density to $\lambda$ and end up with, on expectation, $n = \lambda \cdot a$ points. In this sense, the models are related. To be more precise, Poisson instances can be simulated by first sampling $n$ from a Poisson distribution and then taking a binomial instance with that value of $n$.

See Figure 5.1 for an example instance that can be produced by this two-step process. The senders are indicated using $\bullet$ and, in the drawing, surrounded by a grey unit disc. Their corresponding receiver is picked uniformly at random from this disc and indicated using $\circ$. The correspondence is indicated using an arrow. Note in particular the wrap-around effect of working on a torus.

## 5.3. Computational experiments

In this section we report the results of our computational experiments. The data were obtained using our implementation of Algorithm 1 from Chapter 4. The

[5]See Preliminaries, Section 2.3 on page 19.

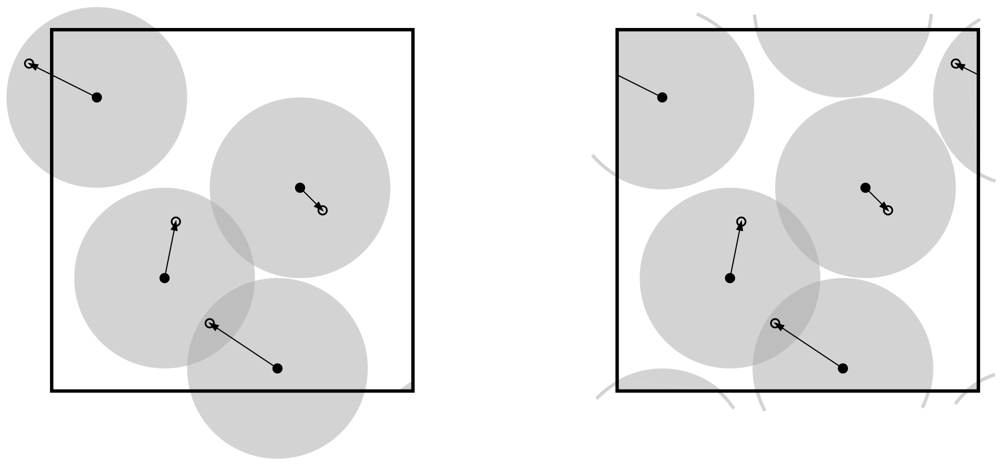

**(a)** Rectangular area **(b)** Toroidal area

**Figure 5.1.** A geometric instance with $n = 4$ senders and receivers. The senders are indicated using $\bullet$ and surrounded by a grey unit disc. Their corresponding receiver is indicated using $\circ$ and an arrow. Note the wrap-around effect of working on a torus.

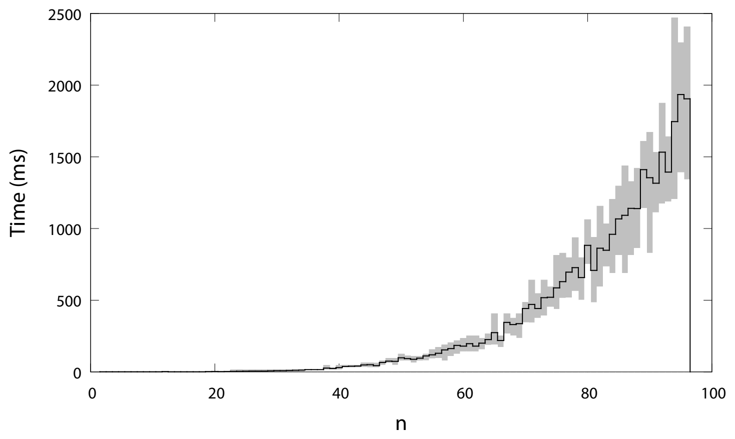
implementation is in C++, is single-threaded and was run on an Intel® Core™ 2 Quad processor at 2.4 GHz.

Here we have only used binomial instances, as these have a fixed value for $n$ and this is a variable we are interested in; using Poisson instances instead would only introduce noise into our data. We have used rectangular regions in these experiments: this is after all a more realistic setting than a torus and easily accomplished in an experiment. This does mean that there might be boundary effects.

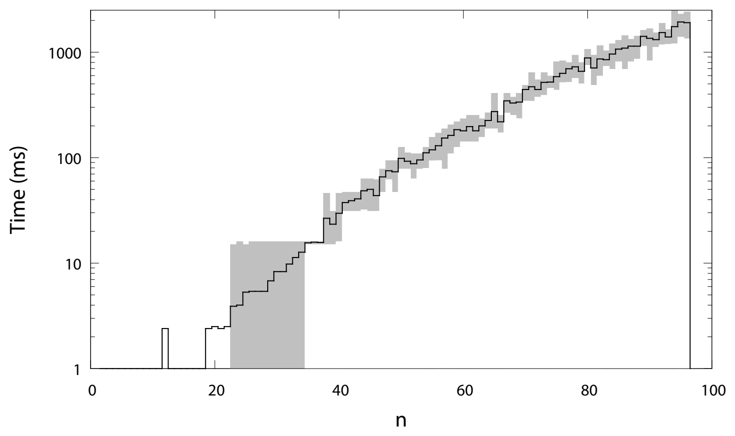
### 5.3.1. Algorithm variants

In order to better understand the runtime of Algorithm 1 we compare several variations. We mentioned in Chapter 4 that one can apply the following general-purpose branch and bound procedure. For a maximisation problem, any feasible solution found by the branching algorithm gives a global lowerbound on the optimum; if some branching subtree does not have enough optional rows remaining to reach this lowerbound, the branching subtree can be pruned because it does not contain the global optimum. We did not include this in the statement of Algorithm 1 and the analysis of its worst-case runtime, but we can certainly see its effect experimentally. This gives the first variation: *with* the branch and bound rule or *without* it.

We get two more variations by considering the branching and reduction rules: if we only use the branching rule for single rows, we get the trivial brute



**(a)** Linear-scale plot



**(b)** Semi-log plot

**Figure 5.2.** Runtime of Algorithm 1 on instances on the region $[0, 1]^2$ as the number of links increases. Data are from 10 random instances for every value of $n$. Average is indicated in black; grey area indicates the full range of results, omitting for every $n$ the minimum and maximum values as potential 'outliers.'

force algorithm, which we will call *enumeration*. This variant can also be done either with or without branch and bound. This gives four algorithm variants in total.

### 5.3.2. Performance

In the first experiment, we look at a fixed region $A = [0, 1]^2$ and vary the number of links, ranging from 2 to 95; see Figure 5.2. As expected, the runtime increases exponentially in $n$. We note that even for $n = 95$, these instances are solved in a couple of seconds each.

Figure 5.3 shows a histogram of runtimes for the four algorithm variants on 1400 random instances with $n = 22$ and $A = [0, 5]^2$. We see that both the branching rules and the branch and bound improve the runtime.

The relative importance of these two aspects depends on the density of the instance. Again with 22 links, but in the smaller region $[0, 2]^2$, good runtime depends almost purely on the branching and reduction rules (see Figure 5.4): a smaller region leads to stronger interference and the rules exploit this. This is Experimental Result 5.2. Meanwhile, it is unlikely that there is a large link independent set to help the branch and bound; indeed, branch and bound was almost useless in this setting.

The effect is reversed on the larger region $[0, 15]^2$, where good runtime depends almost purely on branch and bound: see Figure 5.5. In this setting, the rules have only weak interference to work with. Branch and bound is massively effective, however, because a large solution is quickly found. This is Experimental Result 5.3. As we will see later on, instances with low density $n/a$ are indeed likely to have large solutions.

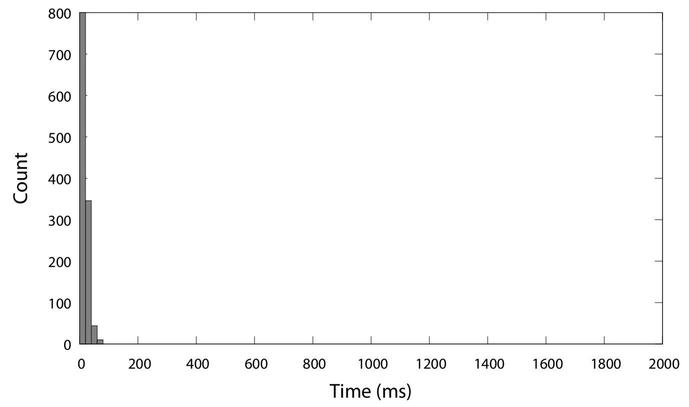
Finally, we consider the effective branching factor of Algorithm 1: in practice it is much better than the worst case of 1.888 that we arrived at in the previous chapter. Figure 5.6 shows histogram of effective branching factors on 735 random instances of moderate density as before: $n = 22$ and $A = [0, 5]^2$. For these settings, we see that without branch and bound we get an effective branching factor around 1.5: already a significant improvement. For the full algorithm with branch and bound we see that the effective branching factor is below 1.1. These results are somewhat affected by the density of the instance, but in general the effective branching factor remains quite good. This is Experimental Result 5.4.

We have also confirmed Experimental Result 5.1: the quality of *Limiting Pair* branches is almost always better than approximately 0.4 unless all optional rows are safe. This is a little worse than is required to always get the worst-case runtime bound from Theorem 4.1. However, as we see from the effective branching factor, the occasional bad branch is more than offset by many good branches.

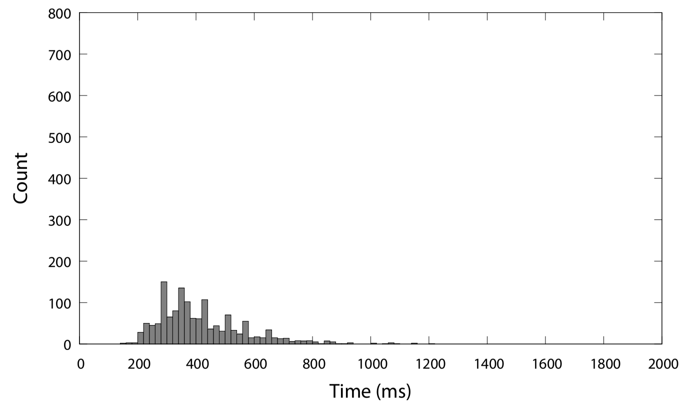
### 5.3.3. Solution size

Now we look at the *cardinality* of optimal link independent sets. In the first experiment, we fix $n$ and look at an increasingly large region $[0, L]^2$, with $L$ ranging from 1 to 100: see Figure 5.7. The data show that as the region becomes larger, the optimum increases until eventually it is highly likely that the entire instance is independent. This is reasonable, since in a large region the links are short relative to the (expected) distance between the senders, which makes significant interference unlikely. Recall that, in this case, the branching algorithm finds the optimum quickly, as it will find a large feasible solution early on and benefit significantly from branch and bound.

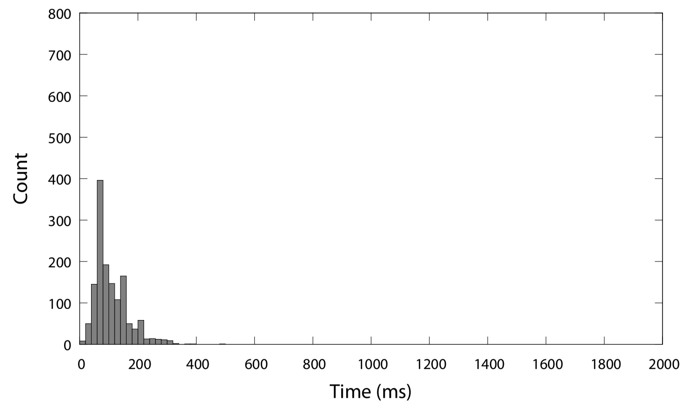
Secondly, we look at this relation from the other direction: a fixed region with an increasing number of links, with $n$ ranging from 2 to 75: see Figure 5.8. In this experiment we see that as long as the number of links increases, so does the size of the optimum. The asymptotic behaviour is unclear from this range of



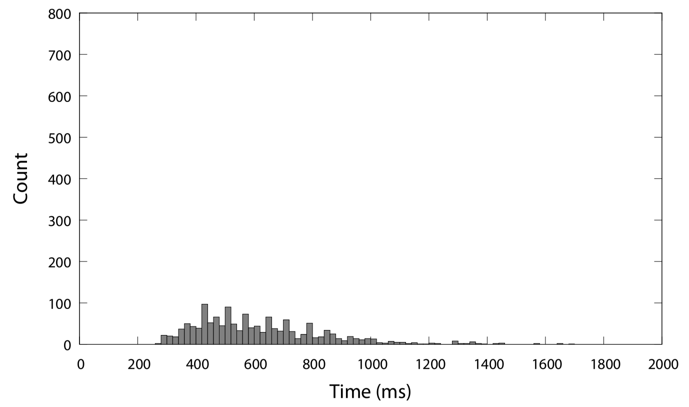
**(a)** Algorithm 1 with branch and bound



**(b)** Enumeration with branch and bound

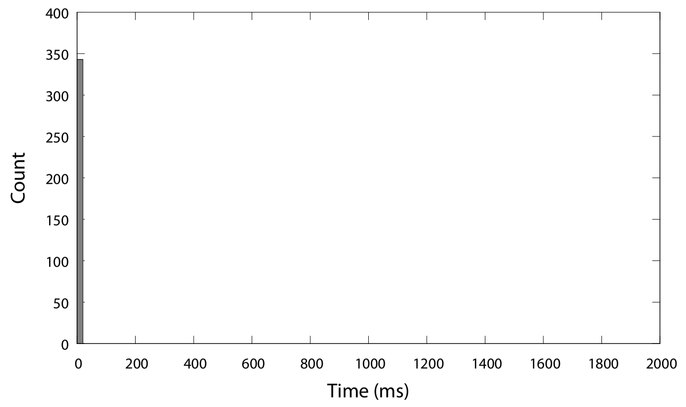


**(c)** Algorithm 1 without branch and bound

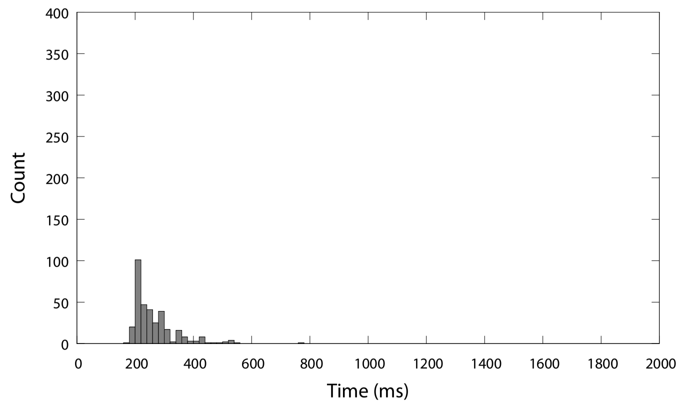


**(d)** Enumeration without branch and bound

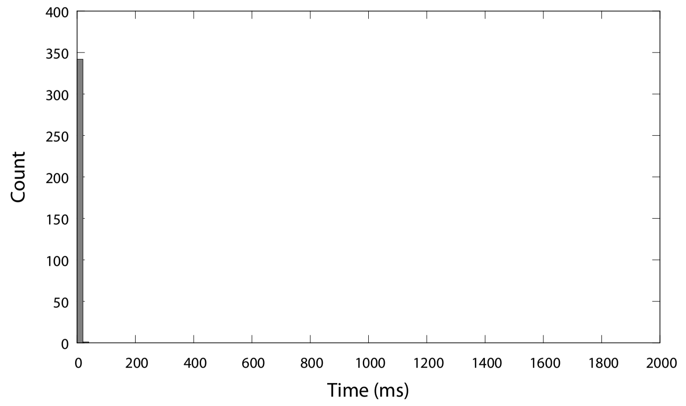
**Figure 5.3.** Variations of Algorithm 1: histogram of runtimes based on 1400 random instances with $n = 22$ and $A = [0, 5]^2$. Good runtime depends on the rules as well as branch and bound.



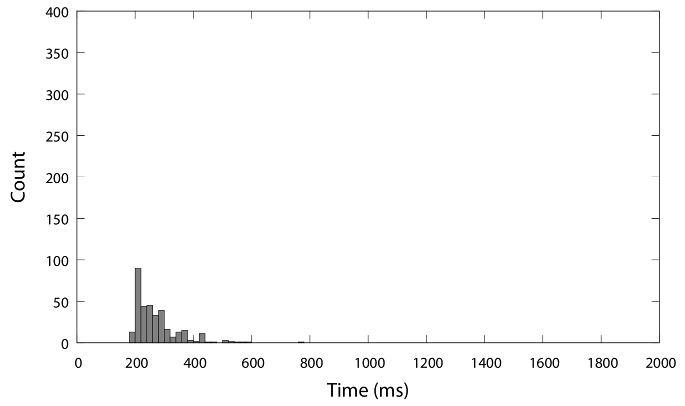
**(a)** Algorithm 1 with branch and bound



**(b)** Enumeration with branch and bound

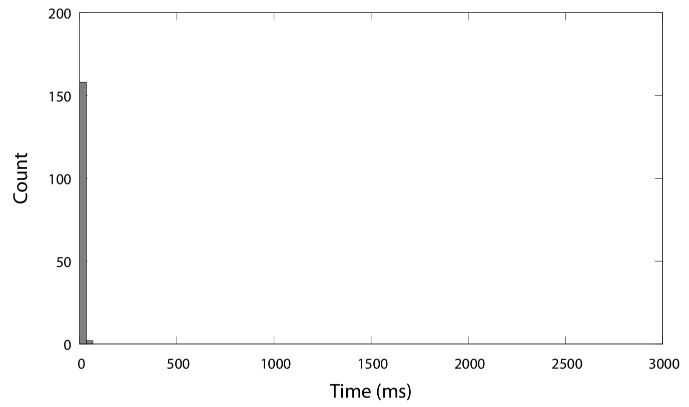


**(c)** Algorithm 1 without branch and bound

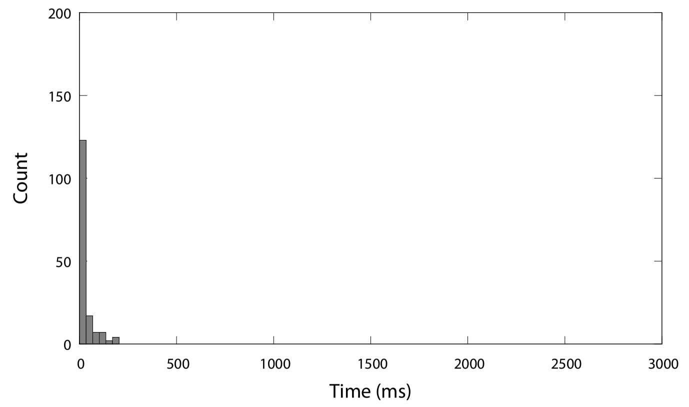


**(d)** Enumeration without branch and bound

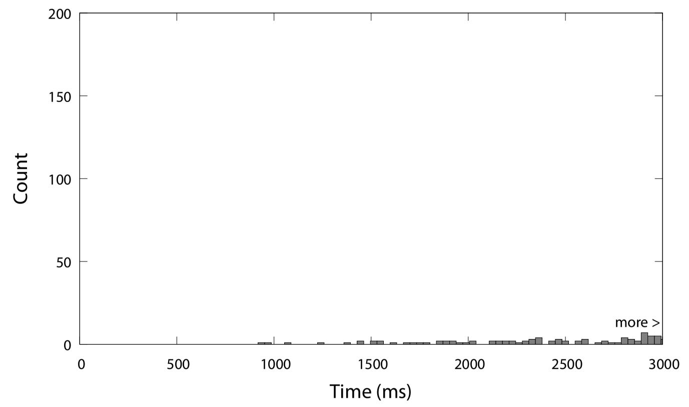
**Figure 5.4.** Variations of Algorithm 1: histogram of runtimes based on 340 random instances with $n = 22$ and $A = [0, 2]^2$. When the density is high, good runtime depends mostly on the branching and reduction rules.



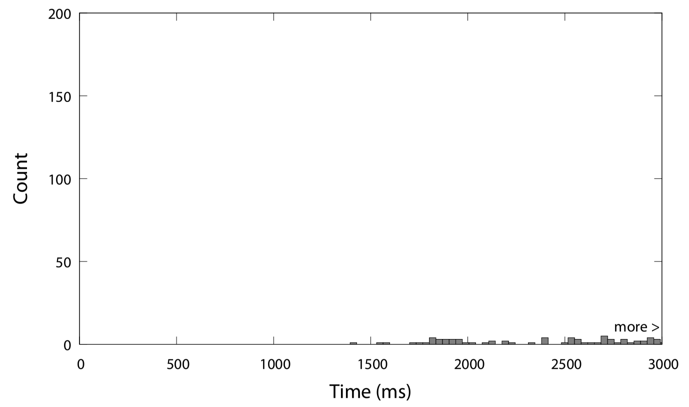
**(a)** Algorithm 1 with branch and bound



**(b)** Enumeration with branch and bound



**(c)** Algorithm 1 without branch and bound



**(d)** Enumeration without branch and bound

**Figure 5.5.** Variations of Algorithm 1: histogram of runtimes based on 160 random instances with $n = 22$ and $A = [0, 15]^2$. When the density is low, good runtime depends mostly on branch and bound.

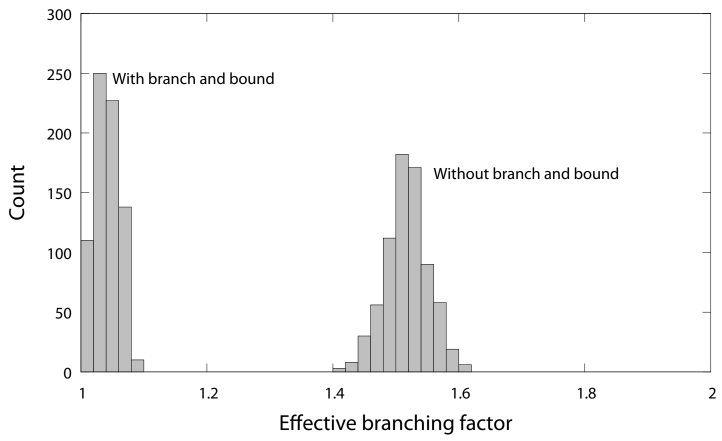


**Figure 5.6.** Histogram of the effective branching factors of Algorithm 1 on 735 random instances with $n = 22$ and $A = [0, 5]^2$, with and without branch and bound.

$n$, however. Unfortunately, experiments with significantly larger $n$ are prohibitively expensive in terms of runtime.

We can qualitatively argue that as $n$ goes to infinity, the size of the optimum becomes arbitrarily large. As $n$ grows, we expect more links of very short length. These links have a strong signal, and the stronger the signal, the more densely it is possible to pack successful transmissions. Indeed, from Theorem 5.6 (which we will prove in Section 5.4.5) we get that for a region of constant measure, the optimum should grow as $\Omega(\sqrt{n})$. This seems consistent with Figure 5.8.

## 5.4. Properties of random geometric instances

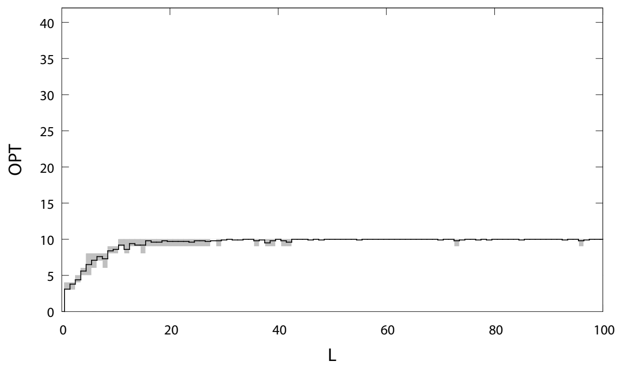
In this section we investigate theoretically some of the properties we have already observed experimentally. We will prove the two theorems mentioned in the introduction. One theorem states that 'very large' link independent sets are unlikely; the other that 'somewhat large' link independent sets *are* indeed likely.

### 5.4.1. Fatness of reception zones

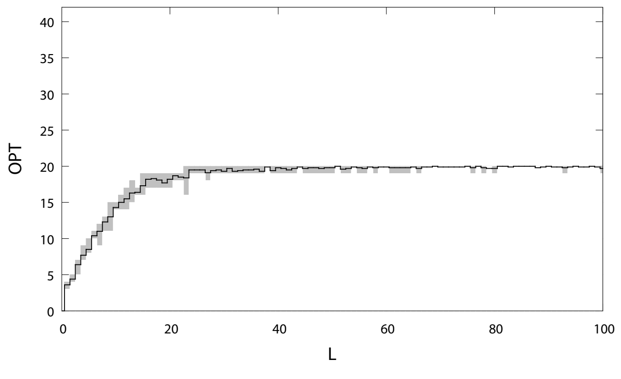
We will use a packing argument based on so-called reception zones to show that large link independent sets are unlikely. Recall that the *reception zone* of sender $i$ is defined as the set of points in the plane at which it can be heard. Division by distance zero adds some technical tedium to the following definition, but the concept is clear (see Definition 2.15).

**Definition 5.7 (Reception zone).** The reception zone of sender $i$ is

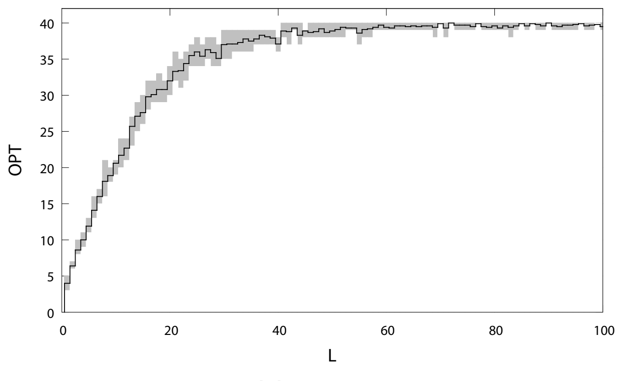
$$\mathcal{H}_i = \{p \in \mathbb{R}^2 - S \mid \mathit{SINR}(s_i, p) \geqslant \beta\} \cup \{s_i\}.$$



**(a)** $n = 10$

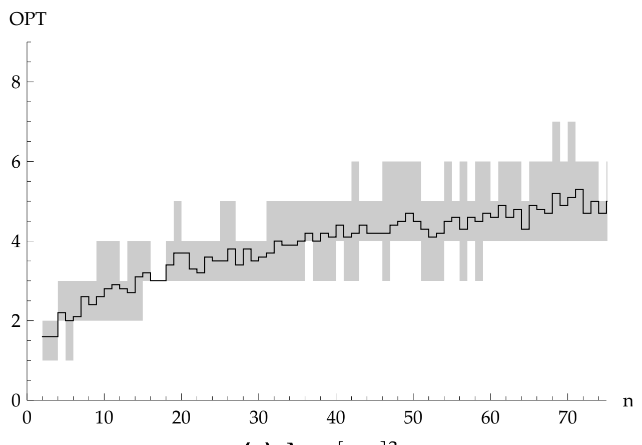


**(b)** $n = 20$

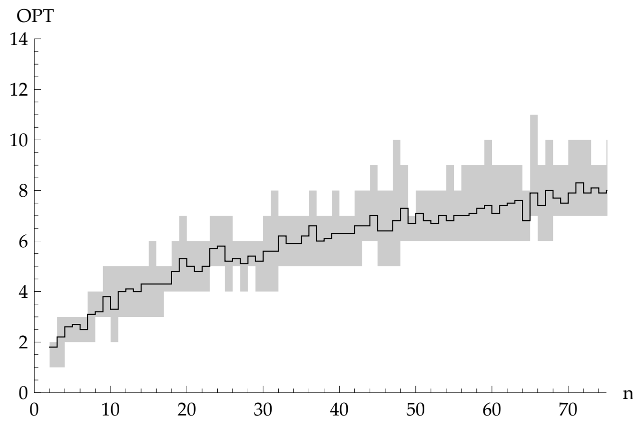


**(c)** $n = 40$

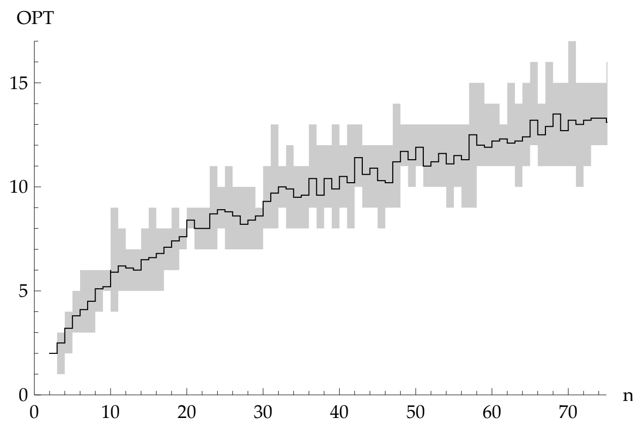
**Figure 5.7.** Size of the optimum solution for a fixed number of input links as the size of the region increases: $A = [0, L]^2$. Data are from 10 random instances for every value of $L$. Average is indicated in black; grey area indicates the full range of results, omitting for every $L$ the minimum and maximum values as potential 'outliers.'



**(a)** $A = [0, 1]^2$



**(b)** $A = [0, 2]^2$



**(c)** $A = [0, 4]^2$

**Figure 5.8.** Size of the optimum solution in a region of fixed size as the number of links increases. Data are from 10 random instances for every value of $n$. Average is indicated in black; grey area indicates the full range of results, omitting for every $n$ the minimum and maximum values as potential 'outliers.'

There exist several non-equivalent definitions of geometric *fatness*. We follow the definition that Avin et al. use [AEK+12], where a *zone* is a subset of the plane (subject to some reasonable, but technical, 'niceness' constraints).

**Definition 5.8 (Fatness).** Let $p$ be a point and $Z$ a zone. Furthermore, let $B(p, r)$ be the ball of radius $r$ around $p$. Then

$$\begin{aligned} \delta(p, Z) &= \sup\{r > 0 \mid B(p, r) \subseteq Z\}, \\ \Delta(p, Z) &= \inf\{r > 0 \mid Z \subseteq B(p, r)\}, \\ \varphi(p, Z) &= \frac{\Delta(p, Z)}{\delta(p, Z)}. \end{aligned}$$

That is, $\delta$ is the radius of the largest ball centered at $p$ and contained in $Z$. Similarly, $\Delta$ is the radius of the smallest ball centered at $p$ that contains $Z$. Finally $\varphi$ is the ratio of these two. A zone $Z$ is called *fat* for a point $p$ if $\varphi(p, Z)$ is bounded by a constant.

As noted before, we consider instances where all senders transmit with equal power. Such instances are called *uniform-power networks*. Under these definitions, the following theorem is known.

**Theorem 5.9 (Avin et al. [AEK+12]).** *The reception zones of a uniform-power network with path-loss parameter $\alpha > 2$ and reception threshold $\beta > 1$ are convex and fat.*

In particular, reception zone $\mathcal{H}_i$ is fat for $s_i$.

### 5.4.2. Lowerbound on area of reception zones

Consider a link independent set $\mathcal{I}$, that is, a subset of $S$ in which all *SINR* conditions are satisfied. Being link independent requires that every sender $s_i$ in this set $\mathcal{I}$ must be heard at the corresponding receiver. Recall that $d_i$ is defined as the distance between sender $s_i$ and the corresponding receiver $r_i$. Let $\Delta_i = \Delta(s_i, \mathcal{H}_i)$. By definition of $\Delta$, and because $r_i \in \mathcal{H}_i$, we get

$$\Delta_i \geqslant d_i. \tag{5.1}$$

By Theorem 5.9 we have that reception areas are fat. Let $\varphi$ be an upperbound on the fatness of reception zones and let $\delta_i = \delta(s_i, \mathcal{H}_i)$. Then we directly get that

$$\delta_i \geqslant \Delta_i/\varphi. \tag{5.2}$$

Let $R_i$ be the area of reception zone $\mathcal{H}_i$. As $\delta_i$ is the radius of a ball contained inside $\mathcal{H}_i$, we get a lowerbound on $R_i$ in terms of $d_i$:

$$R_i \geqslant \pi\delta_i^2 \geqslant \pi\left(\frac{\Delta_i}{\varphi}\right)^2 \geqslant \pi\left(\frac{d_i}{\varphi}\right)^2. \tag{5.3}$$

Let $a$ be the area of the region in which the nodes are distributed. Because $\beta > 1$, none of the reception zones can overlap. Then any link independent set $\mathcal{I}$ has

$$\sum_{i \in \mathcal{I}} R_i \leqslant a. \tag{5.4}$$

Therefore, a necessary condition for link independence is

$$\sum_{i \in \mathcal{I}} \pi \left( \frac{d_i}{\varphi} \right)^2 \leqslant a. \tag{5.5}$$

Consider a set of links $\mathcal{I}$ that, besides being link independent, is arbitrary. It might contain precisely the links with the smallest sender-receiver distances $d_i$. Let

$$d_{\min} = \min\{d_i \mid 1 \leqslant i \leqslant n\}. \tag{5.6}$$

**Lemma 5.10.** *If $\mathcal{I}$ is link independent, then*

$$|\mathcal{I}| \cdot \pi \left( \frac{d_{min}}{\varphi} \right)^2 \leqslant a.$$

*Proof.* By observing that of course $d_{\min} \leqslant d_i$ for all $i$, we see that

$$|\mathcal{I}| \cdot \pi \left( \frac{d_{\min}}{\varphi} \right)^2 \leqslant \sum_{i \in \mathcal{I}} \pi \left( \frac{d_i}{\varphi} \right)^2 \leqslant a$$

The second inequality holds since $\mathcal{I}$ is link independent: we can apply Equation 5.5. □

## 5.4.3. Expected value of lowerbound

We will now develop probabilistic statements on the size of link independent sets. The first step is to calculate the expected value $\mathbb{E}[\ |\mathcal{I}| \cdot \pi \left( \frac{d_{\min}}{\varphi} \right)^2\ ]$, which is the left-hand size of the bound in Lemma 5.10.

**Theorem 5.11.** *On expectation (taken over binomial instances), any subset $\mathcal{I} \subseteq S$ with $|\mathcal{I}| > \frac{\varphi^2 a}{\pi} \cdot \frac{n+1}{n}$ violates Lemma 5.10.*

*Proof.* In a random geometric instance, a receiver $r_i$ is placed at a uniform random point in a disc of radius 1 centered around sender $s_i$. This means that all distances from a sender to the corresponding receiver are independent, identically distributed in $[0, 1]$. The probability density function for the distances is $f(d) = 2d$ for $0 \leqslant d \leqslant 1$ and $f(d) = 0$ elsewhere.

By a standard order statistic calculation,[6] the probability density function for $d_{\min}$ is then (for $0 \leqslant d \leqslant 1$ and with $n = |S|$)

$$\begin{aligned} f_{\min}(d; n) &= n \cdot f(d) \cdot \left( \int_d^1 f(x)\, dx \right)^{n-1} && (5.7) \\ &= n \cdot 2d \cdot (1 - d^2)^{n-1}. && (5.8) \end{aligned}$$

We are now ready to calculate the expected value of our lowerbound of $\sum_{i \in \mathcal{I}} R_i$ (with $k = |\mathcal{I}|$).

$$\begin{aligned} \mathbb{E}\left[\, k \cdot \pi \left( \frac{d_{\min}}{\varphi} \right)^2 \,\right] &= \int_0^1 f_{\min}(x; n) \;\cdot\; k\pi \left( \frac{x}{\varphi} \right)^2 dx && (5.9) \\ &= \int_0^1 n 2x (1 - x^2)^{n-1} \;\cdot\; k\pi \left( \frac{x}{\varphi} \right)^2 dx && (5.10) \\ &= \frac{k\pi}{\varphi^2} \cdot \frac{n}{n+1} && (5.11) \end{aligned}$$

Plugging this into Lemma 5.10 and solving for k gives

$$\frac{k\pi}{\varphi^2} \cdot \frac{n}{n+1} \leqslant a \quad \Leftrightarrow \quad k \leqslant \frac{\varphi^2 a}{\pi} \cdot \frac{n+1}{n} \qquad (5.12)$$

Note that we have only used the link independence of $\mathcal{I}$ and nothing else, so the calculation goes through for any $\mathcal{I}$. This concludes the proof. □

### 5.4.4. Very large link independent sets are unlikely

We will now concern ourselves with the probability that an instance has a link independent set of a least a certain size. We will upperbound it by reasoning similar to the preceding sections and conclude that 'very' large independent sets are unlikely.

For small $|\mathcal{I}|$ this bound will be trivial (that is, there is no claim). This is reasonable since a small link independent set may well exist. For larger $|\mathcal{I}|$ an actual bound arises. This is precisely the claim we are making: *large* link independent sets are unlikely.

**Theorem 5.12.** *If* $k > \frac{a\varphi^2}{\pi}$, *then*

$$\mathbb{P}[\, S \textit{ has link independent set of size} \geqslant k \,] \;\leqslant\; 1 - \left( 1 - \frac{a\varphi^2}{k\pi} \right)^n.$$

*Proof.* We are looking at the existence of a link independent set of size k. We have a necessary condition in Lemma 5.10. Then the probability of satisfying

[6]See Section 2.2 on page 18 for the necessary preliminaries.

Lemma 5.10 is at least the probability of the existence of a link independent set: whenever a link independent set exists, the bound is satisfied.

$$\mathbb{P}[\text{ S has link independent set of size} \geqslant k\,] \;\leqslant\; \mathbb{P}\left[\, k \cdot \pi \cdot \left(\frac{d_{\min}}{\varphi}\right)^2 \;\leqslant\; a\,\right] \quad (5.13)$$

Rewriting gives

$$\mathbb{P}\left[\, k \cdot \pi \cdot \left(\frac{d_{\min}}{\varphi}\right)^2 \;\leqslant\; a\,\right] \;=\; \mathbb{P}\left[\, d_{\min} \;\leqslant\; \frac{\sqrt{a}\varphi}{\sqrt{k}\sqrt{\pi}}\,\right]. \quad (5.14)$$

To calculate this probability we first derive the cumulative distribution function of $d_{\min}$, for $0 \leqslant d \leqslant 1$.

$$F_{\min}(d;n) \;\overset{\text{def}}{=}\; \int_0^d f_{\min}(x;n)\,dx \;=\; 1-(1-d^2)^n.$$

Now we can plug in $d = \sqrt{a}\varphi/\sqrt{k}\sqrt{\pi}$. Note, however, that the above formula for $F_{\min}(d;n)$ only holds if $d \in [0,1]$. Given the lowerbound on $k$, we have that $d$ is indeed in this interval and we get

$$\mathbb{P}\left[\, d_{\min} \;\leqslant\; \frac{\sqrt{a}\varphi}{\sqrt{k}\sqrt{\pi}}\,\right] \;=\; F_{\min}\left(\frac{\sqrt{a}\varphi}{\sqrt{k}\sqrt{\pi}};n\right) \;=\; 1-\left(1-\frac{a\varphi^2}{k\pi}\right)^n.$$

The theorem follows. □

Note that if $k < \frac{a\varphi^2}{\pi}$, then $d > 1$, which gives $F_{\min}(d;n) = 1$ and we get the trivial bound of 1.

We will now make several observations by rearranging the bound of Theorem 5.12. First all, we get Theorem 5.5 from the introduction.

**Theorem 5.5.** *Consider a binomial point process instance with $n$ points on a torus of measure $a$, and let $k < n$. If*

$$k > \frac{a\varphi^2}{\pi(1-\sqrt[n]{1-p})},$$

*then the probability that the instance has a link independent set of cardinality $k$ is at most $p$. This lowerbound on $k$ is $\Omega\left(\frac{a \cdot n}{-\ln(1-p)}\right)$.*

*Proof.* Rearrange Theorem 5.12. □

Note that $k > \frac{a\varphi^2}{\pi}$ in Theorem 5.5; otherwise no such bound would arise. We can likewise get the bound that large link independent sets are unlikely in a small region.

**Corollary 5.13.** *Consider a binomial point process instance with* $n$ *points on a torus of measure* $a$*, and let* $k < n$*. If*

$$a < \frac{\pi k(1 - \sqrt[n]{1-p})}{\varphi^2},$$

*then the probability that the instance has a link independent set of cardinality* $k$ *is at most* $p$*. For constant* $\varphi$*, this upperbound on* $a$ *is* $\mathcal{O}(\frac{-\ln(1-p)\cdot k)}{n})$*.*

### 5.4.5. Large link independent set exists with high probability

In the preceding sections we have used the binomial model for analysing random instances: $n$ links in an area of measure $a$. For the remainder of this section we will switch to the Poisson model, with senders coming from a Poisson point process of density $n/a$ on a region of measure $a$. This will, in expectation, also give $n$ points. We make this switch because the independence properties of the Poisson model will aid our analysis.

In this section we will prove Theorem 5.6 from the introduction.

**Theorem 5.6.** *In Poisson instances of density* $n/a$ *on a square of measure* $a$*, or a torus with square area of measure* $a$*, with high probability a link independent set exists of size* $\Theta(\sqrt{\frac{n\cdot a}{\log a}})$*.*

*Proof.* We give a proof by algorithm: given the Poisson-model instance of density $\lambda = n/a$ we will find, with high probability, a solution of the claimed size.

First of all, we delete all links of length larger than $x$ (value to be determined later). This step accomplishes two things. Any surviving link has length at most $x$, which puts a lowerbound of $x^{-2}$ on the amount of signal. Furthermore, throwing away links reduces the amount of interference.

By our model of random instances, these link lengths are i.i.d. with probability density function $f(d) = 2d$. This means a link survives this step i.i.d. with probability $x^2$. If we start with $n$ links this leaves, in expectation, $n \cdot x^2$ links.

We will now look at the *SINR* conditions that a link independent set must satisfy and check it at every receiver. Recall that the *SINR* condition for link $i$ is as follows, where $g_{ij}$ is the strength of the signal from sender $i$ arriving at receiver $j$:

$$\frac{g_{ii}}{\sum_{j\neq i} g_{ji}} \geqslant \beta. \tag{5.15}$$

In the numerator is only the amount of signal. For any link that survived the first step, this is at least $x^{-2}$. In the denominator is the sum of the interferences, for which we will derive a concentrated expectation next.

For receiver $i$ the interference is the sum of the signal strengths from the other senders.

$$\text{interference at } i = \sum_{j\neq i} g_{ji} = \sum_{j\neq i} d_{ji}^{-2} \tag{5.16}$$

We group the terms of this summation geometrically into annuli surrounding $r_i$, where an annulus $\mathrm{ann}(p; r, R)$ is defined as

$$\mathrm{ann}(p; r, R) = B(p, R) - B(p, r). \tag{5.17}$$

We consider a square region $A$ that has measure $a$: regardless of $r_i$, the entire region is covered by a ball of radius $\sqrt{2a}$. That means the entire region is partitioned by the first $\lceil\sqrt{2a}\rceil$ integer annuli around $r_i$. Because of its location, any sender $j$ in $\mathrm{ann}(r_0, k, k+1)$ has $d_{ji} \leqslant k^{-2}$. Therefore, from Equation (5.16) we get

$$\text{interference at } i \leqslant \sum_{k=0}^{\lceil\sqrt{2a}\rceil} k^{-2} \cdot \#\text{ of senders in } \mathrm{ann}(r_i; k, k+1). \tag{5.18}$$

By the density of the Poisson point process we know the probability distribution of the number of senders in each annulus. Because of the independence property of the point process we can sum over each of these independently, where $\lambda'$ if the density of the point process. (Note that $\lambda' < \lambda$ because we have thrown away some links.) Plugging this into Equation (5.18) gives

$$\text{interference at } i \leqslant \sum_{k=1}^{\lceil\sqrt{2a}\rceil} k^{-2} \cdot \lambda' \cdot \|\mathrm{ann}(r_i; k, k+1)\|. \tag{5.19}$$

On the Euclidean plane the measure of a unit-width annulus is equal to

$$\|\mathrm{ann}(r_i; k, k+1)\| = \pi(k+1)^2 - \pi k^2 = \pi(2k+1). \tag{5.20}$$

For our case of a square region or a torus, not all of this measure may be relevant: part of the annulus may fall outside the square of interest and in case of the torus the set difference between balls has complicated behavior on the annuli. Still, the formula is a valid upperbound in either case. Equation (5.19) then becomes

$$\text{interference at } i \leqslant \sum_{k=1}^{\lceil\sqrt{2a}\rceil} k^{-2} \cdot \lambda' \cdot \pi(2k+1). \tag{5.21}$$

We started, by definition, with $\lambda = n/a$ and then we retained links i.i.d. with probability $x^2$. Then $\lambda' = x^2\lambda$, since we expect $nx^2$ points in an area of measure $a$.

$$\text{interference at } i \leqslant \sum_{k=1}^{\lceil\sqrt{2a}\rceil} k^{-2} \cdot \frac{nx^2}{a} \cdot \pi(2k+1) \tag{5.22}$$

We now move terms not involving $k$ out of the summation and recognise truncated harmonic and over-harmonic series. For some constant $c$ we have

$$\begin{aligned}
\text{interference at } i &\leqslant \sum_{k=1}^{\lceil\sqrt{2a}\rceil} k^{-2} \cdot \frac{nx^2}{a} \cdot \pi(2k+1) && (5.23)\\
&= \frac{2\pi n x^2}{a} \cdot \sum_{k=1}^{\lceil\sqrt{2a}\rceil} k^{-2} \cdot (k+1) && (5.24)\\
&= \frac{2\pi n x^2}{a} \cdot \sum_{k=1}^{\lceil\sqrt{2a}\rceil} \left(\frac{1}{k} + \frac{1}{k^2}\right) && (5.25)\\
&\leqslant \frac{2\pi n x^2}{a} \cdot \left(\ln(\lceil\sqrt{2a}\rceil + 1) + c\right). && (5.26)
\end{aligned}$$

Now we have a bound on the amount of interference. Recall that every remaining link has signal at least $x^{-2}$. Then the *SINR* condition for this node calls for

$$\frac{2\pi n x^2}{a} \cdot \left(\ln(\sqrt{2a}) + c\right) \leqslant x^{-2}.$$

We bound the probability of violating this inequality using a Chernoff bound; a union bound over all nodes then gives us the probability that all surviving nodes are satisfied. We pick $x$ such that this probability is $\mathcal{O}(n^{-1})$. The claimed bound follows. □

## 5.5. Concluding remarks

In this chapter we have considered random instances of LINK INDEPENDENT SET (PHYSICAL MODEL). Note, however, that Algorithm 1 from the previous chapter does not use the positions of the senders and receivers: it simply gets a gain matrix and works with that, without explicitly exploiting that (in this chapter) these gain matrices come from physical-model instances. We have shown experimentally that Algorithm 1 can be expected to perform well in practice. Depending on the density of the instance, this is due to the branching and reduction rules, the branch and bound, or both.

We have furthermore shown that the density $\lambda = n/a$ is a structurally meaningful parameter for the problem: both experiments and theory tell us that the size of the optimal link independent set is closely tied the density.

# Part II

# Graphical models

# 6 Robust recoverable path by backup nodes

In this chapter we consider possible ways of coping with node failures in wireless networks. We focus especially on the so-called single-node failure model, which captures the resilience of networks in a realistic fault setting. For an overview of the practical issues involved, see for example [NRR10, AYB13, WC06].

We introduce a model of recoverable routing under node failure. In this model we ask for a path that can be recovered easily and locally by assigning 'backup nodes.' These nodes play a role in assigning alternate routes when nodes fail (see for example [RDBL12]). We then resolve the basic algorithmic and complexity questions for this model: for some variants of the problem we provide polynomial-time algorithms, and for the others we prove $\mathcal{NP}$-completeness and provide exponential-time algorithms.

## 6.1. Introduction

We model the network we are considering by a simple graph $G = (V, E)$ with a start node $s \in V$ and a destination node $t \in V$. We write paths using square brackets, like $[p_1, p_2, \dots, p_k]$. We want to find a path $P$, not necessarily simple, from $s$ to $t$. We complicate the problem by considering the possibility of node failure, for which we want to provide a certain level of robustness. Our robustness condition—that the path must remain valid in the presence of a single node failure—would, by itself, make us overly cautious. We therefore include a recovery model which allows for the easy recovery of a valid path in case a failure makes our initial solution invalid. In contrast to earlier studies of this fault model, we focus on the crucial complexity issues of the model.

### 6.1.1. Failure model

We consider node failure: if a node $v$ fails, the solution path $P$ is no longer allowed go through $v$. That is, in case node $v$ fails, $P$ must not include $v$. We will concern ourselves with single-node failures: any one node $v \notin \{s, t\}$ can fail.

Asking for a path $P$ that is valid in any failure scenario is then uninteresting. If $(s, t) \in E$, then $P = [s, t]$ is a valid solution. Otherwise there is no solution at all: if the edge $(s, t)$ does not exist, the path $P$ must contain an internal node and that node may fail, making $P$ an invalid solution. To arrive at an interesting problem, we give a more realistic recovery model.

### 6.1.2. Recovery model

In case a failure invalidates our initial path $P$, we don't want a totally different solution. We want there to be a simple, local way to repair to the path $P$. In this way the failure can be dealt with in an online fashion: just travel along $P$, and if the path turns out to be blocked by a node failure, take a local detour and then resume further along on $P$ as originally planned. We allow the following kind of local fix.

A *path-with-backups* $R$ assigns to each *main node* $p_i$ a single *backup node* $b_i$. The intuition is that if node $p_i$ fails, node $b_i$ will be able to take its place. We say "$p_i$ is backed up by $b_i$" and "$b_i$ backs up $p_i$."

We write a path with backups as $[\frac{p_1}{b_1}, \frac{p_2}{b_2}, \dots, \frac{p_k}{b_k}]$. By $\mathrm{path}(R)$ we denote the path formed by the main nodes of $R$, that is,

$$\mathrm{path}\left( [\frac{p_1}{b_1}, \frac{p_2}{b_2}, \dots, \frac{p_k}{b_k}] \right) = [p_1, p_2, \dots, p_k].$$

**Definition 6.1 (Recoverable path).** A path with backups $R = [\frac{p_1}{b_1}, \frac{p_2}{b_2}, \dots, \frac{p_k}{b_k}]$ is called a *recoverable path* if and only if the following two properties are satisfied.

- $\mathrm{path}(R)$ is a path in $G$ (not necessarily from $s$ to $t$).
- The following recovery procedure succeeds for any node $v \notin \{s, t\}$: take $\mathrm{path}(R)$, but wherever $p_i = v$, use $b_i$ instead. The resulting path $P$ must be a path in $G - v$.

The nodes $p_i$ are called the *main nodes of* $R$ and the nodes $b_i$ are called the *backup nodes of* $R$. Note that the recovered path $P$ has the same length as $R$.

**Definition 6.2 (Simple recoverable path).** Simple recoverable path A recoverable path $R$ is called *simple* if and only if $\mathrm{path}(R)$ is simple.

Note that if a failing node $v$ occurs in $\mathrm{path}(R)$ more than once, then it may happen that the recovery procedure substitutes a different backup node for each occurrence: at this point, we do not require that $p_i = p_j$ implies $b_i = b_j$.

This definition of a recoverable path technically allows $p_i = b_i$, but this only works for the nodes $s$ and $t$ since those are the only nodes that cannot fail.

**Lemma 6.3 (No self-backup).** *If $p_i \notin \{s, t\}$, then in a recoverable path $p_i \neq b_i$.*

*Proof.* Let $p_i \notin \{s, t\}$. If $p_i$ fails, the recovery procedure substitutes $b_i$. If $p_i = b_i$, then the recovered path is invalid. Therefore necessarily $p_i \neq b_i$ in a recoverable path. □

In a recoverable path, even the first and last nodes have a backup node. When considering a routing problem, it makes more sense to have a single source and a single destination.

**Definition 6.4 (Recoverable $s$-$t$-path).** A recoverable path $R = [\frac{p_1}{b_1}, \frac{p_2}{b_2}, \dots, \frac{p_k}{b_k}]$ is a recoverable $s$-$t$-path if and only if $p_1 = b_1 = s$ and $p_k = b_k = t$. We write it as $[s, \frac{p_2}{b_2}, \dots, \frac{p_{k-1}}{b_{k-1}}, t]$.

In this chapter we consider the following combinatorial problem.

| | **Recoverable path** |
|---|---|
| *Instance:* | A simple graph $G = (V, E)$. |
| | Nodes $s, t \in V$. |
| *Question:* | Does there exist a recoverable $s$-$t$-path in $G$? |

We will give a polynomial-time algorithm for finding recoverable paths for the basic cases, as well as algorithms and/or hardness proofs for several variants of the problem; the results are summarised in Section 6.1.4. First we make some basic observations. See Figure 6.1 for an illustration of the edge sets involved in the following two statements.

**Lemma 6.5.** $R = [\frac{p_1}{b_1}, \frac{p_2}{b_2}, \dots, \frac{p_k}{b_k}]$ *is a recoverable $s$-$t$-path if and only if the following conditions all hold:*

1. $p_1 = b_1 = s$,
2. $p_k = b_k = t$,
3. $p_i \notin \{s, t\} \implies p_i \neq b_i$,
4. *the following edges are in $E$, for all $1 < i < k$: $(p_{i-1}, p_i)$, $(p_i, p_{i+1})$, $(p_{i-1}, b_i)$ and $(b_i, p_{i+1})$.*

*Proof.* The first two conditions come from the definition of recoverable $s$-$t$-path. The third condition comes from Lemma 6.3.

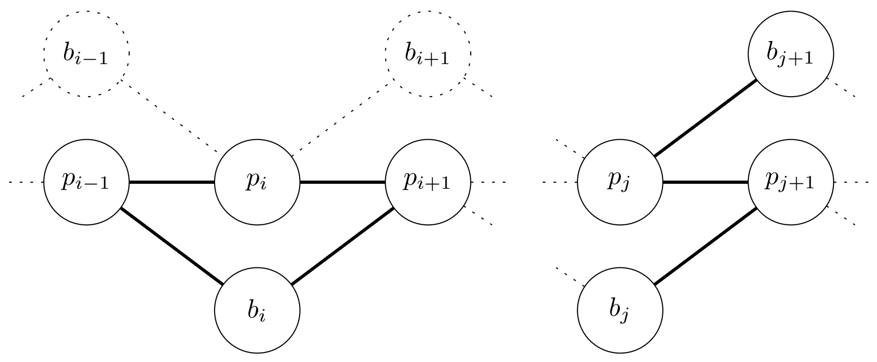


**Figure 6.1.** In a recoverable path at least the following edges must exist among the nodes $p_{i-1}$, $p_i$, $p_{i+1}$ and $b_i$: $(p_{i-1}, p_i)$, $(p_i, p_{i+1})$, $(p_{i-1}, b_i)$ and $(b_i, p_{i+1})$.
Equivalently, the following edges must exist among the nodes $p_j$, $b_j$, $p_{j+1}$ and $b_{j+1}$: $(p_j, p_{j+1})$, $(p_j, b_{j+1})$ and $(b_j, p_{j+1})$.

Consider the edges mentioned in condition four. The first two edges correspond to $\mathrm{path}(R)$ being a path in G. The final two edges correspond to a valid recovery path in case $p_i$ fails. The graph G is simple, so for any $i$ we have $p_i \neq p_{i+1}$. As only a single node can fail, no edge is required from $b_i$ to $b_{i+1}$: at most one of these backups is used. If these edges are present in G, then R is a recoverable $s$-$t$-path. If R is a recoverable $s$-$t$-path, then these edges are present in G. □

**Proposition 6.6.** *The edge set in condition 4 of the preceding lemma is equivalently defined by:*

- *For all* $1 \leqslant i < k$*:* $(p_i, p_{i+1})$, $(p_i, b_{i+1})$ *and* $(b_i, p_{i+1})$.

Note that a recoverable path has $b_{i-1} \neq p_i$, since equality would imply a self-loop on $p_i$. That is, it would imply $(p_i, p_i) \in E$, which is not the case in a simple graph.

### 6.1.3. Recovery variants

In this chapter we will look at four variations of the Recoverable path problem by considering two questions.

- Is a node allowed to back up multiple main nodes, or does $b_i = b_j$ imply $p_i = p_j$? That is, is the relation from main nodes to their backup injective?
- Is a node allowed to be backed up by multiple nodes if it occurs as a main node multiple times, or does $p_i = p_j$ imply $b_i = b_j$? That is, is the relation from main nodes to their backup a function?

**Definition 6.7 (Backup relation: functional, injective).** We consider four versions of the recoverable path problem, depending on the kind of main-node-to-backup relation they allow. Let $B$ be the relation consisting of all pairs $(p_i, b_i)$ occurring on a recoverable $s$-$t$-path. The variations are named as follows.

- $B$ is allowed to be many-to-many: *many-to-many backup*.
- $B$ must be injective but not necessarily a function: *injective backup*.
- $B$ must be a function but not necessarily injective: *functional backup*.
- $B$ must be one-to-one: *one-to-one backup*.

Note that one-to-one backup is both functional and injective. The following lemma shows that a recoverable path has functional backup if and only if it is simple.

**Lemma 6.8.** *The following conditions are equivalent.*

- *(Simple:)* $G$ *has a simple recoverable* $s$-$t$-*path* $R$.
- *(Functional-backup:)* $G$ *has a recoverable* $s$-$t$-*path* $R$ *such that* $p_i = p_j$ *implies* $b_i = b_j$.

*Proof.* We prove the implication both ways.

(Simple $\implies$ Functional-backup.) Every node occurs at most once in $\mathtt{path}(R)$. Then $p_i \neq p_j$ for all $i, j$, and we are done.

(Functional-backup $\implies$ Simple.) If $R$ has functional backup but is not simple, it can be made simple by shortcuts. Let $i < j$ and $p_i = p_j$. Then $b_i = b_j$ because the backup relation is functional. Remove steps $p_{i+1}$ through $p_j$ from $R$. The result is still a recoverable path. Repeat until $R$ is simple. □

It may seem reasonable to restrict our attention only to simple recoverable paths. However, consider Figure 6.2, which has a recoverable $s$-$t$-path but does not have a *simple* recoverable $s$-$t$-path.

**Proposition 6.9.** *Consider the recoverable* $s$-$t$-*path problem with many-to-many backup. There exists a graph* $G$ *which has no simple recoverable* $s$-$t$-*path, but does have a non-simple recoverable* $s$-$t$-*path.*

### 6.1.4. Results

Here we summarise our results, the first being a polynomial-time algorithm for Recoverable path with many-to-many backup, including some weighted versions (Section 6.2). Then we present an $\mathcal{NP}$-completeness proof for the other three variants (Section 6.3). We give exponential-time algorithms for these hard variants (Section 6.4).

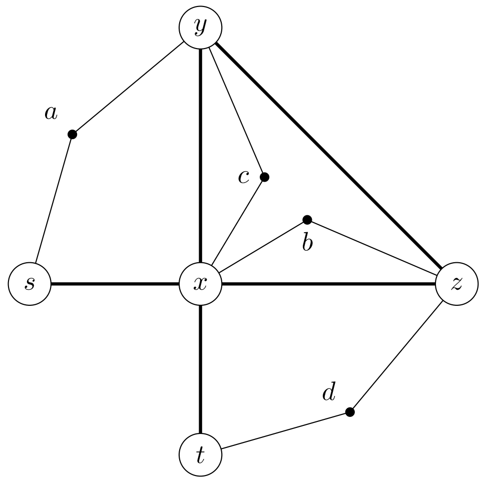


**Figure 6.2.** This graph has the following non-simple recoverable $s$-$t$-path: $[s, \frac{x}{a}, \frac{y}{b}, \frac{z}{c}, \frac{x}{d}, t]$. Note that its backup relation is not a function.
With functional backup, this graph has no recoverable $s$-$t$-path. Equivalently, this graph has no simple recoverable $s$-$t$-path.

The following table organises the complexity of the Recoverable path variants based on the two axes of whether the backup relation is required to be injective or not, and whether it is required to be functional or not.

| | | Injective | |
|---|---|---|---|
| | | Optional | Required |
| Functional | Optional | Polynomial time | $\mathcal{NP}$-complete |
| | Required | $\mathcal{NP}$-complete | $\mathcal{NP}$-complete |

In Section 6.5 we look at a related problem: a normal path is given and we ask whether backups can be assigned to make the path recoverable. We show $\mathcal{NP}$-completeness for the injective case and give an exponential time algorithm. For the other three cases we provide polynomial-time algorithms. This situation is shown in the following table.

| | | Injective | |
|---|---|---|---|
| | | Optional | Required |
| Functional | Optional | Greedy | $\mathcal{NP}$-complete |
| | Required | Greedy | Bipartite matching |

## 6.2. Polynomial-time algorithm for many-to-many backup

Here we present a polynomial-time algorithm for Recoverable path with many-to-many backups. We also solve some weighted variations. All of these problems are solved in $\mathcal{O}(n^4)$ time on general graphs and $\mathcal{O}(n^3)$ time on sparse graphs.

### 6.2.1. Reachability

To begin with, we define an auxiliary graph $G_2 = (V_2, E_2)$. Let $V_2$ be the set of ordered pairs of distinct nodes in $V$, with two additional nodes $(s, s)$ and $(t, t)$. We interpret these pairs as consisting of a main node and a corresponding backup node. Then we add an edge between two such pairs if they can follow each other in a recoverable s-t-path in $G$, which we can decide because of Proposition 6.6. Expanding these definitions gives the following.

**Definition 6.10 (Auxiliary graph $G_2$).** Let $G_2 = (V_2, E_2)$, where

$$V_2 = \{ (p, b) \in V^2 \mid p \neq b \vee p = s \vee p = t \}$$

and

$$E_2 = \{ ((p_1, b_1), (p_2, b_2)) \in {V_2}^2 \mid (p_1, p_2) \in E \wedge (b_1, p_2) \in E \wedge (p_1, b_2) \in E \}.$$

The graph $G_2$ has, by construction, $\Theta(n^2)$ nodes and therefore $\mathcal{O}(n^4)$ edges.

**Lemma 6.11.** *There exists a recoverable* s-t-*path* $R$ *in* $G$ *if and only if there exists a normal* $(s, s)$-$(t, t)$-*path* $P$ *in* $G_2$.

*Proof.* Interpret P's nodes as (main node, backup node) pairs in G and consider whether this is a recoverable s-t-path. Equivalence follows by checking the conditions of Lemma 6.5.

1. Starting from $(s, s) \in V_2$ guarantees $p_1 = b_1 = s$.
2. Going to $(t, t) \in V_2$ guarantees $p_k = b_k = t$.
3. By construction, $(v, v) \in V_2$ if and only if $v \in \{s, t\}$. This guarantees that $p_i \notin \{s, t\} \implies p_i \neq b_i$.
4. By construction, an edge in $E_2$ exists if and only if the edges required by Proposition 6.6 exist in E. □

**Theorem 6.12.** Recoverable path *with many-to-many backup can be solved in* $\mathcal{O}(n^4)$ *time.*

*Proof.* Construct $G_2$ in $\mathcal{O}(n^4)$ time. Check for an $(s, s)$-$(t, t)$-path in $\mathcal{O}(n^4)$ time, for example using depth first search. Correctness follows from Lemma 6.11. □

In case the original graph G is sparse, a better bound arises.

**Lemma 6.13.** *The size of* $E_2$ *is bounded by* $\mathcal{O}(|E|^3)$.

*Proof.* Note in Definition 6.10 that the existence of an edge in $E_2$ corresponds uniquely to the existence of a certain set of three edges in $E$. The number of distinct triples in $E$ bounds the size of $E_2$ by $\mathcal{O}(\, |E|^3 \,)$. □

**Corollary 6.14.** Recoverable path *with many-to-many backup can be solved in* $\mathcal{O}(\, n^3 \,)$ *time on sparse graphs.*

### 6.2.2. Shortest-path versions

We can extend the approach from the previous algorithm to handle several weighted versions of the problem. In a sensor-network context, this can be used to model, for example, delay times or energy costs. Consider a weight function $w : E \to \mathbb{Z}_{\geqslant 0}$.

| | **Recoverable shortest path** |
|---|---|
| *Instance:* | A simple graph $G = (V, E)$. |
| | Nodes $s, t \in V$, |
| | Weight function $w : E \to \mathbb{Z}_{\geqslant 0}$. |
| | Weight limit $W$. |
| *Question:* | Does there exist a recoverable $s$-$t$-path $R$ in $G$ such that the weight of $\mathrm{path}(R)$, according to $w$, is at most $W$? |

**Theorem 6.15.** Recoverable shortest path *with many-to-many backup can be solved in* $\mathcal{O}(\, n^4 \,)$ *time.*

*Proof.* Again we use the auxiliary graph $G_2$ and now introduce a weight function $w_{\text{main}} : E_2 \to \mathbb{Z}_{\geqslant 0}$. The weight of an auxiliary edge is set equal to the weight of the edge between the corresponding main nodes. This way, for any recoverable path $R$, the path that represents $R$ in $G_2$ has the same weight as $\mathrm{path}(R)$ in $G$. That is, we set

$$w_{\text{main}}(\; ((p_1, b_1), (p_2, b_2)) \;) = w(\; (p_1, p_2) \;).$$

We can use a standard technique to find the minimum-weight $(s, s)$-$(t, t)$-path. Since $G_2$ has $\mathcal{O}(\, n^2 \,)$ nodes and can have $\Omega(\, n^4 \,)$ edges, any reasonable implementation of Dijkstra's algorithm gives a runtime of $\mathcal{O}(\, n^4 \,)$; for details, see for example [CLRS09]. □

More advanced algorithms to find a single-pair shortest path exist, but do not improve the worst-case runtime of our algorithm, since the number of edges in $G_2$ can be $\Omega(\, n^4 \,)$.

The preceding version of the problem only considers the weight of the path in case nothing goes wrong. If there is in fact a node failure that impacts the path, we are faced with the recovery procedure, which in general will give a path of different weight. To take this into account, we will now look at the expected length of the recoverable path, including potential recovery. We will work with a probability distribution over which node will fail, if any. To be precise, the sample space is any one node in $V - \{s, t\}$ or 'no failure.' We denote the latter by $\varnothing$.

| | **Expected-shortest recoverable path** |
|---|---|
| *Instance:* | A simple graph $G = (V, E)$. |
| | Nodes $s, t \in V$. |
| | Weight function $w : E \to \mathbb{Z}_{\geqslant 0}$. |
| | Weight limit $W$. |
| | Probability mass function $f : \{\varnothing\} \cup V - \{s, t\} \to \mathbb{Q}$. |
| *Question:* | Considering node failure according to $f$, does there exist a recoverable $s$-$t$-path in $G$ such that the expected weight of the path, according to $w$, is at most $W$? |

**Theorem 6.16.** Expected-shortest recoverable path *with many-to-many backup can be solved in* $\mathcal{O}(n^4)$ *time.*

*Proof.* Again we use the auxiliary graph $G_2$ and introduce a weight function $w_{\text{expect}} : E_2 \to \mathbb{Q}_{\geqslant 0}$. An auxiliary edge in $E_2$ corresponds to three edges in $E$. We can determine the expected weight these three edges contribute when included in a recoverable path. Then by linearity of expectation the shortest path in $G_2$ is the recoverable path in $G$ with the lowest expected length.

Consider an auxiliary edge $((p_1, b_1), (p_2, b_2)) \in E_2$. By construction we have that $p_1 \neq p_2$. Then if $p_1$ fails, $p_2$ does not fail. This means that the edge $(b_1, p_2)$ is used precisely with probability $f(p_1)$. Symmetrically, the edge $(p_1, b_2)$ is used with probability $f(p_2)$. The edge $(p_1, p_2) \in E$ is used if and only if $p_1$ and $p_2$ both don't fail, which happens with probability $1 - f(p_1) - f(p_2)$. Therefore, we set

$$\begin{aligned} w_{\text{expect}}(\,((p_1, b_1), (p_2, b_2))\,) &= w(b_1, p_2) \cdot f(p_1) \\ &\quad + w(p_1, b_2) \cdot f(p_2) \\ &\quad + w(p_1, p_2) \cdot (1 - f(p_1) - f(p_2)). \end{aligned}$$

Runtime is again $\mathcal{O}(n^4)$, by using Dijkstra's algorithm [CLRS09]. □

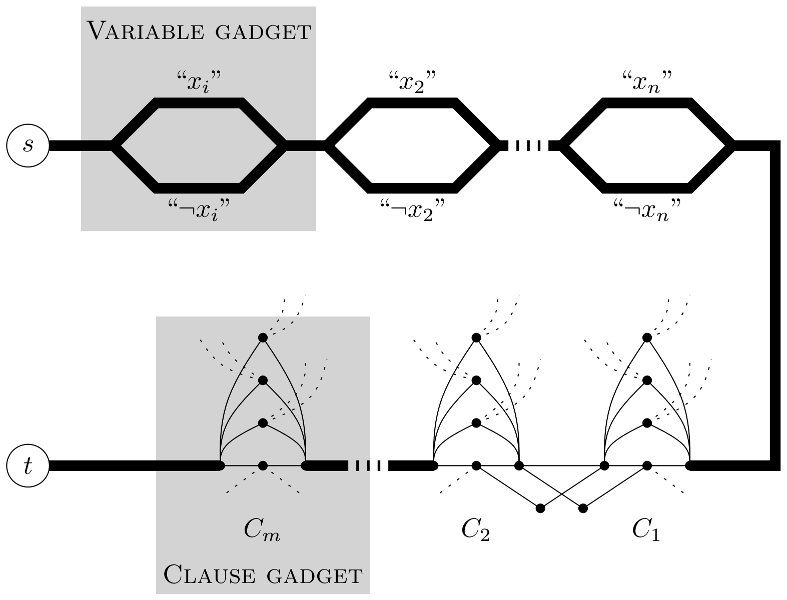


**Figure 6.3.** Overview of the entire instance for injective backup. Fat lines indicate *tracks*. Any recoverable $s$-$t$-path must first go through the $n$ assignment gadgets, then through the $m$ clause gadgets.

## 6.3. $\mathcal{NP}$-completeness of all other cases

We will now prove $\mathcal{NP}$-completeness of the injective, functional, and one-to-one variants of the Recoverable path problem. In all cases, membership of $\mathcal{NP}$ is trivial. We prove $\mathcal{NP}$-hardness by a polynomial-time reduction from 3-cnf-sat. For an instance of 3-cnf-sat with $n$ variables in $m$ clauses, we construct a recoverable-$s$-$t$-path instance that looks roughly as follows:

- In series, for every variable, a gadget with two possible recoverable paths through it. The choice of one or the other forces certain nodes to be used as backups and this represents a truth assignment to the variables of the 3-cnf-sat instance.
- After that, in series, for every clause, a gadget that can only be passed through if the clause is satisfied in the chosen truth assignment.

The overview of this structure can be seen in Figure 6.3. In the analysis of the instance, we will rely heavily on the following property. It is valid in all problem variants.

**Definition 6.17 (Doomed path).** Let $x, y, z \in V - \{s, t\}$ be distinct nodes and let $[x, y, z]$ be a path in $G$. If there does not exist a node $b \in V - \{y\}$ such that the edges $(x, b)$ and $(b, z)$ exist, then $[x, y, z]$ is called *doomed*.

Note that $b \notin \{x, z\}$ since otherwise a self loop on $b$ required (which does not exist because $G$ is simple).

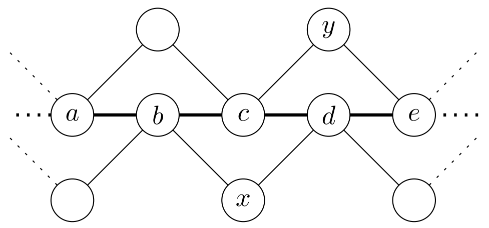


**Figure 6.4.** A stretch of track. Nodes $a$, $b$, $c$, $d$ and $e$ are called centre nodes. The other nodes are called side nodes. Note that, for example, $[a, b, x]$ and $[b, c, y]$ are doomed: the recoverable path must remain 'on the track.'

**Lemma 6.18.** *If a path $[x, y, z]$ is doomed, it does not occur as a subpath of $\mathrm{path}(R)$ for any recoverable $s$-$t$-path $R$.*

*Proof.* In a recoverable path, node $y$ needs a backup. By Proposition 6.3, this cannot be $y$ itself. Call the backup node $b$. Then by Lemma 6.5 the following edges must exist: $(x, b)$ and $(b, z)$. But $[x, y, z]$ is doomed, so by definition no such node exists. □

Now we define the details of the reduction, which are partly the same for both variants. First of all, we observe that we can make a part of the graph that a recoverable path can go through in exactly one way. To do this, we make 'centre' nodes and 'side' nodes with exactly the edges required by Lemma 6.5 to be used as main nodes and backup nodes respectively. We call this a *track* and it is illustrated in Figure 6.4. On any recoverable path coming over the track, we know that the centre will be used as main nodes and that the sides will be used as backup nodes. The basis of our instance is a track from a start node $s$ to a destination node $t$.

We can fork and join such tracks as illustrated in Figure 6.5. The gadget for a variable $x$ is made by forking the main track into an "$x$" and a "$\neg x$" track. These are kept going long enough (to be determined later, depending on number of appearances of the $x$) and then joined again. Any recoverable $s$-$t$-path must take one side or the other and thereby use one set of backup nodes or the other. We interpret taking the "$x$" track as assigning $x$ to true and taking the "$\neg x$" track as assigning $x$ to false.

The clause gadget is a different for the the injective case and for the functional case. We will cover these separately.

**Theorem 6.19.** *Injective variants of the recoverable $s$-$t$-path problem are $\mathcal{NP}$-complete.*

*Proof.* Construct the reduction from 3-CNF-SAT as above. In the injective case, the clause gadget consists of the 10 nodes illustrated in Figure 6.6, of which 3 are shared with variable gadgets. First of all there are the nodes $p_1 \cdots p_5$. These will be main nodes on any recoverable $s$-$t$-path. The nodes $b_2$ and $b_4$ will be backup for $p_2$ and $p_4$ respectively. Finally there are edges to 3 side nodes on specific variable-gadget tracks: call them $c_1$, $c_2$, $c_3$. We call these the *link nodes*

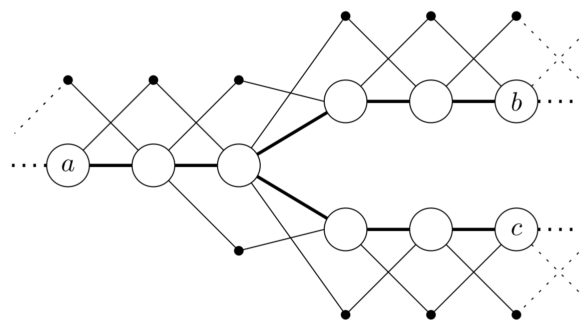


**Figure 6.5.** A track can be forked or joined like this. A recoverable path can go from $a$ to $b$ or from $a$ to $c$.

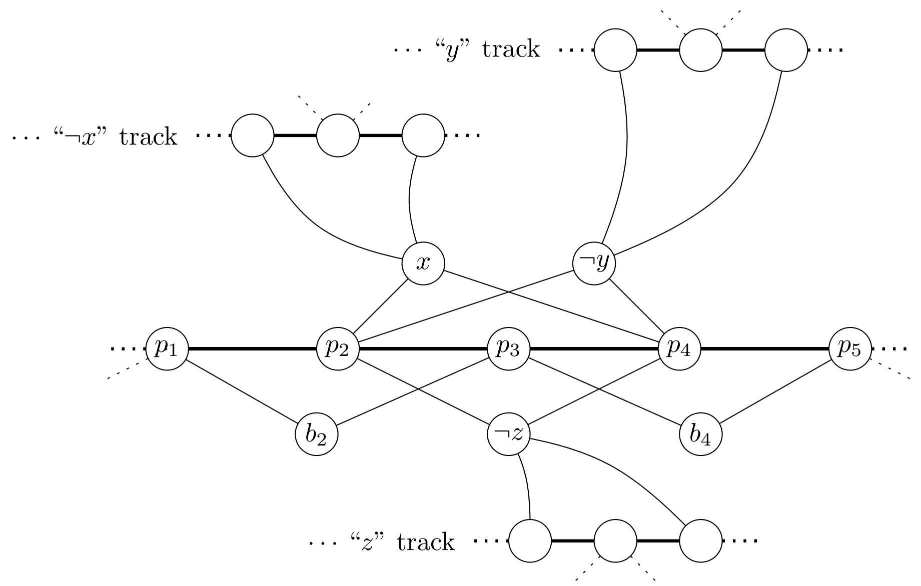


**Figure 6.6.** A clause gadget corresponding to $(x \vee \neg y \vee \neg z)$. To get from $p_1$ to $p_5$, the main nodes of a robust path need to be $[p_1, p_2, p_3, p_4, p_5]$. Consider in particular the node labeled $p_3$. The backup for $p_3$ must be one the nodes labeled $x$, $\neg y$ or $\neg z$. Note that these are shared with the "$\neg x$" track, the "$y$" track and the "$z$" track respectively.

because they will link clauses to variables. Any of these nodes can potentially be backup for $p_3$. Indeed, one of them *must* be backup for $p_3$: by construction there are no other options. Call this node $c_i$. Then in the variable gadget where $c_i$ is a side node, a recoverable $s$-$t$-path cannot pass through the track with $c_i$: with injective backup, a node can back up only a single node.

In particular, for every clause of three literals, we make a clause gadget with the following link nodes: for every literal in the clause, a side node on the track that corresponds to the negation of the literal. We make the variable-gadget tracks long enough so that every link node is shared between only one variable gadget and one clause gadget. Then there is a recoverable path through a clause

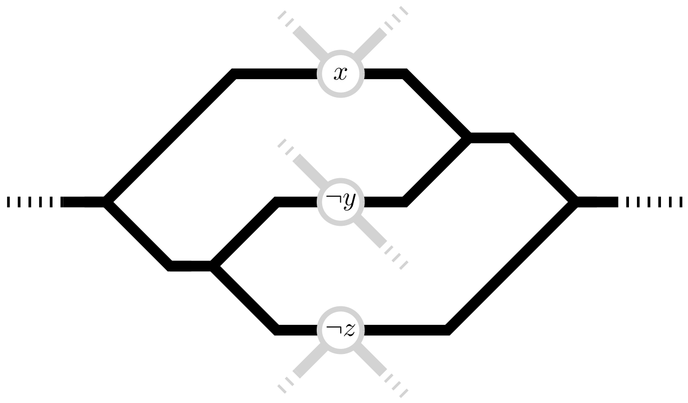


**Figure 6.7.** A clause gadget corresponding to $(x \vee \neg y \vee \neg z)$. To get from one side to the other, the path must go through one of the nodes labeled $x$, $\neg y$ or $\neg z$. There are shared with the "$\neg x$" track, the "$y$" track and the "$z$" track respectively.

gadget if and only if the truth assignment chosen in the variable gadgets satisfies the clause: a particular link node is available as backup for $p_3$ if and only if the corresponding literal is satisfied in the chosen truth assignment.

For correctness, we observe two further properties. Firstly, $p_3$ is main node on any recoverable $s$-$t$-path: for any link node $\ell$, $[p_1, p_2, \ell]$ is doomed. Secondly, a recoverable path cannot use the link nodes to 'escape' from a variable gadget into a clause gadget: any path consisting of a variable-gadget node, a link node and a clause-gadget node is doomed. □

**Theorem 6.20.** *Functional variants of the recoverable $s$-$t$-path problem are $\mathcal{NP}$-complete.*

*Proof.* Construct the reduction from 3-CNF-SAT as before. Here, the clause gadget is larger and consists mostly of track, as illustrated in Figure 6.7. This time, the three indicated nodes function as *link nodes*. They are identified with a centre node on a variable-gadget track as indicated in Figure 6.8. Note that, in Figure 6.8, when taking the variable-gadget track, node $p$ must have backup $a$. When taking the clause-gadget track, the node $p$ must have backup $b$. Then if the backup is functional, $p$ can be used in only one of them. □

## 6.4. Exponential-time algorithms

In this section we provide algorithms that work, with minor modification, for each of the three hard variants of RECOVERABLE PATH: functional, injective and one-to-one. The runtime is $\mathcal{O}^*(2^n)$ for the functional and injective cases and is $\mathcal{O}^*(4^n)$ for the one-to-one case.

First we present a dynamic programming algorithm. In the functional and injective cases, it runs in $\mathcal{O}(2^n \cdot n^3)$ time and $\mathcal{O}(2^n \cdot n^2)$ space. In the one-to-one case, it runs in $\mathcal{O}(4^n \cdot n^3)$ time and $\mathcal{O}(4^n \cdot n^2)$ space.

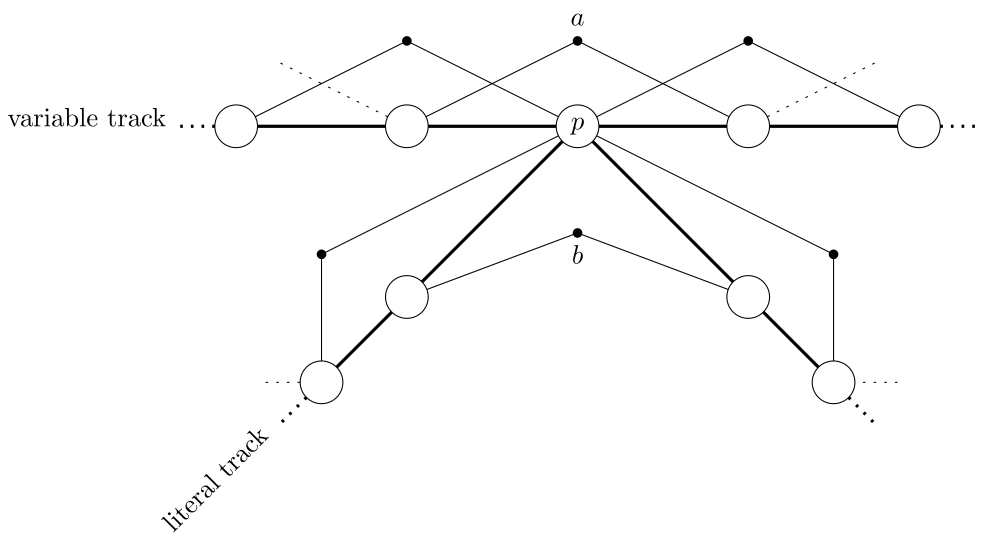


**Figure 6.8.** The connection between a "variable track" and a "clause track." Consider in particular the node labeled $p$. If the variable track is used, $p$ must be backed up by node $a$. If the clause track is used, $p$ must be backed up by node $b$.

Then we present an algorithm based on graph search that runs in $\mathcal{O}(\, c^n \cdot |E_2|\,)$ time and space, with $E_2$ as in Section 6.2.1. Again, $c = 2$ for the functional and injective cases and $c = 4$ for the one-to-one case. Note that $|E_2|$ can be $\Theta(\, n^4\,)$ in the worst case, but also might be $o(\, n^3\,)$.

## 6.4.1. Dynamic programming

We will now develop a dynamic programming algorithm for Recoverable path. For notational convenience we define the concept of a *friend set*.

**Definition 6.21 (Friend set $F(x, y, z)$).** Let $x$, $y$, $z$ be nodes in $G$. The *friend set* of $(x, y, z)$ is the set of nodes $v$ that can backup node $y$ on a recoverable path where $y$ occurs between $x$ and $z$. That is

$$F(x, y, z) = \{\, v \in V \mid (x, v) \in E \wedge (v, z) \in E \ \wedge v \neq y \,\}.$$

Consider functional backup. This means that every occurrence of a node $p$ as main node on a recoverable path must be backed up by the same node. By Lemma 6.8 we know that, in fact, there exists a recoverable $s$-$t$-path with functional backup if and only if there exists a simple recoverable $s$-$t$-path, so once we have used a node as main node we can disregard ever using it again.

We will solve this problem defining a boolean function $p_{\text{fun}}$ in the parameters $y, z \in V$ and $S \subseteq V$, and developing a recurrence relation for it. For arguments $(S, y, z)$ it has the following value: does there exist a simple recoverable path with $p_1 = b_1 = s$, ending with main node $y$ followed by $z$, and using, besides $y$ and $z$, exactly the nodes in $S$ as main nodes.

As a base case for our recurrence relation, we can observe that $p_{\text{fun}}(\varnothing, y, z)$ is true precisely if $y = s$ and $(y, z) \in E$: the path must start at $s$ and the edge must exist. For non-empty $S$, we have that $p_{\text{fun}}(S, y, z)$ is true if and only if the edge $(y, z)$ actually exists, and there exists a predecessor $x$ for $y$ such that

- a valid backup exists for $y$, and
- by recursion, there exists a recoverable path ending in $x$ and $y$ that further uses precisely the nodes in $S - \{x\}$ as main nodes.

This gives the following equations.

$$\begin{aligned} p_{\text{fun}}(\varnothing, y, z) &= \begin{cases} \mathit{True} & \text{If } y = s \wedge (y, z) \in E \\ \mathit{False} & \text{Otherwise} \end{cases} \\ p_{\text{fun}}(S, y, z) &= (y, z) \in E \\ &\quad \wedge\ \exists x \in S : \Big( \exists b \in F(x, y, z)\ :\ p_{\text{fun}}(S - \{x\}, x, y) \Big) \end{aligned} \tag{6.1}$$

Note that checking for the existence of $b \in F(x, y, z)$ corresponds to checking for the edges required in condition 4 of Lemma 6.5.

**Theorem 6.22.** Recoverable path *with functional backup can be solved in* $\mathcal{O}(2^n \cdot n^3)$ *time and* $\mathcal{O}(2^n \cdot n^2)$ *space.*

*Proof.* Check, using dynamic programming, whether $p_{\text{fun}}(S, y, t)$ is true for any $S \subseteq V$ and $y \in V$. Because of the recurrence relation of $p_{\text{fun}}$, this is equivalent to the existence of a recoverable $s$-$t$-path: exactly the edges required by Lemma 6.5 are present. The parameter $S$ ensures that the path is simple, which ensures functional backup by Lemma 6.8.

As for runtime, we start out by noticing that the dynamic program has $\mathcal{O}(2^n \cdot n^2)$ states. These can be calculated in $\mathcal{O}(n)$ time each as follows. Checking $(y, z) \in E$ is of course simple. Then there are existential quantifiers over $x \in S$ and $b \in F(x, y, z)$, both of which might range over $\Theta(n)$ items. Note however that $b$ is not used in the recurrence. We can therefore precompute $(\exists b \in F(x, y, z))$ for all combinations of nodes $x$, $y$, and $z$ in $V$. Then this check runs in constant time by table lookup. □

**Theorem 6.23.** Recoverable path *with injective backup can be solved in* $\mathcal{O}(2^n \cdot n^3)$ *time and* $\mathcal{O}(2^n \cdot n^2)$ *space.*

*Proof.* The approach is similar to the functional case. First we solve the problem in $\mathcal{O}(2^n \cdot n^4)$ time and then we improve the polynomial term in the runtime to $n^3$.

We consider injective backup. This means that every node can be backup for at most one other node. If ever a node occurs as a backup node more than once, then each occurrence backs up the same main node. So in fact, the same main-node-backup-node pair can occur more than once. But then a shorter recoverable $s$-$t$-path exists by shortcutting. This means that we only have to check for a recoverable $s$-$t$-path in which every node occurs as a backup at most once.

We now define a boolean function $p_{\text{in}}(S, y, z)$ in the parameters $y, z \in V$ and $S \subseteq V$ with the following meaning: does there exist a recoverable path with $p_1 = b_1 = s$, ending with main nodes $y$ and $z$ and in which exactly the nodes in $S$ are used as backup nodes, each exactly once. A recurrence relation similar to Equation (6.1) holds.

$$\begin{aligned} p_{\text{in}}(\emptyset, y, z) &= \begin{cases} \mathit{True} & \text{If } y = s \wedge (y,z) \in E \\ \mathit{False} & \text{Otherwise} \end{cases} \\ p_{\text{in}}(S, y, z) &= (y,z) \in E \\ &\quad \wedge\ \exists x \in V : \Big( \exists b \in F(x,y,z) \cap S \ :\ p_{\text{in}}(S - \{b\}, x, y) \Big) \end{aligned} \tag{6.2}$$

Notice the following differences to $p_{\text{fun}}$. Node $x$ is now quantified over all of $V$, no longer $S$: main nodes are no longer restricted. Furthermore, node $b$ is now quantified over $F(x, y, z) \cap S$: in this step of the recurrence we will "use $b$ as a backup" and this must be reflected in $S$. Because of these differences, the recursion is now on the set $S - \{b\}$ instead of $S - \{x\}$.

The base case is unchanged: $p_{\text{in}}(\emptyset, y, z)$ is true if and only if $y = s$ and $(y, z) \in E$. Then a recoverable $s$-$t$-path with injective backup exists if and only if there exists a set $S$ and a node $y$ such that $p_{\text{in}}(S, y, t)$ is true.

The quantification over $b$ can now no longer be precomputed, as it depends on $S$: $b \in F(x, y, z) \cap S$ and also $b$ occurs in the recursion depending on $x$. Then naively computing a dynamic programming state takes $\Theta(n^2)$ time. We improve this to $\mathcal{O}(n)$ as follows.

First we expand the definition of $F(x, y, z)$ in (6.2) to arrive at (6.3), leaving the base case unchanged.

$$\begin{aligned} p_{\text{in}}(S, y, z) \ &= \ (y,z) \in E \\ &\quad \wedge\ \exists x \in V : \Big( \exists b \in S \ :\ (x,b) \in E \\ &\qquad\qquad\qquad\qquad \wedge\ (b,z) \in E \\ &\qquad\qquad\qquad\qquad \wedge\ b \neq y \\ &\qquad\qquad\qquad\qquad \wedge\ p_{\text{in}}(S - \{b\}, x, y) \Big) \end{aligned} \tag{6.3}$$

Then we flip the order of the quantifiers and rearrange to arrive at (6.4).

$$\begin{aligned} p_{\text{in}}(S,y,z) \;=\; & (y,z) \in E \\ \wedge\;\; & \exists b \in S : \Big( \; (b,z) \in E \;\wedge\; b \neq y \\ & \qquad \wedge\, \exists x \in V : (\, (x,b) \in E \wedge p_{\text{in}}(S - \{b\}, x, y)\, ) \Big) \end{aligned} \tag{6.4}$$

Notice that in the subexpression quantified by "$\exists x$" the only free variables are $S$, $y$ and $b$. Now we introduce the function $q_{\text{in}}(S, y, b)$. This allows for a definition of $p_{\text{in}}$ that is mutually recursive with $q_{\text{in}}$ as follows.

$$\begin{aligned} p_{\text{in}}(S,y,z) \;=\; & (y,z) \in E \\ & \wedge\; \exists b \in S : \Big( \, (b,z) \in E \;\wedge\; b \neq y \;\wedge\; q_{\text{in}}(S - \{b\}, y, b) \, \Big) \end{aligned} \tag{6.5}$$

$$q_{\text{in}}(S,y,b) \;=\; \exists x \in V : \Big( \, (x,b) \in E \;\wedge\; p_{\text{in}}(S,x,y) \, \Big) \tag{6.6}$$

The domain of both $p_{\text{in}}$ and $q_{\text{in}}$ has size $\mathcal{O}(\, 2^n \cdot n^2 \,)$: the arguments are a subset of the vertices, and two specific vertices. The value of any $q_{\text{in}}(S, y, b)$ can be calculated in $\mathcal{O}(\, n\, )$ time when the value of $p_{\text{in}}$ can be looked up in constant time for all sets $S'$ of cardinality equal to $S$. The value of any $p_{\text{in}}(S, y, z)$ can be calculated in $\mathcal{O}(\, n\, )$ time when the values of $q_{\text{in}}$ can be looked up in constant time for all sets $S'$ one node smaller than $S$. So all values of both $p_{\text{in}}$ and $q_{\text{in}}$ can be calculated in a total of $\mathcal{O}(\, 2^n \cdot n^3 \,)$ time by computing, level-wise in increasing size of $S$, first $p_{\text{in}}$ and then $q_{\text{in}}$. □

One may note that this mutual recursion between $p_{\text{in}}$ and $q_{\text{in}}$ in the proof has a reasonable interpretation. From parameters $y$ and $z$ we can find a valid backup $b$. From parameters $y$ and $b$ we can find a valid main-path predecessor $x$.

Lastly we consider the one-to-one case. It follows by a simple adaptation of the machinery we have seen. There is, however, a slight complication. This time we need to know two things for every node: is it already used as main node and, independently, is it already used as a backup node? This is because we allow a node to be both main node and (elsewhere) backup node on the same path; this is simply something the definitions permit. In the machinery that we provide, this unfortunately leads to a runtime of $\Theta^*(\, 4^n \,)$. (If we were to disallow nodes being both main node and backup node on the same path, an $\mathcal{O}^*(\, 2^n \,)$-time algorithm like the previous ones would be possible.)

**Theorem 6.24.** Recoverable path *with one-to-one backup can be solved in* $\mathcal{O}(\, 4^n \cdot n^3 \,)$ *time and* $\mathcal{O}(\, 4^n \cdot n^2 \,)$ *space.*

*Proof.* Again we define a function and evaluate it using a recurrence relation. We now have a set $M \subseteq V$ of main nodes and a set $B \subseteq V$ of backup nodes.

Define $p_{bi}$ by

$$p_{bi}(\varnothing,\varnothing,y,z) = \begin{cases} \mathit{True} & \text{If } y = s \wedge (y,z) \in E \\ \mathit{False} & \text{Otherwise} \end{cases}$$

$$\begin{aligned} p_{bi}(M,B,y,z) &= (y,z) \in E \\ &\wedge\ \exists x \in M : \Big(\exists b \in F(x,y,z) \cap B :\ p_{bi}(M-\{x\}, B-\{b\}, x, y)\Big) \end{aligned} \tag{6.7}$$

Like with the preceding algorithm, we split the recurrence relation in two parts.

$$\begin{aligned} p_{bi}(M,B,y,z) &= (y,z) \in E \\ &\wedge\ \exists b \in B : \Big(\ (b,z) \in E \wedge b \neq y \wedge q_{bi}(M, B-\{b\}, y, b)\ \Big) \end{aligned} \tag{6.8}$$

$$q_{bi}(M,B,y,b) = \exists x \in M : \Big(\ (x,b) \in E \wedge p_{bi}(M-\{x\}, B, x, y)\ \Big) \tag{6.9}$$

Using dynamic programming the values of $p_{bi}$ en $q_{bi}$ can be calculated in linear time each, for a total of $\mathcal{O}(\,4^n \cdot n^3\,)$ time (proceeding level-wise in increasing size of $M$ and $B$). □

### 6.4.2. Graph search

Here we present a set of different algorithms for the same problems we considered above. The difference in runtime is polynomial and depends on the input; whereas the runtime of the dynamic programming algorithms is wholly independent of the input graph, the following algorithms have a tighter connection to the graphs involved.

**Many-to-many backup.** We will once more use the auxiliary graph $G_2$ (Definition 6.10) as a starting point. In the many-to-many variant of the problem, $G_2$ captures the entirety of the problem: a recoverable $s$-$t$-path exists in $G$ if and only if an ordinary $(s,s)$-$(t,t)$-path exists in $G_2$. We will now construct an exponential-sized graph that fulfills the same purpose for the other variants.

**Functional backup.** If a path in $G_2$ visits a node $(p,b)$, it should not visit nodes $(p,b')$ for $b' \neq b$: if it did, the corresponding recoverable path would not have functional backup. Observe that by Lemma 6.8 we only have to look for a recoverable path that is simple, that is, in which each node occurs at most once as a main node. So in fact, once we have passed $(p,b)$, we can disregard any path that goes through another node $(p,b')$ for any $b'$.

To model this, we make a directed graph $G_{\mathrm{fun}}$ and for its node set tentatively start with $2^n$ copies of $V_2$: one corresponding to each subset $S \subseteq V$ of nodes of $G$. Write such nodes as $(p,b)_S$. The sets $S$ represent which nodes have already been 'used up' and should not be visited again as a main node. We therefore remove all the nodes $(p,b)_S$ where $p \in S$. This gives

$$V_{\mathrm{fun}} = \{ (p,b)_S \mid (p,b) \in V_2 \wedge S \in 2^V \wedge p \notin S \}.$$

For the arcs, we start from $E_2$ but take care of $S$: an edge $(\,(p_1,b_1),\ (p_2,b_2)\,) \in E_2$ turns into arcs $(\,(p_1,b_1)_S,\ (p_2,b_2)_{S\cup\{p_1\}}\,)$ in $G_{\mathrm{fun}}$ for all appropriate sets $S$. This enforces that once the node $p_1$ is used as main node on the recoverable path, it is never again used as main node: we go to a part of $G_{\mathrm{fun}}$ where $p_1$ never occurs as a main node. This gives

$$\begin{aligned} E_{\mathrm{fun}} = \{\, (\,(p_1,b_1)_S,\ (p_2,b_2)_{S\cup\{p_1\}}\,) \mid\ & ((p_1,b_1),(p_2,b_2)) \in E_2 \\ & \wedge S \in 2^V \\ & \wedge (p_1,b_1)_S \in V_{\mathrm{fun}} \\ & \wedge (p_2,b_2)_{S\cup\{p_1\}} \in V_{\mathrm{fun}} \,\}. \end{aligned}$$

**Lemma 6.25.** *The size of* $E_{\mathrm{fun}}$ *is* $\mathcal{O}(\,2^n \cdot |E_2|\,)$.

*Proof.* Observe the definition of $E_{\mathrm{fun}}$: every arc in $E_{\mathrm{fun}}$ corresponds to a combination of an edge in $E_2$ and an $S \in 2^V$. □

**Theorem 6.26.** Recoverable path *with functional backup can be solved in* $\mathcal{O}(\,2^n \cdot |E_2|\,)$ *time and space.*

*Proof.* By construction, $G_{\mathrm{fun}}$ has a path from $(s,s)_\varnothing$ to $(t,t)_S$ for some $S \subseteq V$ if and only if there is a recoverable $s$-$t$-path in $G$. First we construct $E_2$ in polynomial time. This then allows us to construct $G_{\mathrm{fun}}$ in $\mathcal{O}(\,2^n \cdot |E_2|\,)$ time. We run a depth-first search from $(s,s)_\varnothing$ and stop when we encounter a node $(t,t)_S$ for any $S \subseteq V$. By Lemma 6.25, the number of arcs and therefore the runtime of the depth-first search is $\mathcal{O}(\,2^n \cdot |E_2|\,)$. The space bound follows identically. □

Note that it is not necessary to construct $G_{\mathrm{fun}}$ explicitly before starting the graph search: the neighbourhood of any node can be determined locally during the search.

**Injective backup.** The algorithm for injective backup differs only by searching a slightly different graph.

**Theorem 6.27.** Recoverable path *with injective backup can be solved in* $\mathcal{O}(\,2^n \cdot |E_2|\,)$ *time.*

*Proof.* In the case of injective backup, if a path in $G_2$ visits a node $(p,b)$ it should not visit nodes $(p',b)$ for $p' \neq p$: if it did, the corresponding recoverable path would not have injective backup. Observe that if there exists a path that uses a node $(p, b)$ more than once, there also exists a path that uses $(p, b)$ at most once: use shortcuts. So in fact, once we have passed $(p, b)$, we can disregard any path that goes through any other node $(p', b)$ for any $p'$.

Similarly to the functional-backup case, we define an $\mathcal{O}^*(\, 2^n\,)$-sized directed graph $G_{in}$ as follows. Let

$$V_{in} = \{\, (p, b)_S \mid (p, b) \in V_2 \;\wedge\; S \in 2^V \;\wedge\; b \notin S \,\}$$

and

$$\begin{aligned} E_{in} = \{\, (\, (p_1, b_1)_S,\ (p_2, b_2)_{S \cup \{b_1\}}\,) \mid\ & ((p_1, b_1), (p_2, b_2)) \in E_2 \\ & \wedge S \in 2^V \\ & \wedge (p_1, b_1)_S \in V_{in} \\ & \wedge (p_2, b_2)_{S \cup \{b_1\}} \in V_{in} \,\}. \end{aligned}$$

Search for a path in $G_{in}$ from $(s, s)_\varnothing$ to $(t, t)_S$ for any $S \subseteq V$. Time and space bounds follow like before. □

**One-to-one backup.** Like with the dynamic programming approach, this graph search approach seems to require $\Theta^*(\, 4^n\,)$ time for handling one-to-one backup relations.

**Theorem 6.28.** Recoverable path *with one-to-one backup can be solved in* $\mathcal{O}(\, 4^n \cdot |E_2|\,)$ *time.*

*Proof.* Once more we define an exponential-size graph. In this graph $G_{bi}$ the nodes are subscripted by not one but two subsets: nodes already used as main node and nodes already used as backup. The construction is similar to the previous cases:

$$\begin{aligned} V_{bi} = \{\, (p, b)_{(M,B)} \mid (p, b) \in V_2 \;\wedge\; & M \in 2^V \;\wedge\; p \notin M \\ & \wedge B \in 2^V \;\wedge\; b \notin B \,\} \end{aligned}$$

and

$$\begin{aligned} E_{bi} = \{\, & (\, (p_1, b_1)_{(M,B)},\ (p_2, b_2)_{(M \cup \{p_1\}, B \cup \{b_1\})}\,) \\ & \mid ((p_1, b_1), (p_2, b_2)) \in E_2 \\ & \quad \wedge\ M \in 2^V \;\wedge\; B \in 2^V \\ & \quad \wedge\ (p_1, b_1)_{(M,B)} \in V_{bi} \\ & \quad \wedge\ (p_2, b_2)_{(M \cup \{p_1\}, B \cup \{b_1\})} \in V_{bi} \\ & \}. \end{aligned}$$

The size of $G_{bi}$ is bounded similarly to Lemma 6.25 and again a graph search is made. This gives an algorithm running in the claimed time and space. □

We have now seen two different approaches to solving the $\mathcal{NP}$-complete cases of Recoverable path. Both take $\mathcal{O}^*(2^n)$ time and space on the injective case and on the functional case. On the one-to-one case, the cost in time and space is is $\mathcal{O}^*(4^n)$. By running both approaches in parallel, the polynomial factor in the runtime is $\mathcal{O}(\min\{n^3, |E_2|\})$.

## 6.5. Backup assignment

In this section we take a look at a problem related to finding a recoverable path. This time we are given an $s$-$t$-path $P$ and the question is: does there exist a *recoverable* $s$-$t$-path $R$ such that $\mathrm{path}(R) = P$? We call this the *backup assignment* problem.

| | **Backup assignment** |
|---|---|
| *Instance:* | A simple graph $G = (V, E)$ |
| | Nodes $s, t \in V$ |
| | An $s$-$t$-path $P$ in $G$. |
| *Question:* | Does there exist a recoverable $s$-$t$-path $R$ such that $\mathrm{path}(R) = P$? |

We can again look at four variations based on what kind of backup relation we allow. We show that the injective variant of the problem is $\mathcal{NP}$-complete and give an exponential time algorithm. We give polynomial-time algorithms for the other three variants.

For the analysis of the problem, we use *friend sets* again (compare Definition 6.21). This time it is convenient to index them differently.

**Definition 6.29 (Friend set $F(P, i)$).** Let $P = [p_1, \dots, p_k]$ be a path in $G$ and let $p_{i-1}$, $p_i$, $p_{i+1}$ be consecutive nodes on $P$. The *friend set* of index $i$ is the set of nodes $v$ that can backup $p_i$. That is

$$\begin{aligned} F(P, i) &= F(p_{i-1}, p_i, p_{i+1}) \\ &= \{\, v \in V \mid (p_{i-1}, v) \in E \wedge (v, p_{i+1}) \in E \;\wedge v \neq p_i \,\}. \end{aligned}$$

### 6.5.1. Polynomial cases

We now give polynomial-time algorithms for the three problem variants that are, presumably, not $\mathcal{NP}$-complete. The algorithm for the many-to-many variant is the simplest.

**Theorem 6.30.** Backup assignment *with many-to-many backup can be solved in polynomial time.*

**Algorithm 4:** AssignBackup-Many-to-Many( $G$, $P$ )

```
1 foreach $p_i \in P$ do
2     if $F(P, i) \neq \varnothing$ then
3         Assign to $p_i$ an arbitrary node from $F(P, i)$.
4     else
5         No valid backup assignment exists.
6     end
7 end
```

**Algorithm 5:** AssignBackup-Functional( $G$, $P$ )

```
1 Let $\mathcal{I}(v)$ be the set of indices where node $v$ occurs on $P$.
2 foreach distinct $v \in P$ do
3     Let $B = \bigcap_{i \in \mathcal{I}(v)} F(P, i)$
4     if $B \neq \varnothing$ then
5         Assign to $v$ an arbitrary node from $B$.
6     else
7         No valid backup assignment exists.
8     end
9 end
```

*Proof.* According to Lemma 6.5, the edges required for a node to be in a friend set are exactly those that are required to be a legal backup. Because the backup relation is allowed to be many-to-many, every node can be considered separately. Therefore, in a solution to Backup assignment with many-to-many backup, any node can be backed up by any node from its friend set and only by those.

We can then use a trivial greedy algorithm: assign to every node an arbitrary node from its friend set. The procedure is shown in Algorithm 4. If the algorithm fails—because some $F(P, i)$ is empty—no valid backup assignment exists. The algorithm is clearly polynomial. □

The functional variant is not much more complicated.

**Theorem 6.31.** Backup assignment *with functional backup can be solved in polynomial time.*

*Proof.* Compared to the many-to-many case, we have the extra condition that every time a node $p$ occurs on $P$ it must be assigned the same backup. Therefore we can assign to it a certain backup node $b$ only if it is valid for every occurrence of $p$. This leaves only the nodes that are in the intersection of friend sets of all occurrences of $p$. Among those, the choice can again be made arbitrarily. The procedure is shown in Algorithm 5. The algorithm is clearly polynomial. □

We will solve the one-to-one variant of the problem using bipartite matching.

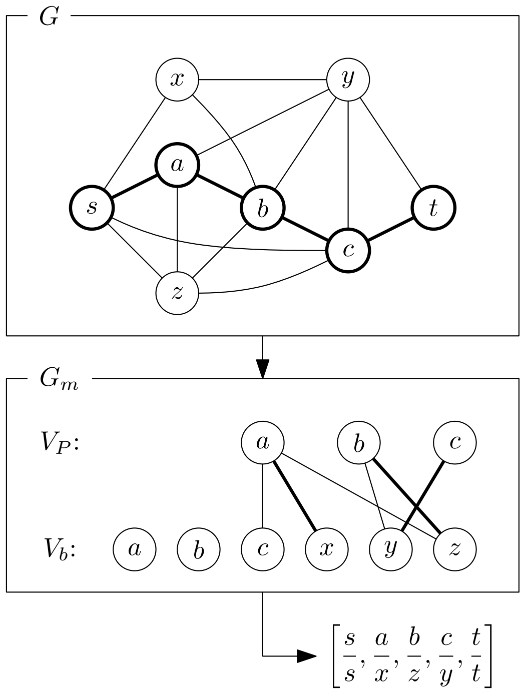

$G$
$x$
$y$
$a$
$s$
$b$
$t$
$c$
$z$
$G_m$
$V_P$:
$V_b$:
$\left[\frac{s}{s}, \frac{a}{x}, \frac{b}{z}, \frac{c}{y}, \frac{t}{t}\right]$

**Figure 6.9.** Example of the matching graph $G_m$ for one-to-one BACKUP ASSIGNMENT.

As the name suggests, we will use the matching to assign main nodes to backup nodes. We will now construct a bipartite graph $G_m$ that models the right constraints.

Note that a node may occur on a recoverable path both as a main node and as a backup node. We therefore construct two sets of nodes, which together form the node set of $G_m$.

- A set of nodes $V_P$ representing the main nodes of the path, with a node for every distinct node occurring on $P$, except $s$ and $t$.
- A set of nodes $V_b$ representing potential backup nodes, with a node for every node in $V - \{s, t\}$.

We exclude $s$ and $t$ because in a recoverable $s$-$t$-path these are necessarily assigned to backup themselves: by definition $p_1 = b_1 = s$ and $p_k = b_k = t$. Then with one-to-one backup the nodes $s$ and $t$ are fully occupied and can be disregarded.

To obtain the edge set of $G_m$, we make an edge between a node $p \in V_P$ and a node $b \in V_b$ if and only if $b$ is a legal backup for $p$. That is, if and only if $b$ is in the friend set of all occurrences of $p$. An example of this construction can be seen in Figure 6.9.

**Theorem 6.32.** BACKUP ASSIGNMENT *with one-to-one backup can be solved in polynomial time.*

**Algorithm 6:** AssignBackup-One-to-One( G, P )

```
1  Let $V_P$ be a set with a node for every distinct node on $P$ except $s$ and $t$.
2  Let $V_b = V - \{s, t\}$
3  Let $\mathcal{I}(v)$ be the set of indices where node $v$ occurs on $P$.
4  Let $E_m = \{ (p, b) \in V_P \times V_b \mid b \in \bigcap_{i \in \mathcal{I}(p)} F(P, i) \}$.
5  Let $M$ be a maximum-cardinality matching in $G_m = (V_p \cup V_b, E_m)$.
6  if $|M| = |V_P|$ then
7      Assign backup according to $M$.
8  else
9      No valid backup assignment exists.
10 end
```

*Proof.* By construction, the graph $G_m[V_P]$ is empty and so is $G_m[V_b]$. Then a matching in $G_m$ has size at most $|V_P|$ and any edge in any matching involves exactly one node from $V_P$ and one from $V_b$. We will interpret an edge in the matching as assigning a main node to a backup node.

By construction of the edge set, there exists a one-to-one backup assignment if and only if there exists a matching of size $|V_P|$ in $G_m$. So we construct $G_m$ and find a maximum-cardinality matching (see Algorithm 6). If it has size $|V_P|$ then we have a valid backup assignment. If the maximum matching is smaller, no valid backup assignment exists. The graph $G_m$ can clearly be constructed in polynomial time and the matching can also be found in polynomial time (see for example [CLRS09]). □

### 6.5.2. Hardness

Now we turn to the remaining case of injective backup, which, as we mentioned already at the beginning of this section, is $\mathcal{NP}$-complete. The proof uses a polynomial-time reduction from 3-cnf-sat.

An instance of Backup assignment consists of both a graph G and a prescribed path P. Note that in an $\mathcal{NP}$-completeness proof this path P will, in general, be non-simple: an injective backup assignment for a simple path is necessarily one-to-one and can, by our preceding results, be found in polynomial time.

**Theorem 6.33.** Backup assignment *with injective backup is $\mathcal{NP}$-complete.*

*Proof.* Membership is $\mathcal{NP}$ is trivial. To prove $\mathcal{NP}$-completeness, we will now construct an equivalent Backup assignment instance $\mathcal{I}$ for a given 3-cnf-sat instance with $n$ variables and $m$ clauses. We construct a gadget for every variable, a gadget for every clause, and a 'central node' $c$. These are connected as shown in Figure 6.10. Note that the central node $c$ is visited $m$ times by the path $P$.

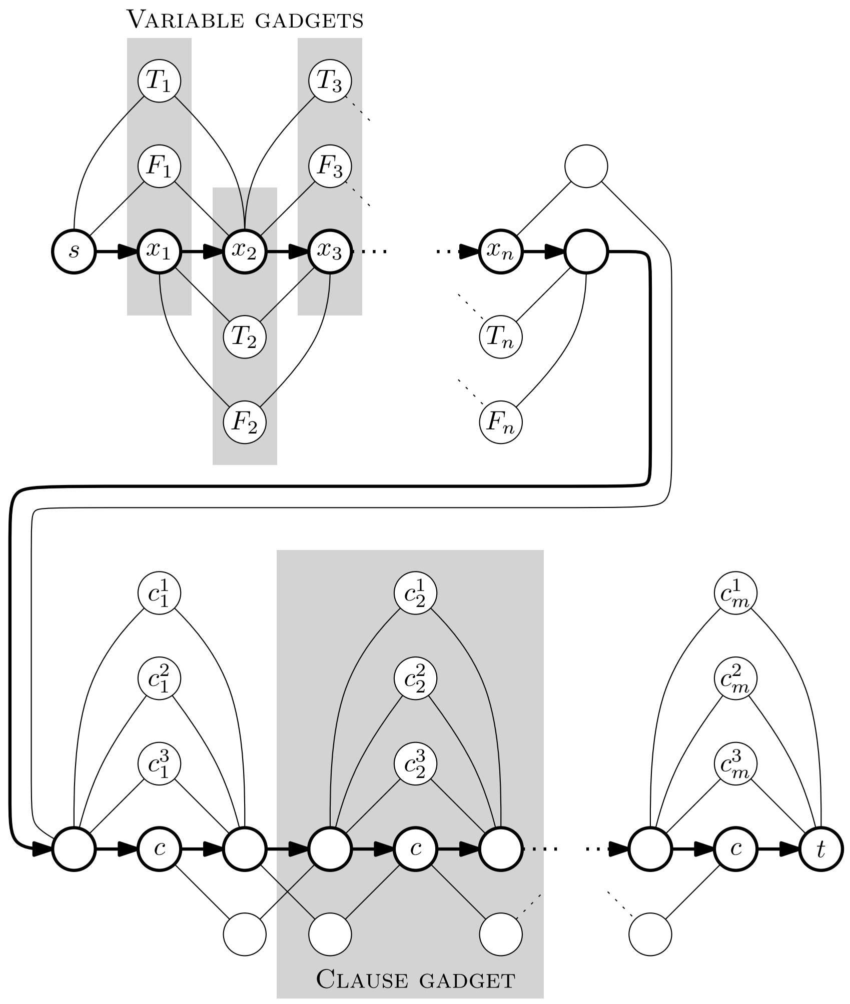


**Figure 6.10.** Overview of the instance of the $\mathcal{NP}$-completeness reduction for injective BACKUP ASSIGNMENT. The path P is indicated in fat lines. All nodes labeled c are in fact the same node, drawn in separate places for clarity. Likewise, depending on the 3-CNF-SAT instance, nodes $c_i^j$ are identified with nodes $T_k$ or $F_k$.

- The gadget for variable $i$ consists of nodes $x_i$, $T_i$ and $F_i$. The instance is such that the friend set of $x_i$ consists solely of $T_i$ and $F_i$. Choosing either one as backup for $x_i$ encodes a truth assignment for variable $i$ of the SAT instance.
- The gadget for clause $i$ consists mainly of three nodes $c_i^1$, $c_i^2$ and $c_i^3$: one for each literal in clause $i$ of the SAT instance. The prescribed path $P$ visits a dummy node, then $c$, then another dummy node. These are connected as in Figure 6.10. As a result, the friend set of this occurrence of $c$ is exactly $\{c_i^1, c_i^2, c_i^3\}$.

Since the backup relation is not required to be functional, the central node $c$

can be assigned a different backup for each clause gadget. However, since the backup relation is required to be injective, if a node $c_i^j$ is used to backup the central node c for the $i^{th}$ gadget, it cannot be used to also backup another node. In particular, it cannot be used as backup for any gadget for a variable. We use this to model a consistent truth assignment of the 3-CNF-SAT instance as follows.

Consider literal $j$ of clause $i$ in the 3-CNF-SAT instance, $j \in \{1, 2, 3\}$. Let $x_k$ be the variable involved. If the variable occurs unnegated, we identify node $c_i^j$ with node $F_k$. Since backup is injective, this means that the node $c_i^j = F_k$ cannot be backup for both variable-gadget node $x_k$ and central node $c$. In particular, if $F_k$ is used as backup in the variable gadget, it cannot be used in any clause gadgets. Likewise, if variable $x_k$ occurs negated, we identify $c_i^j$ with $T_k$. Then if the 3-CNF-SAT instance is satisfiable, there exists an injective recoverable path by this correspondence.

Conversely, suppose the constructed instance has an injective recoverable path. Just like every clause in the 3-CNF-SAT instance needs at least one satisfied literal, every clause gadget needs at least one of $\{c_i^1, c_i^2, c_i^3\}$ to be available to backup $c$. It is essential that all clause gadgets use a single central node $c$. Recall that injective backup requires $\forall i, j : b_i = b_j \implies p_i = p_j$. By using $c$ as main node each time allows a single node $T_k$ or $F_k$ to be used for multiple clause gadgets. In this way the backups of the variable gadgets correspond to a satisfying truth assignment. Then there exists a satisfying assignment for the SAT instance if and only if $G$ admits an injective backup assignment for $P$. □

### 6.5.3. Exponential-time algorithm

We finish this section by presenting an exponential-time algorithm for BACKUP ASSIGNMENT with injective backup. It is based on dynamic programming. Note that since the backup relation is not required to be functional, we really need to assign backups for occurrences of nodes on $P$, not just one backup for every distinct node on $P$.

We start off with an observation about the structure of injective backup assignments and some notation.

**Lemma 6.34.** *Consider an injective backup assignment and let $p_i = p_j$ be distinct occurrences of a single node. Suppose $b_i \neq b_j$ and $b_i \in F(P, j)$. Then changing $p_j$'s backup from $b_j$ to $b_i$ results in another valid injective backup assignment.*

*Proof.* There is no problem with being a backup assignment: $b_i \in F(P, j)$. There is also no problem with injectivity, since $p_i = p_j$. □

This shows that there is some freedom in injective backup assignments: if multiple nodes are used to back up the various occurrences of a single node $v$, these can be freely changed within the limitation of the above lemma. This will be used to argue correctness of some arbitrary choices the algorithm will have to make when picking backup nodes.

**Definition 6.35 (Index set).** The index set $\mathcal{I}(v)$ of a node $v$ is the set of indices where the node $v$ occurs on the path $P$, that is,

$$\mathcal{I}(v) \;=\; \{i \in \mathbb{N} \mid p_i = v\}.$$

**Definition 6.36 (Node multiplicity).** The multiplicity $\mu(v)$ of a node $v$ is the number of times $v$ occurs on the path $P$, that is,

$$\mu(v) \;=\; |\mathcal{I}(v)|.$$

Let $\mu_{max} = \max\{\ \mu(v) \mid v \in V\ \}$.

**Definition 6.37 ($\prec$, $v_{min}$, $v_{max}$, $\mathtt{pred}(v)$).** Fix an arbitrary total order $\prec$ on $V$. Let $v_{min}$ be the minimum node according to $\prec$, and $v_{max}$ be the maximum. By $\mathtt{pred}(v)$ we denote the predecessor of $v$ according to $\prec$.

We will now set up a function for use in the dynamic programming algorithm. We handle the nodes of the graph one by one, in some order; for each node $v$, we consider the occurrences of $v$ in the order that they occur on $P$.

**Definition 6.38.** The boolean function $a(v, O, B)$ is defined for arguments $v \in V$, $O \subseteq \mathcal{I}(v)$, $B \subseteq V$. It is defined to be true if and only if there is a way to assign backups that is injective, where exactly the nodes in $B$ are used as backup, and where exactly the following occurrences are assigned a backup: all $p_i \prec v$, and all $p_j$ for $j \in O$. (Thus leaving all other occurrences unassigned.)

Then we can solve Backup assignment with injective backup as follows. Check whether $a(\ v_{max},\ \mathcal{I}(v_{max}),\ B\ )$ is true for any subset $B$: by definition this means assigning backups to all occurrences on $P$, using any set of backup nodes.

**Theorem 6.39.** Backup assignment *with injective backup can be solved in*

$$\mathcal{O}^*(\ 2^{n+\mu_{max}}\ )$$

*time.*

*Proof.* Calculate $a(v, O, B)$ for all combinations of $v \in V$, $O \in \mathcal{I}(v)$ and $B \subseteq V$, using the following recurrence relation.

If some occurrence of $v$ is already backed up (that is, $O \neq \varnothing$), we can recurse on which node $b \in B$ is its backup. In view of Lemma 6.34, we can then immediately use $b$ to backup as many other occurrences of $v$ as possible: since we are already deciding to use $b$ as backup for *some* occurrence of $v$, it cannot be wrong to use it for more occurrences. There is still the choice of which occurrence in $O$ to recurse on, but this choice can be made arbitrarily. Call this node $\mathtt{arb}(O)$. Then

$$a(\ v,\ O,\ B\ ) \;=\; \exists b \in\ B \cap F(\ P, \mathtt{arb}(O)\ )\ :\ a(\ v,\ O'(b),\ B - \{b\}\ ) \tag{6.10}$$

$$\textbf{where } O'(b) \;=\; \{\ i \in O \mid b \notin F(P, i)\ \}.$$

Here, $O'$ is the set of occurrences that cannot be backed up using a particular choice of $b$.

The preceding case handled $O \neq \varnothing$. The case $O = \varnothing$ is quite simple, since directly from the definition of $a(\cdot)$ we have the following equality (for $v \neq v_{\min}$).

$$a(v, \varnothing, B) \; = \; a(\, \mathrm{pred}(v),\; \mathcal{I}(\mathrm{pred}(v)),\; B\,) \tag{6.11}$$

This leaves setting the base case for our recursion. This is also easily accomplished from the definition. We let $a(v_{\min}, \varnothing, \varnothing) = \text{true}$: it is indeed possible to back up no occurrences using no backup nodes.

The algorithm then checks whether $a(v_{\max}, \mathcal{I}(v_{\max}), B)$ is true for any $B \subseteq V$. Correctness of the algorithm follows from correctness of the recurrence.

Using dynamic programming we make sure that $a(\cdot)$ is only ever evaluated once for every value of the parameters; call these the dynamic programming states. Evaluating a single dynamic programming state can clearly be done in polynomial time. For the runtime up to polynomial factors, it then remains to bound the number of different dynamic programming states. The total number of states is

$$\begin{aligned}\sum_{v \in V} \left(2^{|\mathcal{I}(v)|} \cdot 2^{|V|}\right) &\overset{\text{def}}{=} \sum_{v \in V} \left(2^{\mu(v)} \cdot 2^{n}\right) \\ &\leqslant \sum_{v \in V} \left(2^{\mu_{\max}} \cdot 2^{n}\right) \\ &= n \cdot 2^{\mu_{\max}} \cdot 2^{n}.\end{aligned}$$

This gives total running time of $\mathcal{O}^*(\, 2^{n+\mu_{\max}}\,)$. □

## 6.6. Fixed parameter complexity of backup assignment

We close this chapter by briefly discussing possible parameterisations of the Backup assignment problem with injective backup. Recall that the non-parameterised problem is $\mathcal{NP}$-complete (Theorem 6.33), but perhaps the problem is fixed parameter tractable [DF13] for some reasonable parameter.

Let $M$ be the set of nodes that have multiplicity more than one, that is,

$$M = \{\, v \in V \mid \mu(v) > 1 \,\}.$$

If $|M| = 0$ then the problem is polynomial-time solvable, because in that case any backup assignment is trivially functional; then we are in fact dealing with one-to-one backup and can apply Theorem 6.32. The hardness lies not just in the size of $M$, however, as evidenced by the following simple corollary of Theorem 6.33. Recall that $\mathcal{XP}$ is the class of parameterised problems that can be solved in $\mathcal{O}(\, n^{f(k)}\,)$ time, where $f(k)$ is some computable function of the parameter. Note that this class certainly contains all fixed parameter tractable problems. (For details, see for example [DF13].)

**Corollary 6.40.** *Assuming $\mathcal{P} \neq \mathcal{NP}$,* Backup assignment *with injective backup parameterised by $|M|$ is not fixed parameter tractable and, in fact, not even in $\mathcal{XP}$.*

*Proof.* The instances used to prove $\mathcal{NP}$-completeness of the non-parameterised case have $|M| = 1$ (see the proof of Theorem 6.33). Then Backup assignment with this parameter is $\mathcal{NP}$-complete even for parameter value 1. Given $\mathcal{P} \neq \mathcal{NP}$, this contradicts membership of $\mathcal{XP}$. □

Recall that we defined $\mu_{max}$ as the maximum multiplicity in the graph and that, by Theorem 6.39, the problem can be solved in $\mathcal{O}^*( 2^{n+\mu_{max}} )$ time. This is far from fixed parameter tractable, but suggests $\mu_{max}$ might have some relation to the hardness of an instance. At least our current $\mathcal{NP}$-completeness proof doesn't immediately disprove fixed parameter tractability like Corollary 6.40 did for $|M|$.

As most promising for fixed parameter tractability we suggest the following, larger, parameter:

$$\sum_{v \in V} \Big( \mu(v) - 1 \Big) \tag{6.12}$$

Note that multiplicity-one nodes still do not contribute to this parameter, which is a desirable property: those vertices should not be where the hardness of an instance lies.

## 6.7. Concluding remarks

We have introduced a model of recoverable routing in the single-node failure model. As with other *robust recoverability* models, the motivation is as follows. Choosing a solution that is feasible for any failure scenario is overly cautious—in our case it would only allow paths of one hop. On the other hand, unrestricted replanning in case of a failure can be too costly in terms of computational power or the information available. We therefore plan a route that, in case of failure, can be fixed easily and locally. For this model, we have resolved the basic algorithmic and complexity questions.

In this chapter we have presented several algorithms. For the polynomially-solvable case of Recoverable path we have given an $\mathcal{O}( n^4 )$-time algorithm.

For the functional and injective cases, the runtime of $\mathcal{O}^*( 2^n )$ that our algorithms achieve seems reasonable. When generalised to the one-to-one case, however, our algorithms run in $\Theta^*( 4^n )$ time. It seems to us there should be a better way to handle the one-to-one case.

Finally we suggest looking at parameterised versions of Backup assignment with injective backup.

# 7 Energy Constrained Flow: Hardness and bounded treewidth

In this chapter we study the data gathering problem in sensor networks. Sensor nodes gather data and then relay them to a base station, for as long as they do not run out of battery power. This is modelled in the energy-constrained flow problem. Packets are considered as integral units and are not splittable. The problem is to find the maximum data flow in the sensor network subject to the energy constraint of the sensors and relay nodes. We show that this *integral* version of the problem—with unsplittable packets—is strongly $\mathcal{NP}$-complete and in fact $\mathcal{APX}$-hard. It follows that the problem is unlikely to have a polynomial time approximation scheme. Even when restricted to graphs with concrete geometrically defined connectivity and transmission costs, the problem is still strongly $\mathcal{NP}$-complete. For networks with bounded treewidth greater than two, we show that the problem is weakly $\mathcal{NP}$-complete and provide pseudopolynomial time algorithms. For a special case of graphs with treewidth two, we give a polynomial time algorithm. Approximations and heuristics are described in Chapter 8.

## 7.1. Introduction

As the battery of sensor nodes is typically of limited power and non-replaceable, it is crucial to maximise the lifetime of the wireless sensor network to ensure the continuing function of the whole network. We study the situation of a data-gathering sensor network, where sensors are deployed in the field to gather data and relay the data packets, possibly via other sensor nodes, back to a base station. It is desirable to get as many data packets as possible from the source sensors to the base station, before depletion of batteries prevents further functioning of the network. This can be modelled as a maximum flow problem, subject to the energy constraint of the battery power of each sensor.

Some of the research on the maximum flow problem with energy constraints on wireless sensor networks (e.g. [CT00a, CT00b, CT04, FKKO05, HP06, OK04, XCN05, ZS03]) casts the problem into a linear programming (LP) form and calculates fractional flows. That is, they consider it feasible to split packets arbitrarily into fractional portions without overhead. The corresponding LP-formulations then admit polynomial time algorithms. These papers then present several heuristics that speed up the algorithms and compare various simulation results. A few papers [CT00b, FKKO05, HP06, KDN03] modified the polynomial time approximation scheme (PTAS) of Garg and Könemann [GK98] to obtain fast approximation algorithms.

As data packets are usually quite small, there are situations where splitting of packets into fractional ones is neither desirable nor practical due to overheads. This integral flow routing for wireless sensor networks has also been considered in [GDPV03, GDP04, KDN03, PCV04], where integer programming (IP) formulations are given, each with slightly different constraints.

In this chapter, we use a simple wireless sensor network model where data packets are considered as integral units that cannot be split, with the bare minimum of constraints in the LP-formulation, reminiscent of the standard maximum flow IP-formulation [AMO93]. We call this the *Integer* Maximum Flow problem for Wireless Sensor Network with Energy Constraint: the Integer Max-Flow WSNC problem. We show that even this simple version of the problem is *strongly* $\mathcal{NP}$-complete, and thus unlikely to have a fully polynomial time approximation scheme (FPTAS) or a pseudopolynomial time algorithm unless $\mathcal{P}=\mathcal{NP}$. This result also holds for a class of graphs with geometrically defined connectivity and transmission costs, even when the nodes lie on a line. Furthermore, we show that even for a special fixed range model, the problem is $\mathcal{APX}$-hard, thus unlikely to have even a PTAS (unless $\mathcal{P}=\mathcal{NP}$). In Chapter 8 we will consider the question of approximability further.

Many hard problems have polynomial time solutions when restricted to networks with bounded treewidth (see e.g. [Bod93]). However, we show that for networks with bounded treewidth greater than two, the Integer Max-Flow WSNC problem is *weakly* $\mathcal{NP}$-complete. We provide pseudopolynomial time algorithms to compute integer maximum flows in this case. For a special case of graphs that have treewidth two, namely those graphs that have treewidth two when we add an edge from the single source to the sink, we provide a polynomial time algorithm.

This chapter is organised as follows. First we describe the model and the problem in detail. Sections 7.3 through 7.5 cover the complexity issues. Starting at Section 7.6 we give the results for networks with bounded treewidth.

## 7.2. Energy constrained flow

In this section, we discuss the model we use and the precise formulation of the energy constrained flow problem. We also discuss some variants of the problem.

### 7.2.1. Model

Our model of a sensor is based on the *first order radio model* of Heinzelman et al. [HCB00]. A sensor node has limited battery power that is not replenishable. It consumes an amount of energy $\varepsilon_{elec} = 50\text{nJ/bit}$ to run the receiving and transmitting circuitry and $\varepsilon_{amp} = 100\,\text{pJ/bit/m}^2$ for the transmitter amplifier. In order to receive a $k$-bit data packet, a sensor has to expend $\varepsilon_{elec} \cdot k$ energy, while to transmit the same packet from sensor $i$ to sensor $j$ will cost $\varepsilon_{elec} \cdot k + \varepsilon_{amp} \cdot k \cdot d_{ij}^2$ energy, where $d_{ij}$ is the distance between sensors $i$ and $j$. Note that the cost of a transmission increases quadratically with the distance between sender and receiver. This is also the assumption underlying the geometric *SINR* model (or: physical model) used in Chapters 4 and 5; see Section 2.6 in the preliminaries for some discussion.

We model a wireless sensor network as a *directed* graph $G = (N, A)$, where $N = \{1, 2, \ldots, n\} \cup \{t\}$ are the $n$ sensor nodes along with a special non-sensor sink node $t$, and $A$ is the set of directed arcs $(i, j)$ connecting node $i$ to node $j$, with $i, j \in N$. A sensor node $i$ has energy capacity $E_i$ and each arc $(i, j)$ has a cost $e_{ij}$, the energy cost of receiving a packet and then transmitting it from node $i$ to node $j$. We assume that no data is held back in intermediate nodes other than $t$, that is, data that flows in will flow out again, subject to the battery constraint of these nodes. All $E_i$ and $e_{ij}$ are non-negative *integer* values.

The general model assumes that each sensor can adjust its power range for each transmission. We also consider in the next section the fixed range model, where each sensor has only a few *fixed* power settings. All our graphs are assumed to be *connected*. Furthermore, for each arc $(i, j) \in A$, there is a directed path from a source node to the sink node that uses this arc.

### 7.2.2. The Problem

Given a wireless sensor network $G = (N, A)$, there is a set $S \subseteq N$ of *source* sensor nodes, used for gathering data. The sink node $t$ is a *base station* and is equipped with electricity and thus has unlimited energy to receive all packets. The remaining nodes are just *relay* nodes, used to transfer data packets from the source nodes to the sink node. One would like to transmit as many packets as possible from the source nodes to the sink node. This is feasible as long as the battery power in the network suffices to do so. The transmission process can be viewed as a *flow* of packets from the sources to the sink. The problem is then to find the *maximum flow* of data packets in the network subject to the battery power constraint.

We assume that the data packets are quite small and that it is therefore neither reasonable nor practical to split them further into fractional portions. A *flow* $f$ is a function that assigns to each arc $(i,j)$ a non-negative integer value. We write $f_{ij}$ for the value assigned by $f$ to the arc $(i,j)$. This corresponds to the number of packets being sent from $i$ to $j$. A flow is *feasible* if

$$\sum_j f_{ij} \cdot e_{ij} \leqslant E_i \tag{7.1}$$

for all nodes $i \in N$, where the sum is taken over all $j$ with $(i,j) \in A$; that is, the flow through a node cannot exceed the battery capacity of the node.

We can now formulate the maximum flow problem for wireless sensor networks as the problem of determining the maximum number of packets that can be received by the sink node. We call this problem the Integer Maximum Flow problem for Wireless Sensor Networks with energy Constraints or *Integer Max-Flow WSNC problem* for short. The problem has the following Integer Linear Programming formulation.

Maximise

$$F = \sum_{j \in N} f_{jt} \tag{7.2}$$

Subject to:

$$f_{ij} \text{ integer}, \qquad \forall (i,j) \in A \tag{7.3}$$

$$f_{ij} \geqslant 0, \qquad \forall (i,j) \in A \tag{7.4}$$

$$\sum_{j \in N} f_{ij} = \sum_{j \in N} f_{ji}, \qquad \forall i \in N - S - \{t\} \tag{7.5}$$

$$\sum_{j \in N} e_{ij} \cdot f_{ij} \leqslant E_i. \qquad \forall i \in N \tag{7.6}$$

Condition (7.5) is the conservation of flow constraint. It simply states that with the exception of the source and sink nodes, every node must send along the packets that it has received. Condition (7.6) is the energy constraint for the feasible flow: the energy needed to (receive and) transmit packets must be within the capacity of the battery power of each node. This condition also distinguishes the (integer) max-flow WSNC problem from the standard max-flow problem: there, the constraint condition is just $f_{ij} \leqslant c_{ij}$, where $c_{ij}$ is the flow capacity of arc $(i,j)$.

We note that without loss of generality, we can augment the network with a *super source* node $s$ with unlimited energy and connect it to all the 'real' source nodes with some fixed cost. We can then view the network as having a single source and a single sink with a single commodity, subject to the battery energy constraint. However, note that this may affect the treewidth of the network; the results for networks of bounded treewidth in section 7.7 assume a single source.

This model does not include a 'fairness' constraint: it maximises the number of packets reaching the sink, even if all come from just one sensor. Note that the flows in our hardness proofs do exhibit a fair flow, where all source nodes send an equal amount of packets.

### 7.2.3. Variants

Other variants of the problem formulation exist for wireless sensor networks with energy constraint. For example, Floréen et al. [FKKO05] use the following energy constraint in their LP-formulation of the problem:

$$\sum_{j \in N} \tau_{ij} \cdot f_{ij} + \sum_{j \in N} \rho \cdot f_{ij} \leqslant E_i, \ \forall i \in N,$$

where the parameters $\tau_{ij}$ is the energy expended in sending a packet from node $i$ to node $j$ and $\rho$ is the corresponding energy for receiving a packet. Their objective function in the LP-formulation is also slightly different and has some extra parameters that are non-integral.

Chang and Tassiulas [CT04] give an LP-formulation similar to ours for the single commodity case, except they formulate the problem over the set of all paths from $s$ to $t$ and allow fractional packets. They also consider the multi-commodity case.

Kalpakis et al. [KDN03] give an IP-formulation where the lifetime of the system is measured in terms of the number of rounds $T$ and then the objective is to maximise $T$.

## 7.3. Hardness on general graphs

We will first look at the complexity of the problem on general graphs with arbitrary costs at the arcs. Since each data packet is a self-contained unit and cannot be split, the corresponding LP formulation is an integer program (IP) and may no longer have a polynomial time solution. In fact, we prove that the problem is *strongly* $\mathcal{NP}$-complete.

**Theorem 7.1.** *The decision variant of the Integer Max-Flow WSNC problem is strongly $\mathcal{NP}$-complete.*

*Proof.* The problem is clearly in $\mathcal{NP}$. To prove strong $\mathcal{NP}$-completeness we design a pseudopolynomial reduction from 3-PARTITION, a problem that is well known to be strongly $\mathcal{NP}$-complete [GJ79].

| | **3-Partition** |
|---|---|
| *Instance:* | Multiset $\mathcal{S}$ of $n = 3m$ positive integers. |
| | A positive integer $B$. |
| | Each $x_i \in \mathcal{S}$ has $B/4 < x_i < B/2$. |
| *Question:* | Can $\mathcal{S}$ be partitioned into $m$ subsets (each necessarily containing exactly three elements) such that the sum of each subset is equal to $B$? |

For any instance $I$ of the 3-Partition problem we create an instance $I'$ of a wireless sensor network as follows. Each number $x_i \in \mathcal{S}$ corresponds to a sensor relay node $r_i$. Additionally, we have $m$ source nodes $s_1, \dots, s_m$ each having exactly $B$ energy. These source nodes play the role of the subsets. Now connect each of the source nodes $s_j$ with all the relay nodes $r_i$ with arc cost $e_{ji} = x_i$. The intention is that it will cost each source node exactly $x_i$ energy to send one packet to relay node $r_i$. We further connect all the relay nodes to a sink node $t$. Each relay node $r_i$ has energy $E_i = B$ and arc cost $e_{it} = B$: just sufficient energy to send exactly one packet to the sink node.

Then our instance of the 3-Partition problem has a partition into $m$ subsets $\mathcal{S}_1, \dots, \mathcal{S}_m$, each with sum equals $B$ if and only if for each subset $\mathcal{S}_i = \{x_{i1}, x_{i2}, x_{i3}\}$ the source node $s_i$ sends three packets, one each to relay nodes $r_{i1}, r_{i2}, r_{i3}$ consuming the energies $x_{i1}, x_{i2}, x_{i3}$, thus draining all of its battery power of $B = \sum_{j=1}^{3} x_{ij}$. This will give a maximum flow of $n = 3m$ packets for the whole network. Thus, 3-Partition reduces to the question whether the WSNC network can transmit at least $3m$ packets to the sink.

The given reduction is a pseudopolynomial reduction and thus we conclude that the Integer Max-Flow WSNC problem is strongly $\mathcal{NP}$-complete. □

**Corollary 7.2.** *The Integer Max-Flow WSNC problem has no fully polynomial time approximation scheme (FPTAS) and no pseudopolynomial time algorithm, unless $\mathcal{P} = \mathcal{NP}$.*

*Proof.* This follows from the fact that a strongly $\mathcal{NP}$-complete problem has no FPTAS and no pseudopolynomial time algorithm, unless $\mathcal{P} = \mathcal{NP}$. (See [GJ79]). □

We show in Section 7.5 that even in a restricted case, the Integer Max-Flow WSNC problem is $\mathcal{APX}$-hard, that is, the problem does not even have a PTAS unless $\mathcal{P} = \mathcal{NP}$. This hardness result does no prevent us from searching for constant-factor approximation algorithms for the problem. This was solved by Nutov [Nut08] who gives a c-approximation algorithm with c around 1/16. We discuss the approximability question, including Nutov's result, further in Chapter 8.

## 7.4. Hardness on geometric instances

We will now look at the geometric version of the problem in which the nodes are concretely embedded in space. In this version, each node has a position, and transmitting to a node at distance $d$ costs $d^2$ energy. This quadratic cost is a typical model of radio transmitters; recall that we used it before, in Chapters 4 and 5 where we used the geometric *SINR* model.

**Definition 7.3.** A *geometric configuration* is a complete graph where each node $i \in N$ has a location $p(i)$ and initial battery capacity $E_i$. The cost of the edge between vertices $i$ and $j$ is $e_{ij} = |p(i) - p(j)|^2$.

In this section we show that the Integer Max-Flow WSNC-problem remains strongly $\mathcal{NP}$-complete when restricted to geometric configurations, even when all nodes lie on a line. We first consider the case where we allow variable battery capacities $E_i$ (Theorem 7.4) and then give a more elaborate construction, still with all nodes on a line, where all nodes have equal battery capacity (Theorem 7.9).

**Theorem 7.4.** *The Integer Max-Flow WSNC-problem is strongly $\mathcal{NP}$-complete on geometric configurations on the real line.*

*Proof.* Again, we give a pseudopolynomial reduction from 3-Partition. First we describe how to construct a geometric configuration $\mathbf{C}$ on the real line, where the Integer Max-Flow WSNC-problem is equivalent to a given instance of 3-Partition. We then construct an equivalent configuration $\mathbf{C_P}$ that can be described in polynomial size. These steps together show that the Integer Max-Flow WSNC-problem is strongly $\mathcal{NP}$-complete on geometric configurations.

As before, we have $m$ source nodes $s_1, \dots, s_m$. Each starts with $B = \sum x_i/m$ energy. These source nodes again play the role of the subsets and we place them all at the origin of our geometric configuration, that is, $p(s_i) = 0$ for all $i$. This also means that the cost of sending from one source node to another is zero, but such flow can be disregarded since all these nodes are sources.

Corresponding to each $x_i \in \mathcal{S}$ we have a non-source relay node $r_i$ that serves the same purpose as before: to receive one packet from a source node, costing $x_i$ energy for this source node, and relay the packet to the sink. By setting $p(r_i) = \sqrt{x_i}$ we achieve that a source node must use $x_i$ energy to send a message to $r_i$.

Finally, we place a sink node $t$ with $p(t) = \sqrt{B}$. We want each relay node to have just enough energy to send exactly one packet to the sink, so we set $E_{r_i} = (p(t) - p(r_i))^2 = B + x_i - 2\sqrt{B}\sqrt{x_i}$.

This concludes the construction of our geometric configuration $\mathbf{C}$. The construction is illustrated in Figure 7.1.

**Lemma 7.5.** *Suppose we have a flow of value $n$ in $\mathbf{C}$. Then every relay node receives exactly one packet from a source and sends it to the sink.*

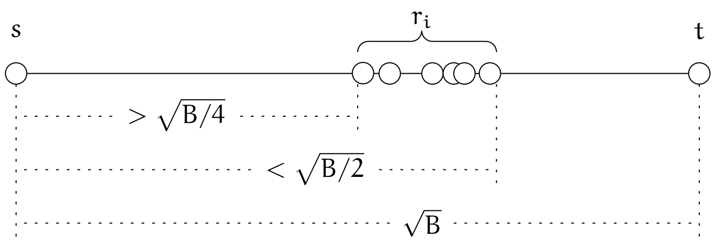


**Figure 7.1.** Configuration **C**.

*Proof.* If $n$ packets reach the sink, then $n$ packets must have left the sources. By the restriction on the values $x_i$ of the 3-Partition instance, each edge leaving the sources costs strictly more than $B/4$ energy. Since the source nodes start with $B$ energy, no source node can send more than 3 packets. There are only $m = n/3$ source nodes, so every source node must send exactly 3 packets. In particular, no packets are sent directly from a source node to the sink, as this would use up all energy of the source node. Therefore, the sink node only receives packets from relay nodes. No relay node can afford to send more than one packet to the sink, so in fact every relay node sends exactly one packet to the sink. □

Using this lemma, the following can now be shown in the same way as Theorem 7.1 for the case on arbitrary graphs.

**Proposition 7.6.** *The configuration* **C** *has a solution of the Integer Max-Flow WSNC-problem with* $n$ *packets if and only if the corresponding* 3-Partition *instance is* Yes.

Note that this does not yet give an $\mathcal{NP}$-hardness proof for the Integer Max-Flow WSNC-problem on geometric configurations on the line, as configuration **C** has nodes at real-valued coordinates and the specification of the location of the points in **C** contains square roots. We shall now construct a geometric configuration $\mathbf{C_P}$ that is, for our purposes, equivalent to **C**, but whose positions are all polynomially representable rational numbers.

We do this by choosing the locations as integer multiples of some $\varepsilon$ (value to be determined later), rounding down. The initial power of the batteries also needs to be quantised. We give the source nodes exactly $B$ energy, which is already integer. We give the relay nodes precisely enough energy to send one packet to the sink; this amount can be calculated from the actual distance in $\mathbf{C_P}$.

**Lemma 7.7.** *The value for* $\varepsilon$ *can be chosen such that* $\mathbf{C_P}$ *is equivalent to* **C** *and can be represented in polynomial size.*

*Proof.* Consider a grid of precision $\varepsilon$, onto which the positions are rounded down. Rounding down makes communication from sources to relay nodes cheaper. For the $\mathcal{NP}$-completeness construction we need that a source cannot send 4 packets. From the 3-Partition instance we have that the $x_i$ are strictly bigger than $B/4$. Since each $x_i$ is an integer, we have in particular that

$x_i \geqslant B/4 + \frac{1}{4}$. This gives the following constraint on $\varepsilon$:

$$4\Big(\sqrt{\frac{B+1}{4}} - \varepsilon\Big)^2 \;>\; B.$$

This is equivalent to

$$\varepsilon \;<\; \frac{\sqrt{B+1} - \sqrt{B}}{2}. \tag{7.7}$$

That is, if we round coordinates down to a multiple of $\varepsilon$ and (7.7) holds, no source node can send more than three packets. Note that this bound is $\Theta(1/\sqrt{B})$.

There is a further constraint on $\varepsilon$. We do not want to enable flows that do not correspond to a 3-Partition. Therefore, a source node should, with its $B$ energy, not be able to send packets to a set of relay nodes if the corresponding $x_i$ sum to strictly more than $B$. This gives the following constraint on $\varepsilon$, for all $a, b, c \in \mathcal{S}$:

$$a + b + c \geqslant B + 1 \;\implies\; (\sqrt{a} - \varepsilon)^2 + (\sqrt{b} - \varepsilon)^2 + (\sqrt{c} - \varepsilon)^2 > B.$$

This holds when the following condition holds; we solve the equation by setting $a + b + c = B + 1$.

$$\varepsilon \;<\; \frac{\alpha - \sqrt{\alpha^2 - 3}}{3}, \;\text{ where }\; \alpha = \sqrt{a} + \sqrt{b} + \sqrt{c}.$$

This upperbound for $\varepsilon$ is monotonically decreasing in $\alpha$ (for $\alpha > \sqrt{3}$) and is therefore minimised by maximising $\alpha$. This occurs at $a = b = c = (B + 1)/3$, giving

$$\varepsilon \;<\; \frac{\sqrt{B+1} - \sqrt{B}}{\sqrt{3}}. \tag{7.8}$$

This bound on $\varepsilon$ is also $\Theta(1/\sqrt{B})$.

The number of grid points for a grid of length $\sqrt{B}$ is then

$$\frac{\sqrt{B}}{\varepsilon} \;=\; \frac{\sqrt{B}}{\Theta(1/\sqrt{B})} \;=\; \Theta(B).$$

A position can therefore be represented in $\Theta(\log B)$ space, just like the numbers in the given 3-Partition instance (since they are between $B/4$ and $B/2$).

Finally, we must specify $\varepsilon$. We can take e.g.

$$\varepsilon = \frac{1}{5\lceil\sqrt{B}\rceil}.$$

This satisfies constraints (7.7) and (7.8) and can be written as a rational number in polynomial space and time. Note that we can also compute the coordinates of all relay nodes and the sink in polynomial time. □

The proof of Theorem 7.4 now follows from Lemmata 7.5, 7.7 and Proposition 7.6: the Integer Max-Flow WSNC-problem is strongly $\mathcal{NP}$-complete on geometric configurations, even when restricted to a line. □

The nodes in $\mathbf{C_P}$ have non-integer positions, but the following shows that this is not essential.

**Corollary 7.8.** *The Integer Max-Flow WSNC-problem is strongly $\mathcal{NP}$-complete on geometric configurations on a line, where each node has an integer coordinate.*

*Proof.* Transform $\mathbf{C_P}$ as follows. Multiply each position by $\varepsilon^{-1}$ and each battery capacity by $\varepsilon^{-2}$. This transformation assures that all coordinates are integer, since all positions in $\mathbf{C_P}$ are multiples of $\varepsilon$. Also, the transformed geometric configuration is equivalent to $\mathbf{C_P}$, since for any set of distances $d_i$, a bound $E$ and a value for $\varepsilon$, we have that the old situation is equivalent to the transformed situation:

$$\sum {d_i}^2 \leqslant E \iff \sum \left(\frac{d_i}{\varepsilon}\right)^2 \leqslant \frac{E}{\varepsilon^2}. \tag{7.9}$$

□

We finish this section by proving that Theorem 7.4 still holds when all battery capacities are equal.

**Theorem 7.9.** *The Integer Max-Flow WSNC-problem is strongly $\mathcal{NP}$-complete on geometric configurations on the real line, even when all battery capacities are equal.*

*Proof.* Recall configuration $\mathbf{C}$: source nodes, a cluster of relay nodes and a sink node. The battery capacity of the relay nodes was used to coerce each relay node to send exactly one packet, and to send it to the sink. Doing this for all relay nodes at once required varying battery capacities; we shall now use a more elaborate construction $\mathbf{C}'$.

Where before we had a single cluster of relay nodes, we shall now have $n$ clusters of relay nodes, spaced $\sqrt{B}$ apart. Then the geometric configuration $\mathbf{C}'$ is as follows.

- Source nodes: $s_1 \dots s_m$, with $p(s_i) = 0$.
- Relay nodes: for each $x_i$, there are $n$ relay nodes, $r_{i,1} \dots r_{i,n}$, with $p(r_{i,j}) = \sqrt{x_i} + (j-1)\sqrt{B}$.
- A sink node $t$, with $p(t) = n\sqrt{B} + \sqrt{x_1}$.
- All nodes have starting battery capacity of $B$.

The configuration is visualised in Figure 7.2. We shall again prove that a flow of value $n$ is possible in this configuration if and only if there is a 3-partition in the given instance.

For convenience, let the $x_i$ be indexed in order of non-decreasing value. Furthermore, consider multiple relay nodes at the exact same location, which occurs

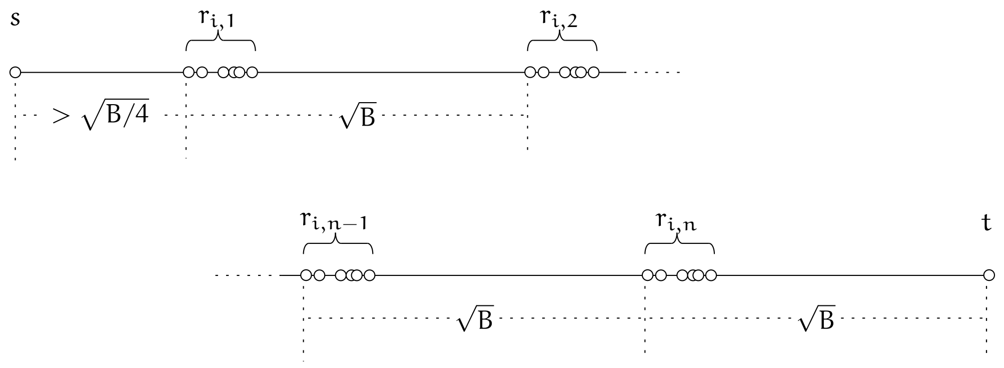


**Figure 7.2.** Configuration $\mathbf{C}'$.

if $x_i = x_j$ for some $i \neq j$. Then the relay nodes for $x_i$ and $x_j$ are interchangeable in any solution. This allows us to proceed in the following proof as if $i < j$ implies $x_i < x_j$.

First, suppose there is a solution to the given instance of 3-Partition. Similar to the earlier proofs, each source node in the configuration can send three packets, such that each relay node in the first cluster (i.e., each $r_{i,1}$) receives exactly one packet. Now, each relay node except those in the last cluster forwards its packet to $r_{i,j+1}$; the relay nodes in the last cluster (that is, all $r_{i,n}$) forward the packet to the sink. This gives a flow of value $n$.

Now suppose we have a flow of value $n$ in $\mathbf{C}'$. We will show that the given instance of 3-Partition has a solution. Note that the gap between the last relay node in the last cluster and the sink is

$$\begin{aligned} p(t) - p(r_{n,n}) &= \sqrt{B} + \sqrt{x_1} - \sqrt{x_n} \\ &> \sqrt{B} + \sqrt{B/4} - \sqrt{B/2} \approx 0.79\sqrt{B}. \end{aligned}$$

The energy required for sending more than one packet over this distance —that is, from any node in the last cluster to the sink— is at least twice this distance squared ($\approx 1.26B$), which no node can afford. So if there is $n$ flow in the network, each relay node in the last cluster must send exactly one packet to the sink. Similarly, every node in cluster $j < n$ must send exactly one packet to a node in cluster $j + 1$.

We cannot rule out that some relay nodes send packets to other nodes in the same cluster, in addition to their packet to the next cluster. We shall prove that some nodes (and in particular, all nodes in the first cluster) cannot do this.

First consider $r_{1,1}$, the leftmost relay node. By the preceding argument, it must send a packet to the next cluster. In particular, this must be $r_{1,2}$: all other nodes in cluster 2 are more than $\sqrt{B}$ away and therefore cannot be reached. Note that sending this particular packet depletes the battery of $r_{1,1}$ exactly.

**Definition 7.10.** A node is called *forced* if, when there is $n$ flow in the network, it necessarily spends all its energy to send exactly one packet.

As we have just argued, $r_{1,1}$ is *forced*. The same argument shows that $r_{1,2}$ must send the packet to $r_{1,3}$ and so forth, on to the sink. This way, all $r_{1,j}$ are forced. Therefore, if there is a flow of $n$ packets, one of the packets must take the path $p_1 = ( s_i, r_{1,1}, r_{1,2}, \dots, r_{1,n}, t )$, for some source node $s_i$.

**Lemma 7.11.** *For all $i, j$, if $i + j \leqslant n + 1$ then relay node $r_{i,j}$ is forced.*

*Proof.* We just argued that all $r_{1,j}$ are forced. We now look at $r_{2,1}$, the second relay node in the first cluster. It must send a packet to the next cluster. It cannot send it to $r_{1,2}$, however: the battery of that node is already completely depleted by the packet it must send for path $p_1$, so any packets sent there would not be able to leave. The packet can be sent to $r_{2,2}$, at cost exactly $B$, but it cannot be sent any further. So $r_{2,1}$ is also forced, and again there is only one choice for where to send the packet: via all the relay nodes for $x_2$. In this way, all $r_{2,j}$'s are forced, except for $r_{2,n}$: the distance between $r_{2,n}$ and $t$ is less than $\sqrt{B}$ and its battery is not exhausted at once.

We now show the lemma with induction on $i$. We argued above the cases $i = 1$ and $i = 2$. Suppose the result holds for $i - 1$, $i \geqslant 3$. Consider a relay node $r_{i,j}$, $i + j \leqslant n + 1$. This node $r_{i,j}$ must send a packet to cluster $j + 1$. Call the node it sends the packet to $r_{i^*,j+1}$. Clearly, $i^* \leqslant i$, since otherwise the transmission would cost too much energy. We obtain a contradiction if $i^* < i$. In that case $r_{i^*,j}$ is forced (induction hypothesis), so it sends a packet to $r_{i^*,j+1}$. Also, $r_{i^*,j+1}$ is forced (again, induction hypothesis), so it sends exactly one packet. Thus $r_{i,j}$ cannot send a packet to $r_{i^*,j+1}$. This leaves $i^* = i$ as the only option, and hence $r_{i,j}$ is forced. □

In particular, we have that all $r_{i,1}$ are forced. This means that all relay nodes in the first cluster receive exactly one packet from a source node, and spend all their energy sending this single packet to the next cluster. This is only possible if the original $x_i$ have a valid 3-partition: there is $n$ flow in $\mathbf{C}'$ if and only if the original 3-Partition instance has a solution.

Again, we need to consider polynomial representation. We will round down the positions in $\mathbf{C}'$ to multiples of some $\varepsilon$ and call the resulting geometric configuration $\mathbf{C}'_\mathbf{P}$.

**Lemma 7.12.** *The value for $\varepsilon$ can be chosen such that $\mathbf{C}'_\mathbf{P}$ is equivalent to $\mathbf{C}'$ and can be represented in polynomial size.*

*Proof.* In addition to constraints (7.7) and (7.8) there is now an additional constraint. The construction requires that the *forced* nodes expend all their energy reaching a node at distance $\sqrt{B}$. After rounding, this distance might be only $\sqrt{B} - \varepsilon$. This is no problem as long as the node's remaining energy is not enough to reach any other node. The smallest distance between two nodes, again considering the rounding, is $\sqrt{B/2} - \sqrt{(B-1)/2} - \varepsilon$. This gives the constraint

$$(\sqrt{B} - \varepsilon)^2 \;+\; (\sqrt{B/2} - \sqrt{(B-1)/2} - \varepsilon)^2 \;>\; B \tag{7.10}$$

This bound on $\varepsilon$ is $\Theta(\frac{1}{B\sqrt{B}})$ and can be satisfied by choosing e.g.

$$\varepsilon = \frac{1}{5B\lceil\sqrt{B}\rceil}.$$

This yields a total of $\Theta(nB^2)$ grid points, leading to $\Theta(\log nB)$ space per node. □

This concludes the proof of Theorem 7.9: the Integer Max-Flow WSNC-problem is strongly $\mathcal{NP}$-complete on geometric configurations on a line, even when all nodes have equal battery capacity. □

## 7.5. Hardness of approximation, fixed range model

Now we return to the case for arbitrary directed graphs. Suppose that every sensor node has only a *fixed* number of power settings, that is, there is just a fixed number of different energy values in each node's outgoing arcs. For example, there may be only *one* setting, so that every node within some range is considered a neighbor; or perhaps there are only *two* settings: *short* and *long* range power settings.

It turns out that for the case when there is only one power setting, there is an easy solution. Since now the energy cost $e_{ij}$ is the same for all neighbors $j$, the maximum flow capacity $f_{ij} = \lfloor E_i/e_{ij} \rfloor$ is also fixed for all outgoing arcs of node $i$. We can then transform the sensor network into a regular flow network by using the splitting technique in flow networks as follows. (See for example the book by Ahuja, et al. [AMO93].) Split each node $i$ into two nodes $i$ and $i'$ and connect them with an arc with capacity $c_{ij} = \lfloor E_i/e_{ij} \rfloor$. The capacity of the original arcs $(i,j)$ will also all be set to $c_{ij}$, for all $(i,j) \in A$. We then have a new graph that is a flow network with twice as many nodes and $n$ additional arcs. Then it is easy to see that this variant of the Max-Flow WSNC problem is just the standard Max-Flow Min-Cut problem and has a polynomial time algorithm of $\mathcal{O}(n^3)$, even in the integer case. This fact has also been noted by Chang and Tassiulas [CT00b] and others. For the sake of completeness, we record this fact here.

**Proposition 7.13.** *If there is only one power setting at each sensor node, then there is a polynomial time algorithm to solve the Integer Max-Flow WSNC problem.*

The situation changes when the number of fixed power settings is increased to two.

**Theorem 7.14.** *If there are two power settings at each sensor node, then there is no PTAS for the Integer Max-Flow WSNC problem, unless $\mathcal{P}$=$\mathcal{NP}$.*

*Proof.* We reduce a restricted version of the Generalised Assignment Problem (GAP) by Chekuri and Khanna [CK05] to the Integer Max-Flow WSNC problem with two power settings.

| | **2-size 3-capacity Generalised Assignment Problem (2GAP-3)** |
|---|---|
| *Instance:* | Set $\mathcal{B}$ of $m$ bins |
| | Each bin $j$ has capacity $c(j) = 3$ |
| | Set $\mathcal{S}$ of $n$ items |
| | For each item $i \in \mathcal{S}$ and bin $j \in \mathcal{B}$: a size $s(i,j) = 1$ or $s(i,j) = 1+\delta$ (for some $\delta > 0$) and a profit $p(i,j) = 1$. |
| *Question:* | Find a subset $\mathcal{U} \subseteq \mathcal{S}$ of maximum profit such that $\mathcal{U}$ has a feasible packing in $\mathcal{B}$. |

Chekuri and Khanna [CK05] show that the 2GAP-3 problem is $\mathcal{APX}$-hard, hence it does not have a PTAS (unless $\mathcal{P}$=$\mathcal{NP}$).

Given an instance I of 2GAP-3 we create an instance $I'$ of the Integer Max-Flow WSNC as follows. For each bin $j \in \mathcal{B}$ we have a source node $j'$ with energy capacity $E_{j'} = 3$. Corresponding to each item $i \in \mathcal{S}$ we have a relay node $i'$. Each source node $j'$ is connected to each of the relay nodes $i'$ by an arc $(j', i')$ with energy cost $e_{j'i'} = 1$ if $s(i,j) = 1$ and $e_{j'i'} = 1+\delta$ if $s(i,j) = 1+\delta$. We also have one sink node $t$. Each of the sensor relay node $i'$ is further connected to the sink node $t$ and provided with just sufficient battery power to send only one packet to the sink node, that is, we set $E_{i'} = 1$ and $e_{i't} = 1$.

As each node $i'$ that represents an item can forward only one packet, each arc of the form $(j', i')$, $j \in \mathcal{B}$, $i \in \mathcal{S}$ can also carry at most one packet. Thus, there is a one-to-one correspondence between integer flows in I fulfilling energy constraints, and feasible packings of sets of items $\mathcal{U} \subseteq \mathcal{S}$: an item $i$ that is placed in bin $j$ corresponds to a unit of flow that is transmitted from $j'$ to $i'$ and then from $i'$ to $t$. The value of the flow equals the total profit of the packed items.

Thus, we can observe that our reduction is an AP-reduction. Since AP-reductions preserve $\mathcal{APX}$-hardness (see for example [ACG+99]), the theorem follows. □

## 7.6. Graphs with bounded treewidth

Many hard graph problems can be solved in polynomial (sometimes even linear) time when restricted to graphs of bounded treewidth. However, this is not the case for the Integer Max-Flow WSNC problem: the problem remains hard, as we will show in the remainder of this chapter. The treewidth parameter nicely delineates the classes of graphs for which the Integer Max-Flow WSNC is of apparently increasing complexity, in the following manner.

- *Graphs of treewidth one:* These are just *forests* and have a very simple linear time algorithm: remove all nodes that do not have a path to the sink $t$; compute in the resulting tree in post-order for each node the number of packets it receives from its children and then the number of packets it can send to its parent.
- *Graphs of treewidth two:* If there is a single source, and the graph with an edge added between this single source and the sink node has treewidth two, then then there is a polynomial time algorithm for the Integer Max-Flow WSNC problem. (Section 7.7.) The general case remains open.
- *Graphs of bounded treewidth greater than two:* We show that in this case, the problem is *weakly* $\mathcal{NP}$-complete and give a pseudopolynomial-time algorithm for this class. (Section 7.8.)
- *Graphs of unbounded treewidth:* The problem is *strongly* $\mathcal{NP}$-complete and even APX-hard, as shown in the previous sections.

## 7.7. Algorithm for graphs of treewidth two

In this section, we will show that the Integer Max-flow WSNC problem (with one source and one sink) is polynomial-time solvable if the treewidth of the graph obtained by adding an edge from the source to the sink is bounded by two. While this is a somewhat specific case, we think this case is interesting because it partially bridges the gap between the trivial case of trees and the $\mathcal{NP}$-hard case of treewidth three, and because the algorithmic technique is of some interest: unlike most algorithms that exploit treewidth, it does not use dynamic programming but a reduction strategy.

For a simple description of the algorithm, we generalise our problem in two ways. First, we allow *parallel* arcs. As a consequence, we need to change our notation somewhat, and denote an arc with a name, instead of by a tuple of nodes. These parallel arcs may require a different energy per packet that is transmitted across them. Secondly, we assume that each arc $p$ has a capacity $c_p$. The capacity of an arc is a positive integer and denotes the maximum number of packets that can be transmitted across the arc. If we have an instance without capacity, we can set, for each arc $p = (i, j)$, $c_p = \lfloor E_i/e_{ij} \rfloor$. Arcs with zero capacity can be removed.

Given a directed graph $G = (N, A)$, with a source $s$ and a sink $t$, we build the undirected graph $G^u = (N, A^u)$ as follows:

- There is an edge in $A^u$ between $i$ and $j$, if and only if there is at least one arc from $i$ to $j$ and/or at least one arc from $j$ to $i$ in $A$.
- We add an edge from $s$ to $t$.

$G_u$ will be used as an auxiliary graph in subsequent constructions.

**Theorem 7.15.** *The Integer Max-flow WSNC problem, with parallel arcs and arc capacities and with a single source* $s$ *and sink* $t$, *can be solved in* $\mathcal{O}(m \log \Delta)$ *time on directed graphs* $G = (N, A)$ *such that the graph* $(N, A \cup \{st\})$ *has treewidth at most two, where* $\Delta$ *is the maximum outdegree of a node in* $G$, *and* $m = |A|$.

Our algorithm is based upon the principle of *reduction*. While most algorithms that solve problems on graphs of small treewidth are based upon dynamic programming, some other algorithms are also based upon reduction. (See e.g. [ACPS93, BvAdF01].)

First, we build $G^u$. Then we can repeat the following step. Find a vertex $i$ in $G^u$ that is neither the source $s$ nor the sink $t$, and that has degree at most two in $G^u$. (Such a vertex exists by Corollary 2.11 in the preliminaries.) We now apply a *reduction* step, that transforms $G$ to an equivalent network without $i$, that is, the number of vertices is decreased by one. We first describe the reduction step, and then will discuss a more efficient implementation.

Let $i$ be a vertex in $G^u$ that is unequal to $s$ and $t$, and that has degree at most two. If $i$ has degree 0 then clearly we can remove $i$ from $G$, and obtain an equivalent network. Suppose now that $i$ has degree one in $G^u$. This means that there is a vertex $j$, such that all arcs in $G$ with $i$ as tail have $j$ as head, and all arcs with $i$ as head have $j$ as tail. Note that there always is an optimal flow that does not transmit any packet to and from $i$. Thus, we can remove $i$ and all arcs that have $i$ as one of its endpoints.

We now look at the most interesting case, namely that $i$ has degree exactly two in $G^u$. Suppose the neighbors of $i$ in $G^u$ are $j$ and $k$. The arcs with $i$ as one of its endpoints are of the form:

- from $j$ to $i$,
- from $i$ to $k$,
- from $k$ to $i$,
- from $i$ to $j$.

Call the arcs of the form $(j, i)$ and $(i, k)$ *forward arcs*, and arcs of the form $(k, i)$ and $(i, j)$ *backward arcs*. Consider a flow with maximum value that has as additional condition that the total energy used by all nodes is minimal. In such a flow, either no packets will be transmitted over the forward arcs, or no packets will be transmitted over the backward arcs: if both packets are transmitted over forward and backward arcs, then we can cancel some and obtain a feasible flow with the same value but smaller total energy. For this reason, we can handle forward and backward arcs independently.

Consider all the forward arcs. We first compute a bound on the number of packets that $i$ can transmit to $k$, as follows. Suppose there are $b$ arcs of the form $(i, k)$. Sort these in order of non-decreasing energy per packet. Let the arcs have energy per packet $e_1 \leqslant e_2 \leqslant \cdots \leqslant e_b$, and the corresponding capacities of the arcs $c_1, c_2, \ldots, c_b$. For all $q$, $1 \leqslant q \leqslant b$, there is always an optimal flow that transmits a packet on the $q^{th}$ arc only when all $p^{th}$ arcs, with $p < q$, have totally used up their capacity. Thus, we can compute a bound $C_{ik}$ on the number of packets that $i$ can transmit to $k$ as follows.

If $\sum_{p=1}^{b} e_p \cdot c_p \leqslant E_i$, then take $C_{ik} = \sum_{p=1}^{b} c_p$. Otherwise, suppose that

$$\sum_{p=1}^{q} e_p \cdot c_p \leqslant E_i < \sum_{p=1}^{q+1} e_p \cdot c_p$$

that is, we have sufficient energy to use all the capacity of the first $q$ arcs $(i, k)$, but not for the first $q + 1$ arcs. We thus use all capacity of these first $q$ arcs, and possibly some part of the capacity of the $(q + 1)^{th}$ arc. Note that, after we used up all the capacity of the first $q$ arcs, we only have energy left to transmit at most

$$\left\lfloor \left( E_i - \sum_{p=1}^{q} e_p \cdot c_p \right) / e_{q+1} \right\rfloor$$

packets. Thus, we set

$$C_{ik} = \sum_{p=1}^{q} e_p \cdot c_p + \left\lfloor \frac{E_i - \sum_{p=1}^{q} e_p \cdot c_p}{e_{q+1}} \right\rfloor .$$

Observe that when $C_{ik}$ packets arrive at $i$, they all can be forwarded to $k$, but we can never transmit more than $C_{ik}$ packets from $i$ to $k$.

In the reduction, we build a new graph $G'$, where we have removed $i$ and its incident arcs, and added possibly a number of arcs between $j$ and $k$ (possibly in both directions). $G'$ has vertex set $N - \{i\}$, and each arc in $G$ that does not involve $i$ is also an arc in $G'$. The arcs of the form $\{j, k\}$ in $G'$ are obtained as follows.

Suppose there are $d$ arcs of the form $\{j, i\}$ in $G$. Sort these in order of non-decreasing energy, and suppose these arcs have energies $e_1 \leqslant e_2 \leqslant \cdots \leqslant e_d$, with corresponding capacities $c_1, c_2, \cdots, c_d$. Again, we may assume that we transmit only packets over the arc with energy $e_q$ if we have used up all capacities over all arcs with energy $e_p$, $p < q$.

If $\sum_{p=1}^{d} c_p \leqslant C_{ik}$, then we replace each arc of the form $(j, i)$ in $G$ by an arc of the form $(j, k)$ with the same energy cost and capacity: all packets transmitted through these arcs can be forwarded to $k$, so we have the same number of packets that can go from $j$ to $k$, using the same amount of energy at $j$. Otherwise, suppose that for some node $i$ with $0 \leqslant i < s$,

$$\sum_{p=1}^{q} c_p \leqslant C_{ik} < \sum_{p=1}^{q+1} c_p .$$

By the assumption made above, it follows that we will not be transmitting packets over the arcs with energy larger than $e_{q+1}$, and do not use the full capacity of the $(q+1)^{th}$ arc. Thus, in $G'$ we take an arc $jk$ with capacity $c_p$ and energy cost $e_p$ for each $p \leqslant q$, and an arc with capacity

$$c'_{q+1} = C_{ik} - \sum_{p=1}^{q} c_p$$

and energy cost $e_{q+1}$. (If $c'_{q+1} = 0$, we do not take the arc.) Note that the total capacity of the arcs $jk$ is now $C_{ik}$. It is not hard to see that we can transmit the same number of packets in $G$ from $j$ to $k$ via $i$ as over the new arcs $jk$ in $G'$.

For the backward arcs, we perform exactly the same step, except that the roles of $j$ and $k$ are switched. In this fashion, we obtain an equivalent network, but now with one fewer vertex.

This finishes the description of the reduction step. It is straightforward to see that this step can be done in polynomial time. Different data structures may help to speed up the performance of the reduction step.

We now describe how the mergeable-heap data structure can be used to obtain an overall time of $\mathcal{O}(m \log \Delta)$, with $\Delta$ the maximum outdegree of a node in $G$.

Note that the outdegree will never increase during a reduction step, and hence at each point in the algorithm, each node has an outdegree that is at most $\Delta$.

We use a mergeable-heap data structure, with an element for each arc. For each node $i$ and $j$, we have a set with all arcs from $i$ to $j$, if there is at least one such arc. We also maintain the graph $G^u$. As keys we use the cost of arcs; each arc has also its capacity stored. We further store the total capacity of all arcs from $i$ to $j$; we denote this value by $C'_{ij}$.

Now, a reduction step can be carried out as follows. We only consider the forward arcs as the backward arcs are similar. We use $i$, $j$, $k$, $b$, and $d$ as in the description of the reduction step for nodes of degree two in $G^u$, i.e., we look at the arcs $(j,i)$ and $(i,k)$, and obtain arcs $(j,k)$.

The procedure has three main steps. First, we consider the arcs from $i$ to $k$, and compute the value $C_{ik}$ as described above. It is not hard to see that we can do this in $\mathcal{O}(d \log d)$ time, if there are $d$ such arcs. As each of the arcs from $i$ to $k$ can be deleted after this step, and $i$ has outdegree at most $\Delta$, the total time of this step over all reductions is bounded by $\mathcal{O}(m \log \Delta)$.

The second step is only carried out if $C'_{ij} > C_{ik}$. For a faster implementation, we do not sort the arcs from $j$ to $i$, but instead we delete and/or update the capacities of the arcs from $j$ to $i$ in order of non-increasing cost.

We repeat the following step until $C'_{ji} \leqslant C_{ik}$. Obtain from the mergeable-heap data structure an arc from $j$ to $i$ with maximum cost $e_{ji}$. Using the same notation as above, suppose this is the $p^{th}$ arc, with cost $e_p$ and capacity $c_p$. If $C'_{ij} - c_p \geqslant C_{ik}$, then we delete this arc, and decrease $C'_{ij}$ by $c_p$. This operation can be done in $\mathcal{O}(\log \Delta)$ time on the mergeable-heap data structure. If $C'_{ij} - c_p < C_{ik}$, then we must update the capacity of the arc: its new capacity must be

$C_{ik} - C_{ji} + c_p$, as $C_{ji} - c_p$ is the sum of the capacities of the first $p - 1$ arcs from $j$ to $i$ (in order of non-decreasing cost). We also set $C_{ji}$ to $C_{ik}$.

One can verify that this step indeed gives the same collection of arcs and capacities as the procedure described above where we first sorted the arcs in order of non-decreasing cost. Note that all but possibly one of the forward arcs that we considered are permanently deleted. Thus, if we delete $d$ arcs in a reduction step, the step can be carried out in $\mathcal{O}(\,(d+1)\Delta\,)$ time, which amounts to a total of $\mathcal{O}(\,m \log \Delta\,)$.

In the third step, all arcs from $j$ to $i$ now become arcs from $j$ to $k$. We do not need to update this information for each arc, as it is sufficient if each set knows the endpoints of the arcs it represents. The third step has two cases. If, before the reduction, there was a data structure with arcs from $j$ to $k$, then we take the union of the data structure for arcs $(j, i)$ and arcs $(j, k)$: these arcs now all are arcs from $j$ to $k$. If there were no arcs from $j$ to $k$ before the reduction, then the data structure for the arcs $(j, i)$ now acts as data structure for arcs $(j, k)$.

The time for this third step thus is dominated by the time for one union in the mergeable-heap data structure, that is, it costs $\mathcal{O}(\,\log \Delta\,)$ per reduction. As we perform $\mathcal{O}(\,n\,)$ reductions, the total time over all third steps is $\mathcal{O}(\,n \log \Delta\,)$.

Thus, the total time is bounded by $\mathcal{O}(\,m \log \Delta\,)$. This completes the proof of Theorem 7.15.

The problem for treewidth two graphs with multiple sources is still open. If we apply the construction that transforms a graph with multiple sources to a graph with one source, then in many cases the treewidth may grow from two to three. So the Integer Max-Flow WSNC problem for general graphs of treewidth two remains open.

## 7.8. Algorithm for graphs of bounded treewidth greater than two

Our second result on treewidth is that the Integer Max-Flow WSNC problem is $\mathcal{NP}$-hard for graphs of treewidth three. This shows that a result like the previous one for treewidth two graphs cannot be found when the treewidth is three or more (assuming $\mathcal{P} \neq \mathcal{NP}$).

**Theorem 7.16.** *On graphs of treewidth three, the Integer Max-Flow WSNC problem is $\mathcal{NP}$-hard.*

*Proof.* The proof resembles the proof of strong $\mathcal{NP}$-hardness for general graphs, but we use here the (weakly) $\mathcal{NP}$-complete problem 2-Partition instead of the strongly $\mathcal{NP}$-complete problem 3-Partition.

| | **2-Partition** |
|---|---|
| *Instance:* | A multiset $\mathcal{S}$ of $n$ positive integers $a_1, \ldots, a_n$ |
| | A positive integer $B = \sum_{i=1}^{n} a_i/2$ |
| *Question:* | Can $\mathcal{S}$ be partitioned into two subsets each sum $B$? |

Given an instance $I$ of 2-Partition, we create an instance $I'$ of the Integer Max-Flow WSNC problem as follows.

Each number $a_i$ represents a sensor node $w_i$ each with energy $E_i = B$. Additionally, we have a source node $s$ and sink node $t$ along with two special nodes $v_1$ and $v_2$, each of them with energy $B$. Now connect each $v_j$ $(j = 1, 2)$ to each $w_i$ with arc cost $e_{ji} = a_i$. The source node $s$ is connected to $v_1$ and $v_2$ with arc cost $1$ each, and each node $w_i$ is connected to the sink node $t$ with cost $B$.

Now, there is an energy constrained flow with $n$ packets from $s$ to $t$, if and only if $a_1, \ldots, a_n$ can be partitioned into two subsets, each of sum $B$.

If $a_1, \ldots, a_n$ can be partitioned into two subsets $\mathcal{S}_1$, $\mathcal{S}_2$, each of sum $B$, then we can build a flow as follows: $s$ transmits $|\mathcal{S}_1|$ packets to $v_1$ and $|\mathcal{S}_2|$ packets to $v_2$. For each $a_i \in \mathcal{S}_1$, $v_1$ transmits a packet to $w_i$, and similarly for each $a_i \in \mathcal{S}_2$, $v_2$ transmits a packet to $w_i$, each packet consuming $a_i$ energy from each node. Finally, each $w_i$ now transmits one packet to $t$. This fulfills the requirements of transmitting $n$ packets from $s$ to $t$.

Suppose now there is an energy constrained maximum flow of $n$ packets from $s$ to $t$. Note that each $w_i$ can transmit at most one packet to $t$, and as there are $n$ packets to $t$, each $w_i$ must transmit exactly one packet to $t$. So, for each $i$, either $v_1$ or $v_2$ transmits a packet to $w_i$. If $v_1$ transmits a packet to $w_i$, then put $a_i$ in $\mathcal{S}_1$, otherwise put $a_i$ in $\mathcal{S}_2$. It is clear that $\mathcal{S}_1$ and $\mathcal{S}_2$ partition $\mathcal{S}$. For each element $a_i \in \mathcal{S}_1$, $v_1$ must transmit a packet with cost $a_i$, hence the sum of the elements in $\mathcal{S}_1$ is at most $B$. Similarly, the sum of the elements in $\mathcal{S}_2$ is at most $B$, and as the sum of all $a_i$'s equals $2B$, the sum of all elements in $\mathcal{S}_1$ equals $B$, and likewise for $\mathcal{S}_2$.

The constructed graph has a feedback vertex set of size two: if we remove $v_1$ and $v_2$ of the graph, we obtain a forest. Thus, it has treewidth at most three (see for example [Bod98]). Its treewidth is also at least three, as it contains the complete graph $K_4$ as a minor (remove $w_3, \ldots, w_n$, contract $s$ to $v_1$, and $w_2$ to $t$). Hence it has treewidth exactly three. $\square$

This result rules out a polynomial-time algorithm for the Integer Max-Flow WSNC problem on graphs of bounded treewidth, unless $\mathcal{P}$=$\mathcal{NP}$.

However, we show now that there is a pseudopolynomial time algorithm whenever the treewidth is bounded.

**Theorem 7.17.** *The Integer Max-Flow WSNC problem can be solved in pseudopolynomial time on graphs of bounded treewidth.*

*Proof.* We begin with a number of auxiliary definitions. A *boundary directed graph* is a triple $(N, A, X)$, with $(N, A)$ a directed graph, and $X \subseteq N$ a set of distinguished vertices, called the *boundary*.

Suppose $(N', A', X)$ is a boundary directed graph with $(N', A')$ a subgraph of the given graph $G = (N, A)$, with given costs and energies, and source $s$ and sink $t$.

A *partial energy constrained flow*, or in short *pecf*, in this boundary directed graph is a function that assigns to each arc $(i, j)$ in $A'$ a flow-value, such that

- for all $i \in N' - \{s, t\}$, the inflow of $i$ equals the outflow of $i$, and
- for all $i \in N'$, the cost of the outflow of $i$ is at most $E_i$.

The *signature* of a *pecf* is a pair $(q, \varepsilon)$, with $q$ a function that maps each $i \in X$ to the outflow of $i$ minus the inflow of $i$:

$$q(i) = \sum_{ij \in A} f_{ij} - \sum_{ji \in A} f_{ji}$$

and $\varepsilon$ a function that maps each $i \in X$ to the energy used at sensor $i$:

$$\varepsilon(i) = \sum_{ij \in A} f_{ij} \cdot e_{ij} .$$

For ease of notation, we also use $f_{ij}$ when there is no arc $(i, j) \in A$; in such a case, $f_{ij} = 0$. The algorithm we will design uses a special type of tree decomposition: a *nice tree decomposition* where, in addition to the usual requirements, we assume that the root of the tree decomposition $r$ has a bag that contains only $s$: $X_r = \{s\}$. A nice tree decomposition has four types of nodes:

- **Leaf** nodes: node $\alpha$ which is a leaf of the tree with $|X_\alpha| = 1$.
- **Introduce** nodes: node $\alpha$ with one child $\beta$ with $X_\alpha = X_\beta \cup \{i\}$ for some node $i$.
- **Forget** nodes: node $\alpha$ with one child $\beta$, with $X_\alpha = X_\beta - \{i\}$ for some node $i$.
- **Join** nodes: node $\alpha$ with two children $\beta, \gamma$ with $X_\alpha = X_\beta = X_\gamma$.

For each node $\alpha$ in the tree decomposition, we define a boundary directed graph $G_\alpha = (N_\alpha, A_\alpha, X_\alpha)$, with $N_\alpha$ the set of all vertices in $X_\alpha$ or $X_\beta$ with $\beta$ a descendant of $\alpha$, and $A_\alpha$ the set of all arcs in $A$ between vertices in $N_\alpha$.

A small modification of the construction in [BK96] shows that we can obtain in linear time, given a tree decomposition of width at most $k$ of a graph $G$, a nice tree decomposition of $G$ of width at most $k$, such that the root node $r$ has $X_r = \{s\}$.

Our algorithm mainly consists of iteratively computing for each node $\alpha$ in the tree decomposition a table consisting of all signatures of all pecfs in $G_\alpha$. The tables are computed in post-order, i.e., we compute a table for a node when the tables of its children are known.

Note that the size of each table is pseudopolynomial, as there are pseudopolynomially many signatures of pecfs in $G_\alpha$: for each $i \in X_\alpha$, $\varepsilon(i)$ is an integer between 0 and $E_i$, the outflow of $i$ is an integer between 0 and $E_i$, the inflow of $i$ is an integer between 0 and $\sum_{ji \in A} E_j$, and hence $q(i)$ is an integer between $-\sum_{ji \in A} E_j$ and $E_i$.

We now give for each of the four types of nodes a description of a procedure that computes for a node $\alpha$ the table of all signatures of all pecfs in $G_\alpha$, given such tables for the children of $\alpha$.

**Leaf nodes:** It is trivial to see that the table can be computed directly for a **Leaf** node.

**Introduce nodes:** Suppose $\alpha$ is an **Introduce** node with $\beta$ the unique child of $\alpha$ and $X_\alpha = X_\beta \cup \{i\}$. See Algorithm 7. For each element $(q^\sigma, \varepsilon^\sigma)$ from the table of signatures of pecfs in $G_\beta$, the algorithm does the following. It enumerates all assignments of flow values to all arcs $(i, j) \in A$ and $(j, i) \in A$, where $f_{ij}$ is an integer between 0 and $E_i$, and $f_{ji}$ is an integer between 0 and $E_j$. For each of these assignments of flow values, it computes a corresponding signature for a pecf in $G_\alpha$, checks if no vertex in $X_\alpha$ uses too much energy, and if so, stores it in the table for $\alpha$.

**Lemma 7.18.** *Algorithm 7 correctly computes the table of signatures of all pecfs in $G_\alpha$ for an* **Introduce** *node $\alpha$.*

*Proof.* Suppose $g$ is a pecf in $G_\alpha$. Let $g'$ be the pecf in $G_\beta$, obtained by restricting $g$ to the arcs in $G_\beta$. Consider the iteration in Algorithm 7, where $f_{ij} = g_{ij}$ and $f_{ji} = g_{ji}$ for all $j \in X_\beta$, and where $(q^\sigma, \varepsilon^\sigma)$ is the signature of $g'$. It is not hard to verify that we need to store the signature of $g$ in this iteration, if the energy constraint is satisfied. So, we store the signature of each pecf in $G_\alpha$.

Suppose we store a signature $\sigma'$ in the table, and suppose this is in the iteration where we use signature $(q^\sigma, \varepsilon^\sigma)$ for $\beta$, and values $f_{ij}$, $f_{ji}$ for all $j \in X_\beta$. Let $g'$ be the pecf in $G_\beta$ with signature $\sigma$. Let $g$ be the function, obtained by taking $g_{ij} = f_{ij}$, $g_{ji} = f_{ji}$ for all $j \in X_\beta$, and $g_{j\ell} = g'_{j\ell}$ for all arcs $j\ell$ in $G_\beta$. One can verify that $\sigma'$ is the signature of $g$. Thus, the element in the table is the signature of a pecf in $G_\alpha$. □

**Forget nodes:** Suppose $\alpha$ has one child $\beta$, with $X_\beta = X_\alpha - \{i\}$. Note that $i \neq s$, as the root of the tree decomposition contains $s$. See Algorithm 8.

Note that $G_\alpha$ and $G_\beta$ have the same vertex and arc sets. As the boundary of $G_\alpha$ is a subset of the boundary of $G_\beta$, each pecf in $G_\alpha$ is a pecf in $G_\beta$. A pecf in $G_\beta$ is also a pecf in $G_\alpha$, if and only if the inflow for $i$ equals the outflow or $i \in \{s, t\}$. This is because $i$ is a new internal vertex. Because the root of the tree decomposition contains $s$, we have $i \neq s$, and the condition can be stated as

**Algorithm 7:** Computation for **Introduce** nodes

```
Input: Tree-decomposition node $\beta$.
Output: Tree-decomposition node $\alpha$.
foreach signature $(q^\sigma, \varepsilon^\sigma)$ for $\beta$ do
    foreach assignment $f_{ij} \in \{0, \dots, E_i\}$ for all $j \in X_\beta$ with $(i,j) \in A$ do
        foreach assignment $f_{ji} \in \{0, \dots, E_j\}$ for all $j \in X_\beta$ with $(j,i) \in A$ do
            Determine a signature $\sigma' = (q', \varepsilon')$ as follows.
            $\varepsilon^{\sigma'}(i) = \sum_{j \in X_\beta} f_{ij} \cdot e_{ij}$
            foreach $j \in X_\beta$ do
                $\varepsilon^{\sigma'}(j) = \varepsilon^\sigma(j) + f_{ji} \cdot e_{ji}$
            end
            $q^{\sigma'}(i) = \sum_{j \in X_\beta} f_{ij} - \sum_{j \in X_\beta} f_{ji}$
            foreach $j \in X_\beta$ do
                $q^{\sigma'}(j) = q^\sigma(j) + f_{ji} - f_{ij}$
            end
            if for all $j \in X_\alpha$: $\varepsilon^{\sigma'}(j) \leqslant E_j$ then
                Store $\sigma'$ in the table of signatures for $\alpha$
            end
        end
    end
end
```

**Algorithm 8:** Computation for **Forget** nodes

```
Input: Tree-decomposition node $\beta$.
Output: Tree-decomposition node $\alpha$.
foreach signature $(q^\sigma, \varepsilon^\sigma)$ for $\beta$ do
    if $q^\sigma(i) = 0$ or $i = t$ then
        Determine a signature $\sigma' = (q', \varepsilon')$ as follows.
        foreach $j \in X_\alpha$ do
            $q^{\sigma'}(j) = q^\sigma(j)$
            $\varepsilon^{\sigma'}(j) = \varepsilon^\sigma(j)$
        end
        Store $\sigma'$ in the table of signatures for $\alpha$
    end
end
```

$q^\sigma(i) = 0$ or $i = t$. If this condition holds, Algorithm 8 stores the restriction of the signature to the nodes in $X_\alpha$ in the table of signatures for $\alpha$. Thus:

**Lemma 7.19.** *Algorithm 8 correctly computes the table of signatures of all pecfs in* $G_\alpha$

**Algorithm 9:** Computation for **Join** nodes

```
Input: Tree-decomposition nodes $\beta$ and $\gamma$.
Output: Tree-decomposition node $\alpha$.
1  foreach signature $(q^\sigma, \varepsilon^\sigma)$ for $\beta$ do
2      foreach signature $(q^{\sigma'}, \varepsilon^{\sigma'})$ for $\gamma$ do
3          Determine a signature $\sigma'' = (q'', \varepsilon'')$ as follows.
4          foreach $i \in X_\alpha$ do
5              $\varepsilon^{\sigma''}(i) = \varepsilon^\sigma(i) + \varepsilon^{\sigma'}(i)$
6              $q^{\sigma''}(i) = q^\sigma(i) + q^{\sigma'}(i)$
7          end
8          if for all $i \in X_\alpha$: $\varepsilon^{\sigma''}(i) \leqslant E_i$ then
9              Store $\sigma''$ in the table of signatures for $\alpha$
10         end
11     end
12 end
```

*for a* **Forget** *node $\alpha$.*

**Join nodes:** Suppose $\alpha$ has two children $\beta$ and $\gamma$, with $X_\alpha = X_\beta = X_\gamma$. Below, we assume a function to be 0 for all elements outside its domain.

**Lemma 7.20.** *Let $f$ be a function, mapping each arc in $G_\alpha$ to a non-negative integer. Then $f$ is a pecf in $G_\alpha$, if and only if there is a pecf $f'$ in $G_\beta$ and a pecf $f''$ in $G_\gamma$, with*

1. $f = f' + f''$, *and*
2. *for all* $i \in X_\alpha$, $\sum_{ij \in A,\, j \in N_i} f_{ij} \leqslant E_i$.

*Proof.* Suppose $f$ is a pecf in $G_\alpha$. Let $f'$ be the restriction of $f$ to the arcs in $G_\beta$. Let $f''$ be obtained by taking $f''_{ij} = 0$ if $i \in X_\alpha$ and $j \in X_\alpha$, and $f''_{ij} = f_{ij}$ for all arcs in $G_\beta$ with at least one endpoint not in $X_\alpha$. One can verify that $f'$ is a pecf in $G_\beta$, and $f''$ is a pecf in $G_\gamma$. Clearly $f = f' + f''$. The last stated condition follows directly, as $f$ is a pecf in $G_\alpha$.

Now, suppose we have a pecf $f'$ in $G_\beta$ and a pecf $f''$ in $G_\gamma$ that fulfill the two conditions. Note that when the inflow equals the outflow for a node $i$ in $f'$ and in $f''$, it also does so in $f = f' + f''$. So, the first condition of a pecf holds, and the second condition holds by assumption. □

We now compute the table of the signatures of the pecfs in $G_\alpha$. See Algorithm 9.

Then now establish its correctness.

**Lemma 7.21.** *Algorithm 9 correctly computes the table of the signatures of pecfs in $G_\alpha$ for a* **Join** *node $\alpha$.*

*Proof.* One can verify that if $\sigma$ is the signature of a pecf $f'$ in $G_\beta$, and $\sigma'$ is the signature of a pecf $f''$ in $G_\gamma$, then the signature $\sigma''$ as computed by Algorithm 9 is the signature of $f' + f''$. Correctness follows now directly from Lemma 7.20. □

Finally we complete the proof of Theorem 7.17. Using the procedures for the different types of nodes, we can compute all tables, in post-order. As each table has pseudopolynomial size, and the time per table is polynomial in the number of signatures of the tables of the children for **Join**, **Introduce** and **Forget** nodes, and $\mathcal{O}(1)$ for **Leaf** nodes, this costs pseudopolynomial time.

**Lemma 7.22.** $F$ *packets can be transmitted from* $s$ *to* $t$ *in the network, if and only if the table of the root* $r$ *of the tree decomposition contains a signature* $(q, \varepsilon)$ *for some* $\varepsilon$ *with* $q(s) = F$.

*Proof.* Note that $G_r$ has the same vertices and arcs as $G$. Thus, a pecf in $G$ is a flow in $G$ that fulfills the energy constraint, and vice versa. The value of a flow with signature $(q, \varepsilon)$ equals $q(s)$. Now the lemma follows. □

After all tables have been computed, we can find the value of the optimal flow by inspecting the table of the root node: by Lemma 7.22, this value equals the maximum $F$, such that a signature $(q, \varepsilon)$ for some $\varepsilon$ with $q(s) = F$ belongs to the table of the root of the tree decomposition. With additional bookkeeping, one can also find the corresponding flow. □

## 7.9. Concluding remarks

The Maximum Flow WSNC problem is an interesting and relevant problem, with practical implications in the context of wireless networks. In this chapter, we studied the integer version of the problem, that is, a version where packets cannot be split. We proved $\mathcal{NP}$-hardness and $\mathcal{APX}$-hardness in various settings. Specifically, the problem is strongly $\mathcal{NP}$-complete even on geometric configurations on a line (Corollary 7.8), and $\mathcal{APX}$-hard even if there are only two possible transmission powers (Theorem 7.14). We also studied how the complexity of the problem depends on the treewidth of the network. We found that except for the case where each sensor has one fixed power setting or when the underlying graph is of treewidth two with an edge joining the source and sink nodes, the problem is weakly $\mathcal{NP}$-complete for bounded treewidth greater than two.

# 8 Energy Constrained Flow: Approximation and advantage

In this chapter, we continue our study of the energy-constrained flow problem. The results are mainly based on the fractional relaxation from the previous chapter. First we give an approximation algorithm that finds an integer flow of value $F^*_{\text{frac}} - m$, where $F^*_{\text{frac}}$ is the fractional optimum and $m$ is the number of arcs in the network. Then we develop several variants of a column generation algorithm for energy-constrained flow. We experimentally demonstrate that the best version outperforms a direct linear programming approach for the fractional relaxation. Furthermore, our best column generation algorithm provides a good heuristic for solving the integer program.

Next we consider approximation. Nutov [Nut08] has given a constant-factor approximation algorithm based on bicriteria optimisation. This approach involves the concept of energy advantage: finding a large integer flow that violates the energy constraints by at most a certain factor. Nutov's approximation algorithm works by first giving an algorithm with constant advantage; based on this he achieves a constant-factor approximation algorithm. For $st$-planar graphs, we give an algorithm with advantage 3, which is an improvement over Nutov's algorithm. This also gives an approximation algorithm of improved performance ratio for $st$-planar graphs. This algorithm has a combinatorial interpretation on the embedding of the graph, in contrast to the primarily polytopal proof of the other results. We furthermore show that a natural randomised-rounding version of our column generation algorithm has energy advantage of almost surely $\mathcal{O}(\log n / \log \log n)$. This is worse than Nutov's constant factor, but a bound on the advantage of this natural algorithm is nevertheless interesting, in particular since our randomised-rounding algorithm can be expected to have better runtime in practice.

## 8.1. Introduction

Most of this chapter revolves around the fractional relaxation of the energy-constrained flow problem, in which the constraint that the flow has to be integral is dropped. This version of the problem is polynomial-time solvable, for example by linear programming, and is the basis on which the results in this chapter are built.

First we provide a simple approximation algorithm, which does not have a constant-factor performance ratio. Instead, it has an additive guarantee: it finds an integer flow of value at least the fractional optimum minus the number of arcs in the network. If the fractional optimum is low, this might not guarantee any flow at all. If, on the other hand, the fractional optimum is high, this can be a good guarantee. This is reasonable: when the battery capacities are high, even an integer flow is relatively 'fluid' and not a lot gets lost in our rounding procedure.

In the previous chapter, we used a linear-programming formulation of the problem with a flow variable for every arc in the network; call this the *edge formulation*. Now we construct an equivalent formulation with a flow variable for every simple path from a source to the sink; call this the *path formulation*. We apply column generation to the path formulation and find that a naive implementation of this algorithm does not perform particularly well in practice. However, with an alternative pricing problem, the column generation algorithm outperforms directly solving the edge formulation in the fractional case, and provides a good heuristic in the integer case.

Next we consider the concept of *energy advantage*. These are algorithms that find a good flow for a related problem: the algorithm is allowed to violate the battery capacities by at most a certain factor and returns an integer flow that is at least as big as the optimum solution for the actual battery capacities. In the literature, such results are considered in the context of *bicriteria optimisation*. Results are known, for example, for the minimum-weight bounded-degree spanning tree problem [SL07]: in that setting, 'advantage' means being allowed to violate some degree constraints.

We give two algorithms with energy advantage: one for general networks, and one for $st$-planar networks. For general networks we give a randomised algorithm that achieves $\mathcal{O}(\log n / \log\log n)$ advantage almost surely. This is based on a natural randomised rounding of the path formulation. For $st$-planar networks we give a deterministic algorithm with advantage 3. These algorithm do not directly lead to approximation algorithms of the corresponding factor for the original problem, but are themselves an interesting result. By a result of Nutov [Nut08], these algorithms with advantage do lead to an approximation algorithm. For $st$-planar networks, the approximation factor resulting from our algorithm improves on previous results. For general networks our randomised rounding does not; still, it is of interest for its practical efficiency, as demonstrated by our experiments with the column generation algorithm on which it is based.

Before we begin, we restate the edge formulation of energy constrained flow; the fractional relaxation is obtained by dropping constraint (8.2) and thereby allowing fractional arc flows $f_{ij}$. The total amount of flow into the sink, here denoted by $\mathtt{flow}(f)$, is also called the *value* of the flow.

Program 1: edge formulation

Maximise

$$\mathtt{flow}(f) = \sum_{j \in N} f_{jt} \qquad (8.1)$$

Subject to:

$$f_{ij} \ \mathtt{integer}, \qquad \forall (i,j) \in A \qquad (8.2)$$

$$f_{ij} \geqslant 0, \qquad \forall (i,j) \in A \qquad (8.3)$$

$$\sum_{j \in N} f_{ij} = \sum_{j \in N} f_{ji}, \qquad \forall i \in N - S - \{t\} \qquad (8.4)$$

$$\sum_{j \in N} e_{ij} \cdot f_{ij} \leqslant E_i. \qquad \forall i \in N \qquad (8.5)$$

## 8.2. An additive approximation algorithm

As the Integer Max-Flow WSNC problem has no PTAS (unless $\mathcal{P}$=$\mathcal{NP}$), our hope is to find some approximation algorithms instead. The idea is to first solve the fractional LP-formulation in polynomial time, and then find an integral flow from the solution. For example, Kalpakis et al. [KDN03] use this idea and then apply the standard integral maxflow-mincut flow algorithm [AMO93] to the resulting fractional LP-formulation after the fractional values have been rounded down to integer values. However, they do not provide a bound on the quality of their approximation.

Below we also first solve in polynomial time the fractional LP-formulation and then try to find a large integer flow that is close to the optimum value. We then give a lower bound for the result of the approximation algorithm.

**Theorem 8.1.** *There is a polynomial-time approximation algorithm for the Integer Max-Flow WSNC problem, which computes an integer maximum flow with value* $F_{\text{approx}} \geqslant F_{\text{opt}} - m$, *where* $m$ *is the number of arcs of the network.*

*Proof.* We give the proof for the case when there is only one source. It is a simple exercise to generalise the proof to the case with multiple sources.

First, we solve the relaxation of the problem optimally, that is, we allow flows to be of real value. As this is an LP, the ellipsoid method gives us in polynomial time an optimal solution $F^*$ that can be realised within the energy constraints.

We now find a large integer flow inside $F^*$ in the following way. We use an integer flow function $F$, which invariantly will map each arc $ij$ to a non-negative integer $f_{ij}$ with $f_{ij} \leqslant f^*_{ij}$, and which has conservation of flows; that is, $F$ will invariantly be a flow that fulfills the energy constraints. Initially, set $F$ to be $0$ on all arcs.

Now, repeat the following step while possible. Find a path $P$ from $s$ to $t$, such that for each arc $ij$ on the path, $f^*_{ij} - f_{ij} \geqslant 1$. Let $f_P = \min_{ij \in P} \lfloor f^*_{ij} - f_{ij} \rfloor$ be the minimum over all arcs $ij$ on the path $P$. Note that $f_P \geqslant 1$. Now add $f_P$ to each $f_{ij}$ for all arcs $ij$ on the path $P$. Observe that the updated function $F$ fulfills the energy constraint conditions.

The process ends when each path from $s$ to $t$ contains an arc with $f^*_{ij} - f_{ij} < 1$. We note that $F^* - F$ is a flow, and standard flow techniques show that its value is at most $m$, the number of arcs. (For let $\mathcal{S}$ be the set of nodes, reachable from $s$ by a path with all arcs fulfilling $f^*_{ij} - f_{ij} \geqslant 1$. Since $t \notin \mathcal{S}$, $(\mathcal{S}, V - \mathcal{S})$ is a cut and its capacity with respect to the flow $F^* - F$ is at most the number of arcs across the cut.) Since the value of $F^*$ is at least $F_{opt}$, and hence the value of $F$ is at least $F_{opt} - m$.

In each step of the procedure given above, we obtain at least one new arc $ij$ with $f^*_{ij} - f_{ij} < 1$; this arc will no longer be chosen in a path in a later step. Thus, we perform at most $m$ steps. Each step can be done easily in linear time. Hence, the algorithm is polynomial, using the time of solving one linear program plus $O(m(n+m))$ time for computing the approximate flow. □

## 8.3. Basic column generation

We now develop an alternative LP formulation for the energy-constrained flow problem. The original formulation has a flow variable for every arc of the network; this new formulation has flow variable for every simple path from a source to the sink. This is the *path formulation*, as opposed to the *edge formulation* from before. It has a large set of variables, to which we will apply a column generation approach. In the experimental evaluation, we call this algorithm `CG`.

Let $Q$ be the set of all simple paths from a source to the sink. For every $q \in Q$, we have a decision variable $x_q$. We use the following notation to indicate whether some arc $(i, j)$ lies on path $q$:

$$P(q, i, j) = \begin{cases} 1 & \text{if arc } (i, j) \text{ is on path } q \\ 0 & \text{otherwise} \end{cases} \tag{8.6}$$

Note that these values $P(q, i, j)$ are constant, given the network and the path. We can now write the path-based linear program as follows.

Program 2: path formulation

Maximise

$$F = \sum_{q \in Q} f_q \tag{8.7}$$

Subject to:

$$f_q \ \mathit{integer}, \qquad \forall q \in Q \tag{8.8}$$

$$f_q \geqslant 0, \qquad \forall q \in Q \tag{8.9}$$

$$\sum_{j \in N} \sum_{q \in Q} P(q, i, j) \cdot e_{ij} \cdot f_q \leqslant E_i. \qquad \forall i \in N \tag{8.10}$$

Equations (8.8) and (8.9) carry over immediately from before: the flow should be integral and non-negative. There is no need for a flow-conservation constraint like Equation (8.4) since flow is already tied to variables representing an entire path from a source to the sink: conservation of flow is guaranteed by this choice of variables. This leaves the battery capacity. The energy usage of a node can be calculated as a sum over all paths going through it, where the weight depends on the outgoing arc used by the path. This is done in Equation (8.10) using a sum over all paths, where $P(q, i, j)$ is used to determine whether path $q$ actually goes through node $i$.

In order to apply column generation, we must now consider the fractional relaxation of the above linear program, that is, we drop Equation (8.8). It follows from the theory of linear programming that we have solved this relaxation to optimality if and only if the reduced cost of every possible column is at most zero. The non-negativity of the variables, as enforced by Equation (8.9), is part of the standard form of linear programs. The only remaining constraint is Equation (8.10): let $\pi_i$ be the dual multiplier associated with these constraints.

Let $q'$ be a path not currently included as a column and consider sending 1 flow over it. Looking at Equation (8.10), we see that its reduced cost is

$$1 - \sum_{i \in N} \sum_{j \in N} P(q', i, j) \cdot e_{ij} \cdot 1 \cdot \pi_i \tag{8.11}$$

Since $P(q', i, j)$ is zero whenever the arc $(i, j)$ is not on $q'$, we can recognise that instead of summing over all pairs of nodes, we may as well sum over the arcs of $q'$. For those arcs, $P(q', i, j)$ equals 1 so we drop it.

$$= \ 1 - \sum_{(i,j) \in q'} e_{ij} \cdot \pi_i \tag{8.12}$$

This formula for reduced cost allows us to determine whether the current set of columns is optimal or, if not, which columns could be added to improve the objective value. Instead of enumerating all possible paths and checking them, we directly find the path that maximises this expression.

Notice that Equation (8.12) is a constant minus a weighted sum over the arcs in $q'$ and is therefore maximised by minimising the cost of the path. This is a shortest path computation from any source to the sink, where the arc $(i, j)$ is weighted with $e_{ij} \cdot \pi_i$. If there exists a path $q'$ from a source to the sink with length at most one, then it has positive reduced cost (Equation 8.12) and we can add it to the column pool; we take for $q'$ a shortest path. Solving of the linear program is resumed with this additional variable. If no such path exists, the algorithm terminates with an optimal fractional solution.

Column generation as applied here is not valid for integer programs, so the above approach only solves the fractional relaxation. In order to find optimal integral solutions, it is possible to use a branch-and-price approach, but we have not pursued this. Instead we have implemented the following heuristic, which is commonly applied to column generation algorithms for integer programs.

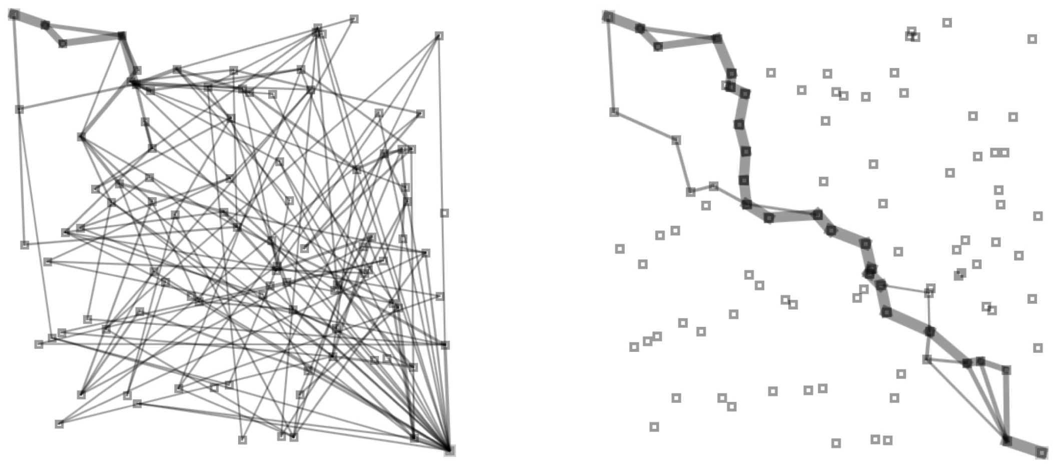
First we use column generation to solve the fractional relaxation to optimality and we save the column pool. Then we reintroduce the integrality constraint and use an ILP solver to get an optimal integer solution given the available columns. Call this the *integer heuristic*. Note that the quality of this heuristic depends on the specific contents of the column pool. Indeed, there is no guarantee that the columns necessary for the actual integer optimum are available in this process: hence the resulting algorithm is heuristic.

Using an implementation of this column generation procedure we see that it performs poorly: finding a fractional optimum is slower than solving the edge formulation directly, and so is finding an integer solution (that is not even optimal). We observe that many paths are found that are, upon visual inspection, clearly bad. This can occur because early during the column generation, not many energy constraints are tight and the pricing problem is, in that case, not particularly informative: there may be many paths that are optimal according to the pricing problem even though they are extremely wasteful in terms of energy expended by nodes whose energy constraint is not currently tight. Indeed, much effort is expended on finding and using columns that an integral optimal solution can 'surely' do without.

In the next two sections we will provide improvements in this area. In order to give a unified presentation of the experimental results, we first describe these improvements; the experiments follow in Section 8.6.

## 8.4. Alternating column generation

Now we consider the fractional relaxation again. There can be much freedom in the routing of the flow, even in an optimal solution. See for example Figure 8.1, where two different flows are visualised. Both have optimal value, but they look very different. Figure (a) shows a solution found by the algorithm from



**(a)** Arbitrary optimal flow.

**(b)** Optimal flow that uses minimum energy.

**Figure 8.1.** Two flows in the same network with geometric transmission costs (see Section 8.6 for details). The network is complete; arcs are not drawn. The grey lines indicate the flow, where thicker lines indicate more flow, and the absence of a line indicates zero flow. Both flows achieve optimal value. Figure (a) is found using CG, the basic column generation algorithm from Section 8.3. Figure (b) is, among all flows of optimal value, the flow with minimum energy usage: it uses approximately 20% as much energy. It is found by CG_A, the column generation algorithm from Section 8.4.

the previous section. Having computed this solution, we know the optimal flow value in this network. Figure (b) shows a flow of equal value, but with minimum energy usage among all flows of this value. We observe that this solution looks much nicer, and in particular uses fewer and more 'reasonable' paths.

Let us now be more specific about this minimum-energy solution. By energy usage we mean the sum over all nodes of the energy spent by a node. Let $b_q$ be the energy consumed by unit flow over path $q$, that is,

$$b_q = \sum_{(i,j)\in q} e_{ij}. \tag{8.13}$$

Then the total energy usage of a flow $f$ is simply a weighted sum over all paths.

$$\mathit{energy}(f) = \sum_{q\in Q} b_q \cdot f_q \tag{8.14}$$

Consider the following optimisation problem: minimise energy usage, subject to the constraint that the amount of flow is maximum. This will help us find better columns to use for the integer heuristic.

Program 3: minimum energy

Minimise

$$\mathrm{energy}(f) = \sum_{q\in Q} b_q \cdot f_q \tag{8.15}$$

Subject to:

$$\sum_{q\in Q} f_q \text{ is maximum} \tag{8.16}$$

$$f_q \text{ integer}, \qquad \forall q \in Q \tag{8.17}$$

$$f_q \geqslant 0, \qquad \forall q \in Q \tag{8.18}$$

$$\sum_{j\in N}\sum_{q\in Q} P(q,i,j)\cdot e_{ij}\cdot f_q \leqslant E_i. \qquad \forall i \in N \tag{8.19}$$

This problem can be solved in two steps. First calculate a flow of maximum value without regard for energy, for example using the edge formulation or the column generation from the previous section. Then replace Equation 8.16 with

$$\sum_{q\in Q} f_q \geqslant F_{opt} \tag{8.20}$$

where $F_{opt}$ is the optimal flow value calculated without regard for energy consumption.

Let $\lambda$ be the dual multiplier associated with Equation 8.20, and $\pi_i$ as before (now corresponding to Equation 8.19). Taking this new constraint into account, the reduced cost of a column $q$ is

$$\sum_{(i,j)\in q} e_{ij} - \left(\sum_{(i,j)\in q} e_{ij}\cdot\pi_i\right) - \lambda. \tag{8.21}$$

This reduced cost is positive if

$$\sum_{(i,j)\in q} e_{ij} - e_{ij}\cdot\pi_i < \lambda. \tag{8.22}$$

The resulting pricing problem is again a shortest path computation: now the arc weights are $e_{ij} - e_{ij}\cdot\pi_i$ and we are looking for a path of length at most $\lambda$.

When we apply column generation to this energy-minimisation problem, we end up with a different column pool than before. This column pool leads to better solutions when used as a heuristic for the integer problem. (Experimental results follow in Section 8.6.) However, the algorithm starts by solving the original max flow problem (in order to find $F_{opt}$), so this is not an improvement in terms of runtime. This can be improved by interleaving the new column generation procedure with the one from the previous section as follows.

Note that the maximum flow and the minimum energy problem use the same set of variables. Call the two pricing problems `max-flow` and `min-energy`. Start by adding a `max-flow` column. This achieves some flow $F_1$. Then, keeping the same column pool, solve `min-energy` to optimality, with $F_1$ as the lowerbound on the flow. This is possible since the column pool (and indeed, the basis) contains a set of columns that feasibly achieves a flow of value $F_1$. Repeat the following process:

1. add a column using `max-flow`, improving the flow value to $F_i$, and then
2. solve `min-energy` to optimality with $F_i$ as the lowerbound on flow.

The algorithm terminates when `max-flow` finds no improving column. Then we have found a flow of optimal value, since otherwise `max-flow` would find an improving column. Among flows of this value, we achieve minimum energy usage, since otherwise `min-energy` would find an improving column.

This alternating (or: interleaved) column generation procedure if called `CG_A` in the experiments. It performs much better than `CG`, the naive column generation procedure from the previous section. It solves the fractional relaxation much faster, using a smaller column pool: the energy minimisation along the way seems to steer the process to more reasonable columns sooner. Besides being calculated more quickly, this column pool also exhibits higher quality when used in the integer heuristic. Details of these experimental results follow in Section 8.6.

## 8.5. Back to one objective

In this section we look at a third column generation procedure. It is faster still, but does not actually find a fractional solution with optimum flow value: we change the objective function to be a weighted combination of flow maximisation and energy minimisation. Let $f$ be a flow and denote its value by $\mathrm{flow}(f)$, that is,

$$\mathrm{flow}(f) = \sum_{q \in Q} f_q \tag{8.23}$$

We want a solution with high flow, but would like to do so with little energy. We therefore introduce the following objective function, to replace the objective function in Program 2 (Equation 8.7). Let $W$ be some large constant.

$$\textbf{Maximise } \mathrm{combined}(W, f) = W \cdot \mathrm{flow}(f) - \mathrm{energy}(f) \tag{8.24}$$

We want to pick $W$ such that the energy minimisation doesn't interfere (too much) with the flow maximisation. First we derive a reasonable value for $W$ and then we do column generation for this new objective.

Consider two flows $f$ and $f'$ and let $d = f' - f$, their component-wise difference. Then $\mathrm{flow}(d)$ is the difference in flow value between $f$ and $f'$. Meanwhile, the difference in objective value is $\mathtt{combined}(W, d)$. To find the actual maximum flow using $\mathtt{combined}$, we would need to pick $W$ such that

$$\mathtt{flow}(d) > 0 \implies \mathtt{combined}(W, d) > 0.$$

Unsurprisingly no $W$ exists that guarantees this. For an approximation guarantee we set $W$ such that the following condition is satisfied: if the flow in a solution $f'$ is at least $\varepsilon$ larger than in $f$, then its combined objective value is strictly higher.

**Definition 8.2 ($\varepsilon$-Safety of $W$).** A value of $W$ is called $\varepsilon$-safe if, for any flows $f$ and $f'$

$$\mathtt{flow}(f' - f) > \varepsilon \implies \mathtt{combined}(W, f') > \mathtt{combined}(W, f)$$

**Lemma 8.3.** *Let $W$ be $\varepsilon$-safe. Let $f^*_{\mathtt{comb}}$ be a flow that is optimal for $\mathtt{combined}(W, \cdot)$ and let $f$ be any flow. Then $\mathtt{flow}(f^*_{\mathtt{comb}}) \geqslant \mathtt{flow}(f) - \varepsilon$.*

*Proof.* Suppose to the contrary that

$$\mathtt{flow}(f^*_{\mathtt{comb}}) < \mathtt{flow}(f) - \varepsilon.$$

Then by linearity of $\mathtt{flow}(\cdot)$, and rearranging we have

$$\mathtt{flow}(f - f^*_{\mathtt{comb}}) > \varepsilon.$$

By $\varepsilon$-safety of $W$ this implies that

$$\mathtt{combined}(W, f) > \mathtt{combined}(W, f^*_{\mathtt{comb}}).$$

This contradicts the optimality of $f^*_{\mathtt{comb}}$. □

We will now derive a lowerbound on $W$ that guarantees $\varepsilon$-safety. At first we use the unknown optimum flow value $F_{\mathrm{opt}}$, but we can upperbound this by the amount of flow that can possibly leave the sources: battery capacity divided by the cost of the cheapest outgoing arc.

**Lemma 8.4.** *Let $F_{opt}$ be the maximum flow value in the network and $b_{max} = \max_{q \in Q} b_q$. If $W \geqslant \frac{b_{max} F_{opt}}{\varepsilon}$, then $W$ is $\varepsilon$-safe.*

*Proof.* Let $W \geqslant \frac{b_{\max} F_{\mathrm{opt}}}{\varepsilon}$. Then for any flow $f$,

$$\mathtt{combined}(W, f) \geqslant \mathtt{combined}(\frac{b_{\max} F_{\mathrm{opt}}}{\varepsilon}, f). \tag{8.25}$$

Let $f$ and $f'$ be flows, and let $d = f' - f$ be their difference. Then the definition of $\varepsilon$-safety is

$$\mathtt{flow}(d) > \varepsilon \implies \mathtt{combined}(W, d) > 0. \tag{8.26}$$

Expanding the definition further and plugging in our bound for $W$ gives a sufficient condition for $\varepsilon$-safety:

$$\sum_{q\in Q} d_q > \varepsilon \implies \frac{b_{\max}F_{\text{opt}}}{\varepsilon} \cdot \left(\sum_{q\in Q} d_q\right) - \sum_{q\in Q} b_q d_q > 0. \tag{8.27}$$

Since $d$ is the difference between two flows, it can have positive as well as negative elements. Define $d^+$ as follows: $d^+_q = \max\{0, d_q\}$; similarly $d^-_p = \min\{0, d_p\}$. That is, $d^+$ has the positive elements of $d$ and zero elsewhere, and $d^-$ has the negative elements of $d$ and zero elsewhere. Then $d = d^+ + d^-$, which we plug into Equation (8.27).

$$\sum_{q\in Q} d_q > \varepsilon \implies \frac{b_{\max}F_{\text{opt}}}{\varepsilon} \cdot \left(\sum_{q\in Q} d_q\right) - \left(\sum_{q\in Q} b_q d^+_q + \sum_{q\in Q} b_q d^-_q\right) > 0 \tag{8.28}$$

To guarantee that the implication holds, we now bound the values $\sum_{q\in Q} b_q d^+_q$ and $\sum_{q\in Q} b_q d^-_q$. Note that all elements of $b$ are positive (they are sums of energy costs) and all elements of $d^-$ are non-positive. Therefore

$$\sum_{q\in Q} b_q d^-_q \leqslant 0. \tag{8.29}$$

Regarding $\sum_{q\in Q} d^+_q$, we have the very crude bound that it is at most the maximum flow $F_{\text{opt}}$ in the network. Combining this with $b_i \leqslant b_{\max}$ gives

$$\sum_{q\in Q} b_q d^+_q \leqslant b_{\max}F_{\text{opt}} \tag{8.30}$$

and, combined with Equation (8.29), gives

$$\sum_{q\in Q} b_q d_q \ < \ b_{\max}F_{\text{opt}}. \tag{8.31}$$

Plugging this back into Equation (8.27), we see that $\varepsilon$-safety holds:

$$\sum_{q\in Q} d_q > \varepsilon \implies \frac{b_{\max}F_{\text{opt}}}{\varepsilon} \cdot \left(\sum_{q\in Q} d_q\right) - b_{\max}F_{\text{opt}} > 0. \tag{8.32}$$

□

**Theorem 8.5.** *Let $W > \frac{b_{max}F_{opt}}{\varepsilon}$ and let $f^*$ maximise* $\mathtt{combined}(W, \cdot)$*. Then $f^*$ is an additive $\varepsilon$-approximation for maximising* $\mathtt{flow}(\cdot)$.

Having established a good bound for $W$, we can now do the column generation. The resulting algorithm is called `CG_BAT` in the experiments.

The linear program is again Program 2, except for the objective function: now we have Equation (8.24). We still have dual multipliers $\pi_i$ for the energy constraints. With the new objective function, the reduced cost of column $q$ is now

$$W - \sum_{q \in Q} e_{ij} - \sum_{q \in Q} e_{ij}\pi_i. \tag{8.33}$$

In a maximisation problem, an improving column has positive reduced cost. Again, the pricing problem is a shortest path problem: we are looking for a path $q$ of length at most $W$ in the network where the edges are weighted by $e_{ij} + e_{ij}\pi_i$.

## 8.6. Experimental evaluation

In the preceding sections we have seen three column generation algorithms.

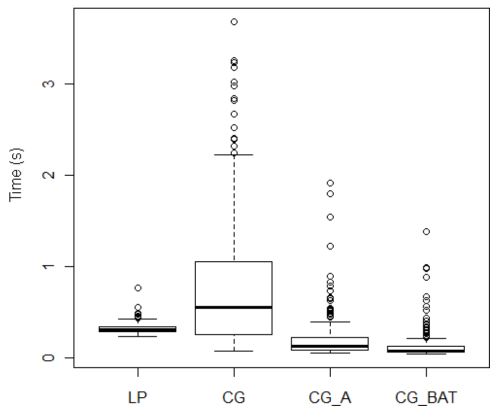
- A straightforward approach: call this `CG`.
- An approach that alternates between two pricing problems: call this `CG_A`.
- An approach with an objective function combining flow value and battery usage: call this `CG_BAT`.

As a baseline, we also compare these algorithms to the following.

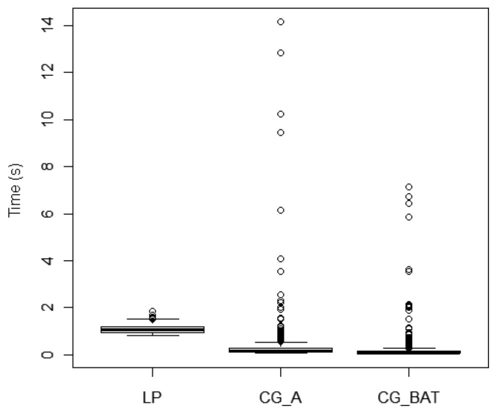
- The edge formulation, that is, a linear program with a flow variable per edge (Program 1): call this `LP`.

In this section we report the results of our computational experiments. The data were obtained using an implementation in C++. The implementation is single-threaded and the experiments were run on an Intel® Core™ 2 Duo processor at 2.4 GHz with 4GB of RAM. Solving linear programs, both fractional and integer, was done using the 32-bit version of CPLEX 11.0.1

**Input** As input for the experiments, we have generated random networks in the plane as follows; the number of networks used is indicated for each experiment (usually at least 100). Place a single source $s$ and the sink $t$ in opposing corners of a square region and place $n - 2$ relay nodes in this same region. We pick the locations of the relay nodes using a uniform binomial point process in this rectangle. As network we take the complete bidirected network on this node set: any node can transmit to any other node. The transmission cost $e_{ij}$ of arc $(i, j)$ is set to $\mathtt{dist}(i, j)^2$, where $\mathtt{dist}$ is the Euclidean distance. As seen several times before in this thesis, this cost is a typical model for the energy cost associated with a wireless transmission (see Section 2.6 for a discussion). The instance in Figure 8.1 was generated using this process.



**(a)** $n = 100$ (200 runs)

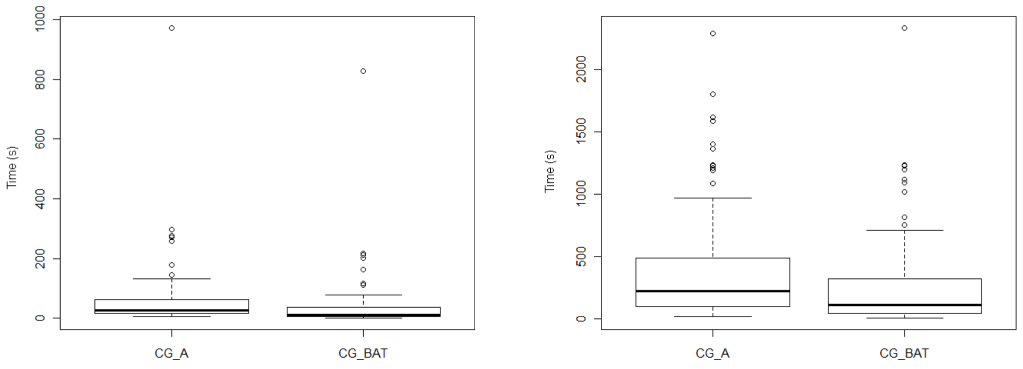


**(b)** $n = 200$ (500 runs)

**Figure 8.2.** Box plots of runtime for finding an optimal fractional solution. Basic column generation (CG) performs poorly, even at $n = 100$. The other two column generation algorithms perform better than LP. The difference is small at $n = 100$, but becomes more noticeable at $n = 200$. Do note that CG_BAT does not have the exact same objective function as the others.

**Fractional relaxation** First of all we consider the runtime of finding a fractional optimum. In Figure 8.2 we see that the basic column generation (CG) is not an improvement of the edge formulation (LP). In fact, it has the worst performance of all options, by a wide margin. The two improved column generation algorithms (CG_A and CG_BAT) are faster than the edge formulation. For $n = 100$ the absolute difference in runtime is quite small, but at $n = 200$ we see a pronounced difference in runtime. It should be noted, however, that the column generation algorithms have some outliers with high runtime, and that CG_BAT does not have the exact same objective function as the others.

We now look at much larger instances, specifically $n = 5000$ and $n = 10000$. For networks of this size, memory usage becomes an issue. Since we test on complete bidirected networks, the edge formulation involves giving the LP solver a dense matrix of energy costs $e_{ij}$. This matrix itself may not be prohibitively large for these values of $n$ (approximately 100 and 400 megabyte respectively, assuming double precision floats), but including overhead and working space, CPLEX runs out of memory. Even though we treat the LP solver as a black box, it is entirely reasonable that we do our own implementation for the pricing problems. By working directly with the node positions, there is no need for an explicit matrix of energy costs: the shortest-path computations can evaluate any $e_{ij}$ whenever needed. In this way, the memory usage of the column generation algorithms is not necessarily $\Omega(n^2)$. On the other hand, memory usage scales with the number of columns found, for which we have no reasonable worst-case bound; we shall discuss shortly that this does not pose a problem in practice.



**(a)** $n = 5000$ (50 runs) **(b)** $n = 10000$ (135 runs)

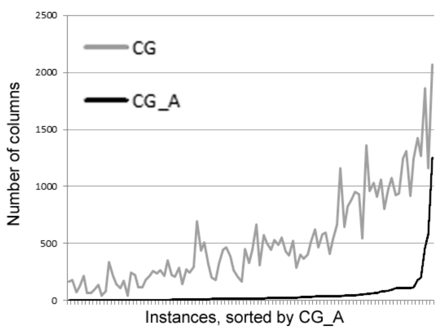
**Figure 8.3.** Box plots of runtime for finding an optimal fractional solution on large instances. For networks of this size, the basic `LP` cannot be used: CPLEX runs out of memory. We see that `CG_BAT` performs better than `CG_A`. The basic version (`CG`) performs much worse than either and it not included in these diagrams.

In Figure 8.3 we compare the runtime of `CG_A` and `CG_BAT` on large instances, and see that the latter is faster. We don't include results for `LP` because it fails for lack of memory; we don't include results for `CG` because it is prohibitively slow on instances of this size. We conclude that the column generation approach is successful for the fractional relaxation: it is able to solve instances that are too large for the edge formulation and, when the improved algorithms are used, it is faster.

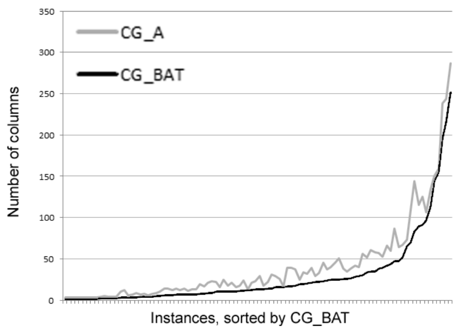
The efficacy of the improved algorithms (`CG_A` and `CG_BAT`) can be confirmed by looking at the size of the column pool once optimality is reached, that is, the number of columns generated over the course of the algorithm. Figure 8.4 shows the same relation between the algorithms as we saw for runtime: `CG` is the worst, by a large margin, and `CG_BAT` is the best, by a small margin. This corresponds to our observation that `CG` tends to generate many paths that are, in a sense, unnecessary. Note in particular that, with the exception of `CG`, the distribution of column-pool size is heavily biased to the smaller sizes, with some large outliers. In the specific case of $n = 500$, for example, we see that fewer than 50 columns suffice to find an optimal solution in approximately 90% of instances. This explains why memory usage is better than in the edge formulation: the column pool is typically much smaller than $n$.

**Integer heuristic** The column generation algorithms can also be used as a heuristic for finding an integer flow. To do this, we first find an optimal fractional solution; then we reintroduce the integrality constraint and use an ILP solver to find the integer optimum using the column pool arrived at in the first step.

We no longer explicitly present runtime results for `CG`, since it is thoroughly



**(a)** $n = 200$ (100 runs)



**(b)** $n = 500$ (100 runs)

**Figure 8.4.** Comparison of the number of columns generated by the three algorithms during their execution. In (a) we see that `CG_A` terminates with a much smaller column pool than `CG` does. This corresponds to our observation that the basic algorithm seems to finds many unreasonable columns and it explains the improved runtime of `CG_A`. In (b) we see that `CG_BAT` consistently generates fewer columns still, but not by a wide margin.

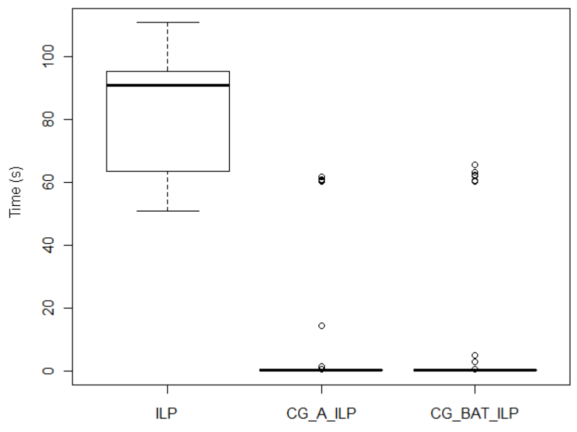
outclassed by the other algorithms: we compare the following three.

- The solution to the edge formulation with integrality constraint, as found by CPLEX within a time limit of two minutes: call this `ILP`. This time limit was enforced to make the experiments feasible in the presence of extreme outliers in runtime, but does means that on some of the larger instances, `ILP` may not actually be optimal.
- The integer heuristic based on `CG_A`: call this `CG_A_ILP`.
- The integer heuristic based on `CG_BAT`: call this `CG_BAT_ILP`.

We see in Figure 8.5 that the heuristics are indeed faster than finding an optimal solution using `ILP`. Looking at $n = 500$, we see that finding an optimal solution typically takes more than a minute, whereas the heuristics typically run in at most a few seconds. The heuristics do have some outliers at slightly over 60 seconds: in these cases, CPLEX was unable to prove optimality of its candidate solution (for the particular column pool) and stopped on a 60 second time-out. This time-out should be configured much lower in practice, since we see in these experiments that it is rarely encountered and the algorithm is heuristic anyway.

The good runtime of the integer heuristics can be explained by the column generation: as we saw in Figure 8.4, the column pool typically contains far fewer than $n$ columns, so the number of integer variables can be surprisingly low.

The contents of the column pool also influence the quality of the solution. In Figure 8.6 we compare the flow value found by the various algorithms. We see at $n = 200$ that the flow found by `ILP` and `CG_A_ILP` are equal in almost all



**(a)** $n = 500$ (100 runs)

**Figure 8.5.** Box plots of runtime for finding an integer solution: ILP is optimal and the other two algorithms are heuristic. We observe that, as the instances become larger, computing the optimal solution using the basic ILP quickly becomes infeasible. The heuristics remain fast even for larger instances.

instances; since ILP did in fact find the optimum in most of those instances, we know CG_A_ILP was also optimal in those cases. The basic column generation algorithm (CG_ILP) consistently gives worse solutions. Next we consider $n = 500$. At this size, CPLEX has trouble proving optimality of its solutions for ILP; indeed, there are instances where no solution is returned within two minutes, which we report as 0 flow. This seems to occur mostly when the heuristics find a large flow. We see that CG_A_ILP and CG_BAT_ILP almost always find flow of identical value and that both are at least as good at ILP, even though they run in several seconds instead of two minutes.

We conclude that the column pools found by the improved column generation algorithms are very reasonable for constructing integer flows. We have seen in particular that CG_A_ILP and CG_BAT_ILP run much faster than ILP, while finding solutions that are often optimal.

## 8.7. Energy advantage for $st$-planar networks

In this section we will show that the energy-constrained flow problem in $st$-planar networks can be solved with *energy advantage* 3. We will explain what that means shortly, but first we note that $st$-planar networks can be considered a reasonable class of networks even though planarity does not seem like a particularly meaningful property of wireless networks. Indeed, in much of this chapter, we have considered complete networks, which are not planar for $n \geqslant 5$. However, every graph with Euclidean distances has a planar spanner with small stretch (or: *dilation*, *spanning ratio*), that is, a planar subnetwork in which dis-

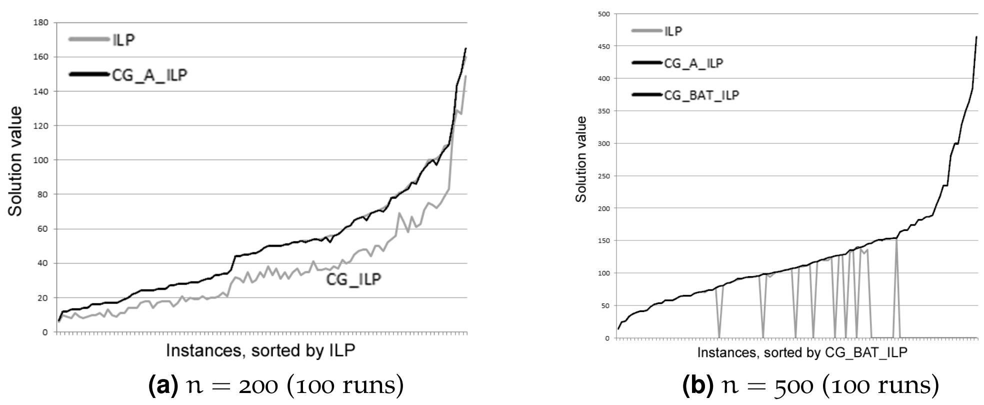


**Figure 8.6.** Comparison of solution value: ILP versus the heuristic solution found by the column generation algorithms. In (a) we see that CG_A_ILP almost always finds an optimal solution, while CG_ILP performs noticeably worse. In (b) we look at larger instances and see that, a lot of the time, both CG_A_ILP and CG_BAT_ILP give solutions almost equal to ILP. However, the latter sometimes reports 0, which indicates that CPLEX failed to find a feasible solution within the time-out of 120 seconds. This seems to occur mostly when the heuristics find a large flow.

tances are increased by at most a small constant factor. Routing in this subnetwork instead of the full network has limited overhead in terms of energy costs. The Delaunay triangulation of the relay-node positions (along with $s$ and $t$), for example, is planar and has stretch less than 1.998 [Xia13]. If $s$ and $t$ additionally lie on the triangulation's outer face (as they did in the experiments of Section 8.6, for example), then the network is also $st$-planar.

Now we consider energy advantage. Consider an instance and its fractional relaxation: let the optimum be $F^*_{\text{frac}}$. The algorithm will find an integer flow of at least this value, which we can of course not expect to be feasible. We will show, however, that each of the battery capacities is violated by at most a factor 3. In this sense, the flow is a constant factor away from being feasible.

We use the following notation. For a graph $G = (V, E)$ and a flow $F : E \to \mathbb{R}$, we write $G[F]$ for subgraph of $G$ induced by the edges over which there is non-zero flow in $F$. We now describe the Decompose/Reroute algorithm. It will give a flow that is 'at least optimal,' given an energy 3-advantage. The algorithm comprises the following steps.

1. Solve the fractional relaxation: get flow $F^*_{\text{frac}}$.
2. Greedily pick integer flow $F_{\text{greedy}}$ within $F^*_{\text{frac}}$.
3. Some flow $F_{\text{small}} = F^*_{\text{frac}} - F_{\text{greedy}}$ remains. Decompose $F_{\text{small}}$ into an ordered set of paths.

4. Merge/reroute paths in $F_{\text{small}}$ into an integer flow $F_{\text{merge}}$.

In the remainder of this section, we will explain the steps in more detail. We will show that the resulting integer flow $F_{\text{greedy}} + F_{\text{merge}}$ has value at least $\lfloor F^*_{\text{frac}} \rfloor$ while exceeding energy capacity by at most a factor 3. Let $G = (V, E)$ be given. We now present the details of the algorithm. Each subsection deals with a step in the informal outline.

### 8.7.1. Solve fractional relaxation

After some preprocessing on the graph, we will first find an optimal fractional flow $F^*_{\text{frac}}$ by linear programming.

As a preprocessing step, we remove all edges with $e_{ij} > E_i$: the cost of sending one unit of flow over an arc with this property exceeds the battery capacity of the transmitting node and therefore this arc cannot be used at all in an integer solution. It is therefore safe to remove these arcs. By removing such arcs from the instance we make sure the fractional relaxation will not use these edges.

Note that $F^*_{\text{frac}}$ is an upperbound on the integer optimum $F^*$.

### 8.7.2. Greedily pick integer flow

Next we will greedily find integer paths of flow within $F^*_{\text{frac}}$, like in Theorem 8.1, and set this flow aside as $F_{\text{greedy}}$. When these is no more path that admits an integer flow of value at least 1, then $F_{\text{greedy}} \geqslant F^*_{\text{frac}} - m$: this follows from the approximation algorithm from Section 8.2. Call the remaining flow $F_{\text{small}}$. Alternatively, if we have used the column generation algorithm to find $F^*_{\text{frac}}$, we already have a decomposition into paths. For every path $q$ with positive flow, the integer part is set aside in $F_{\text{greedy}}$ and all fractional parts remain as $F_{\text{small}}$.

**Proposition 8.6.** $F_{greedy} \geqslant F^*_{frac} - m$

### 8.7.3. Decompose remaining flow into ordered set of paths

We now decompose $G[F_{\text{small}}]$ into an ordered set of paths $P = (p_1, \ldots, p_{|P|})$. Then we can represent $F_{small}$ in terms of these paths: a flow amount per path instead of per edge. That is, we will construct a flow $F_{paths} : P \to [0, 1)$. Note that by construction of $F_{\text{small}}$, every path carries strictly less that 1 flow.

If we have used the column generation algorithm to finds $F_{\text{small}}$, we will now forget about the paths used by the column generation: we find the set of paths $P$ as follows, by looking just at the total flow through edges. It is in this step that we will use the $st$-planarity of $G$.

In order to determine $P$, we first find a drawing $\Gamma$ of $G[F_{\text{small}}]$ in which $s$ is the bottommost vertex and $t$ is the topmost vertex. This is possible since $G$ itself is $st$-planar and therefore has a drawing with both $s$ and $t$ on the outer face.

Let $p$ be the path from $s$ to $t$ that is leftmost in $\Gamma$. We now move as much flow as possible from $F_{\text{small}}$ to $F_{\text{paths}}(p)$. That is, let $F_{\text{paths}}(p) = \min_{e \in p} F_{\text{small}}(e)$ and subtract this flow from $F_{\text{small}}$. In this way we 'peel off' the flow $F_{\text{small}}$ into the set of paths $P$.

**Proposition 8.7.** *All paths in $P$ are distinct and $|P| \leqslant m$.*

*Proof.* In every step, at least one arc on the leftmost path is removed from $G[F_{\text{small}}]$. This changes the leftmost path. There are only $m$ arcs. □

**Proposition 8.8.** *For all $p \in P$, $F_{paths}(p) < 1$.*

*Proof.* If $F_{\text{paths}}(p) \geqslant 1$, then $G[F_{\text{small}}]$ contained a path with at least one flow. However, such paths were handled in the greedy step and this flow would already have been moved to $F_{\text{greedy}}$. Contradiction. □

**Proposition 8.9.** *The flow in $F_{small}$ is strictly less than $m$.*

*Proof.* All flow from $F_{\text{small}}$ ends up in $F_{\text{paths}}$. There are at most $m$ paths, each with strictly less than 1 flow. (We already knew this by the flow/cut argument from Proposition 8.6.) □

Denote by $V[p]$ the set of vertices on path $p$. The following observation follows directly from the construction so far.

**Proposition 8.10.** *Let $i < j$. Any vertices in $V[p_i] - V[p_j]$ are strictly to the left of $p_j$. Any vertices in $V[p_j] - V[p_i]$ are strictly to the right of $p_i$.*

We group the paths in $P$ into *bunches* $B_1, \ldots, B_k$ as follows, where $k$ results from the process. Greedily proceed from left to right and group paths such that the sum of the flow over the paths in a group is exactly 1. We take all flow in a path if the sum in the current bunch remains below 1. Otherwise we take as much flow to make the flow in this bunch equal to 1 and leave the remaining flow for the next bunch. This step is possible until in the end some paths remain (on the right side) that have total flow less than 1. We ignore this remaining flow. Note that $k = \lfloor F_{\text{small}} \rfloor$.

**Proposition 8.11.** *Let $i < j$. Then for all $p_a \in B_i$ and $p_b \in B_j$ we have $a \leqslant b$.*

**Proposition 8.12.** *For all vertices $v$, the bunches in which $v$ occurs are consecutive. That is, there exist indices $\ell(v)$ and $r(v)$ such that $v \in B_i$ if and only if $\ell(v) \leqslant i \leqslant r(v)$.*

### 8.7.4. Merge/reroute paths

We will now route 1 unit of integer flow through each of the bunches $B_1 \ldots B_k$ in a way that exceeds energy constraints by a factor at most 3. This will give an integer flow of value $k = \lfloor F_{small} \rfloor$ with energy advantage 3.

Denote by $V[B]$ the set of vertices in a bunch $B$, that is $V[B] = \cup_{p \in B} V[p]$. Denote by $G[B]$ the graph induced by the vertices of $B$, that is $G[B] = G[V[B]]$.

**Definition 8.13.** Let $B$ be a bunch, $p_\ell = \mathtt{left}(B)$ be its leftmost path and $p_r = \mathtt{right}(B)$ be its rightmost path. The vertices $V[B]$ are classified as one of the following, according to their membership of $V[p_\ell]$ and $V[p_r]$:

- $v$ in $V[p_\ell]$ but not in $V[p_r]$: Left
- $v$ in $V[p_r]$ but not in $V[p_\ell]$: Right
- $v$ in neither $V[p_\ell]$ nor $V[p_r]$ Middle
- $v$ in both $V[p_\ell]$ and $V[p_r]$: Articulation.

The reason for the last name is that a vertex classified as Articulation is an articulation point in $G[B]$.

**Proposition 8.14.** *For all $i < k$, the overlap between adjacent bunches is characterised as follow:*

$$V[B_i] \cap V[B_{i+1}] = \mathtt{right}(B_i) \cap \mathtt{left}(B_{i+1}).$$

**Lemma 8.15.** *Consider a vertex $v \in V$ and the varying classifications it has in the bunches $B_1 \dots B_k$.*

- *A vertex is* Middle *in at most one bunch. If it is, it is never* Left *and never* Right.
- *A vertex is* Left *in at most one bunch.*
- *A vertex is* Right *in at most one bunch.*

*Proof.*

- Let $B_i$ be a bunch in which vertex $v$ is Middle. Then $v$ is in no other bunch: by definition $v$ is neither on $\mathtt{left}(B_i)$ nor on $\mathtt{right}(B_i)$ and therefore $v$ does not occur in any other bunches.
- Let $B_i$ be a bunch in which vertex $v$ is Left. Then $v \notin B_{i+1}$ since $v \notin \mathtt{right}(B_i)$. Possibly $v \notin B_{i-1}$, in which case we are done. Otherwise $v \in B_{i-1}$. Then, since $v \in B_i$ we have $v \in \mathtt{right}(B_{i-1})$ and $v$ is not Left there. By induction, the same holds for further $j < i$: if $v \in B_j$ then $v \in \mathtt{right}(B_j)$.
- The case for Right is symmetric to Left. □

Note that a vertex $v$ can be Articulation multiple times.

We will now route 1 integer flow through each bunch $B_i$ using an extremely simple rule. We send the integral unit of flow from $s$ to $t$ through $G[B_i]$, over any simple path with only the following constraint. Any vertex on the path that is of type Articulation in $B_i$ must send this unit of flow over the cheapest outgoing arc that is available in $G[B_i]$.

**Lemma 8.16.** *Such a path through* $G[B_i]$ *exists.*

*Proof.* The network $G[B_i]$ certainly contains an $s$-$t$-path (since by construction the network is a union of $s$-$t$-paths), but an arbitrary path may not satisfy the routing constraint. Consider the biconnected components of $G[B_i]$: they are separated by articulation points. By definition of the types of vertices, each articulation point has type ARTICULATION in $B_i$. We do the following for every articulation point $v$. For the purpose of simple paths, the outgoing arcs of $v$ can be classified as *toward* $s$ or *toward* $t$: let $A_t(v)$ be the outgoing arcs at $v$, going toward $t$. Then we delete all arcs in $A_t(v)$ except for the cheapest among them. Let $G'[B_i]$ be the result of doing this to every vertex of type ARTICULATION in $G[B_i]$. The network $G'[B_i]$ contains an $s$-$t$-path: the above procedure does not delete any bridges. □

Call the resulting flow $F_{\text{merge}}$. We will now argue that $F_{\text{merge}}$ together with $F_{\text{greedy}}$ exceeds battery capacity by only a factor of 3.

**Lemma 8.17.** *Let* $v$ *be a vertex of type* ARTICULATION *in bunch* $B_i$. *Considering the 1 flow through* $B_i$, $v$ *uses less energy in* $F_{\textit{merge}}$ *than it does in* $F_{\textit{small}}$.

*Proof.* In both $F_{\text{merge}}$ and $F_{\text{small}}$ there is a flow of value exactly 1 through $v$. In $F_{\text{small}}$ this flow uses arbitrary outgoing arcs in $B_i$. In $F_{\text{merge}}$ it uses $v$'s cheapest outgoing arc in $B_i$. □

**Lemma 8.18.** *Let* $v$ *be a vertex of type* LEFT, RIGHT *or* MIDDLE. *Considering the 1 flow through* $B_i$, $v$ *uses at most* $E_v$ *energy.*

*Proof.* At the beginning of the algorithm we have removed any arcs with cost $e_{vw} > E_v$. Since we find a simple path through each bunch, the 1 unit of flow through $B_i$ uses only one outgoing arc of $v$ and therefore spends at most $E_v$ energy on this flow. □

**Theorem 8.19.** *Algorithm* DECOMPOSE/REROUTE *finds, in polynomial time, an integer flow of value at least* $\lfloor F^*_{\text{frac}} \rfloor$ *that exceeds energy capacity by a factor at most 3.*

*Proof.* We observe that any vertex $v$ spends at most $3 \cdot E_v$ energy. It spends energy on the following things.

- Flow in $F_{\text{greedy}}$ and flow through bunches where $v$ is of type ARTICULATION. This energy is at most the energy spent in $F^*_{\text{frac}}$ and therefore at most $E_v$ (Lemma 8.17).
- Flow in bunches where $v$ is type LEFT, RIGHT or MIDDLE. In each of these cases the vertex spends at most $E_v$ energy (Lemma 8.18). This happens at most twice (Lemma 8.15).

This totals at most $3 \cdot E_v$ energy per vertex. □

We have now shown that the energy-constrained flow problem admits an algorithm with energy advantage 3 on $st$-planar networks. However, this does not trivially give an approximation algorithm.

**Corollary 8.20.** *There is a 17-approximation for energy-constrained flow on $st$-planar networks. For high flow values, the approximation ratio approaches 9.*

*Proof.* This follows from a lemma of Nutov [Nut08, Lemma 2.3], given that we have an algorithm with advantage 3: the lemma shows, specifically for the energy-constrained flow problem, how to convert an algorithm with advantage into an approximation algorithm. The result follows from substituting our algorithm for $st$-planar networks in place of Nutov's Corollary 2.2. □

## 8.8. Energy advantage for general networks

We will now give a randomised rounding algorithm for general networks that almost surely has advantage $\mathcal{O}(\log n / \log\log n)$.

The algorithm starts off the same as for $st$-planar networks: solve the fractional relaxation and decompose the flow into at most $m$ paths $p_1, \dots, p_m$ with flow amounts $f_1, \dots, f_m$. This representation of the flow will already be available if we have used column generation to find the fractional optimum. In this way, the following algorithm can be seen as randomised rounding of the path formulation.

If there is any path $p_i$ with $f_i \geqslant 1$, we can remove the integer part of the flow, set it aside, and update the battery capacities involved; at the end of the algorithm we add it back to our solution. This guarantees that in the following we have that $f_i < 1$ for all $i$. We then round all variables as follows, calling the result $F_i$: we set $F_i$ to 1 with probability $f_i$ and to 0 otherwise. Note that this gives, on expectation, the correct flow value.

Now we show that this almost surely violates the energy constraints by a factor $\mathcal{O}(\log n / \log\log n)$. Consider a node $v$. Let $t$ the largest integer such that $2^t \leqslant E_v$. For a vertex $v$ and an integer $r$, we define a *heavy class* $H_v^r \subseteq \{0, ..., m\}$ as a set of path indices $i$ such that the corresponding paths $p_i$ satisfy the following three conditions.

- The paths contain $v$, that is, $v \in p_i$ for each $i$ in the set.
- The arc used to leave $v$ has a certain cost: with $(v, u) \in p_i$ this outgoing arc must have $2^r \leqslant e_{vu} < 2^{r+1}$.
- The flow is at least one: $\sum_{i \in H_v^r} f_i \geqslant 1$.

Similarly we define a *light class* $L_v^r \subseteq \{0, ..., m\}$ as a set of indices where the third condition is instead:

- The flow is less than one: $\sum_{i \in L_v^r} f_i < 1$.

First we bound how much energy a node spends on any light class it is part of. A similar bound for heavy classes follows afterward. Combined, we arrive at the advantage bound.

**Lemma 8.21.** *For every* $v \in V$ *and integer* $r$,

$$\mathbb{P}\left[\sum_{i \in L_v^r} F_i \leqslant (1+\gamma)/4\right] \geqslant 1 - 1/n^8.$$

*Proof.* By use of the Chernoff bound[7] and by setting $(1+\gamma') = (1+\gamma)/4 = 8\log n/\log\log n$ we obtain:

$$\begin{aligned}
\mathbb{P}\left[\sum_{i \in L_v^r} F_i > (1+\gamma')\right] &< \left[\frac{e^{8\log n/\log\log n - 1}}{(8\log n/\log\log n)^{(8\log n/\log\log n)}}\right] \\
&< \left[\frac{e}{(8\log n/\log\log n)}\right]^{(8\log n/\log\log n)} \\
&< \left[\frac{e\log\log n}{8\cdot 2^{\log\log n}}\right]^{(8\log n/\log\log n)} \\
&< \left[\frac{e}{8}\right]^{8\log n} \\
&< \left[\frac{1}{2}\right]^{8\log n} \\
&< \frac{1}{n^8}
\end{aligned}$$

□

**Lemma 8.22.** *For every* $v \in V$,

$$\mathbb{P}\left[\sum_{r=1}^{t} \sum_{u: vu \in E} e_{vu} \sum_{i: i \in L_v^r \wedge vu \in P_i} F_i \leqslant (1+\gamma)E_v\right] \geqslant 1 - 1/n^7.$$

*Proof.* It follows easily from Lemma 2 and the Chernoff bound that

$$\mathbb{P}\left[\bigvee_{r=1}^{t}\left(\sum_{i \in L_v^r} F_i \leqslant (1+\gamma)/4\right)\right] \geqslant 1 - 1/n^7.$$

[7]See Proposition 2.3 on page 19.

Rewriting gives the following.

$$
\begin{aligned}
&\mathbb{P}\left[\bigvee_{r=1}^{t}\left(\sum_{i\in L_v^r} F_i \leqslant (1+\gamma)/4\right)\right] && (8.34)\\
&= \mathbb{P}\left[\bigvee_{r=1}^{t}\left(2^{r+1}\sum_{i\in L_v^r} F_i \leqslant 2^r(1+\gamma)/2\right)\right] && (8.35)\\
&< \mathbb{P}\left[\bigvee_{r=1}^{t}\left(\sum_{u:vu\in E} e_{vu} \sum_{i:i\in L_v^r\wedge vu\in P_i} F_i \leqslant 2^r(1+\gamma)/2\right)\right] && (8.36)\\
&< \mathbb{P}\left[\sum_{r=1}^{t}\sum_{u:vu\in E} e_{vu} \sum_{i:i\in L_v^r\wedge vu\in P_i} F_i \leqslant \sum_{r=1}^{t} 2^r(1+\gamma)/2\right] && (8.37)\\
&< \mathbb{P}\left[\sum_{r=1}^{t}\sum_{u:vu\in E} e_{vu} \sum_{i:i\in L_v^r\wedge vu\in P_i} F_i \leqslant 2E_v(1+\gamma)/2\right] && (8.38)\\
&= \mathbb{P}\left[\sum_{r=1}^{t}\sum_{u:vu\in E} e_{vu} \sum_{i:i\in L_v^r\wedge vu\in P_i} F_i \leqslant E_v(1+\gamma)\right] && (8.39)
\end{aligned}
$$

□

**Lemma 8.23.** *For every* $v \in N$ *and integer* $r$*,*

$$\mathbb{P}\left[\sum_{i\in H_v^r} F_i \leqslant (1+\gamma)/4 \sum_{i\in H_v^r} f_i\right] \geqslant 1 - 1/n^8.$$

*Proof.* Similar to the proof of Lemma 8.21. □

**Lemma 8.24.** *For every* $v \in N$*,*

$$\mathbb{P}\left[\sum_{r=1}^{t}\sum_{u:vu\in E} e_{vu} \sum_{i:i\in H_v^r\wedge vu\in P_i} F_i \leqslant (1+\gamma)E_v\right] \geqslant 1 - 1/n^7.$$

*Proof.* It follows easily from Lemma 2 and the Chernoff bound that

$$\mathbb{P}\left[\bigvee_{r=1}^{t}\left(\sum_{i\in H_v^r} F_i \leqslant (1+\gamma)/4 \sum_{i\in H_v^r} f_i\right)\right] \geqslant 1 - 1/n^7.$$

Rewriting gives the following.

$$\mathbb{P}\left[\bigvee_{r=1}^{t}\left(\sum_{i\in H_v^r} F_i \leqslant \sum_{i\in H_v^r} f_i(1+\gamma)/4\right)\right] \tag{8.40}$$

$$= \mathbb{P}\left[\bigvee_{r=1}^{t}\left(2^{r+1}\sum_{i\in H_v^r} F_i \leqslant 2^r \sum_{i\in H_v^r} f_i(1+\gamma)/2\right)\right] \tag{8.41}$$

$$< \mathbb{P}\left[\bigvee_{r=1}^{t}\left(\sum_{u:vu\in E} e_{vu} \sum_{i:i\in H_v^r \wedge vu\in P_i} F_i \leqslant 2^r \sum_{i\in H_v^r} f_i(1+\gamma)/2\right)\right] \tag{8.42}$$

$$< \mathbb{P}\left[\sum_{r=1}^{t}\sum_{u:vu\in E} e_{vu} \sum_{i:i\in H_v^r \wedge vu\in P_i} F_i \leqslant \sum_{r=1}^{t} 2^r \sum_{i\in H_v^r} f_i(1+\gamma)/2\right] \tag{8.43}$$

$$< \mathbb{P}\left[\sum_{r=1}^{t}\sum_{u:vu\in E} e_{vu} \sum_{i:i\in H_v^r \wedge vu\in P_i} F_i \leqslant E_v(1+\gamma)/2\right] \tag{8.44}$$

□

**Theorem 8.25.** *There is a polynomial time algorithm that returns an integer solution with optimal flow and that exceeds the energy usage of each vertex by at most a factor* $O(\log n/\log\log n)$ *with probability at least* $1 - 2/n^6$.

*Proof.* From Lemma 8.21 and Lemma 8.23 we have that for every single vertex, the solution does not exceed the available energy by more than a factor $O(\log n/\log\log n)$ with probability $1 - 2/n^7$. It follows that with probability at least $1 - 2/n^6$ this holds *for all* vertices. From an easy Chernoff estimation we get that almost surely $\sum_i F_i = \sum_i f_i$, that is, that we get the optimal flow. □

This result, in terms of energy advantage, is inferior to the existing advantage-4 algorithm of Nutov [Nut08]. It can be remarked, however, that our algorithm is likely to run faster in practice: it is a simple randomised rounding procedure on top of a column generation algorithm that we have experimentally demonstrated to be efficient. Analysis or experimental evaluation of the runtime of Nutov's algorithm is unavailable, but as the algorithm is based on *iterated* rounding of a large linear program, it can be expected to perform poorly in comparison.

## 8.9. Concluding remarks

We have given a basic approximation algorithm for the energy-constrained flow problem and have shown that it has an additive approximation guarantee. We have developed a column generation algorithm for the fractional relaxation of the problem and use an implementation of this algorithm to show that it outperforms the normal linear-programming formulation. We demonstrate experimentally that this approach also leads to an effective heuristic for the integer problem. We have given two algorithms with advantage. Based on a result of Nutov, these also give approximation algorithms. In the case of $st$-planar networks, we improve on the best known approximation factor. In the case of general networks, our algorithm has worse advantage: $\mathcal{O}(\log n / \log \log n)$ instead of constant, but the algorithm is simpler and more efficient.

# 9 Concluding remarks

In this thesis we have studied various problems in the context of wireless sensor networks. First we considered two problems in physical-model networks.

In Chapter 3 we have introduced a new model for localisation in wireless networks, and sensor networks in particular. The model is based on a range-free model of radio transmissions. We have presented several localisation schemes in this model. These cope with range-freeness and are designed to be cheap in terms of local computation and energy consumption.

The first scheme is randomised and we have analysed its expected performance. Then we introduced the concept of a *splitline schedule*, which is the central concept for the rest of the chapter. We conjectured that a certain class of schedules (the *regular* schedules) is optimal. We have proved this conjecture for some restricted cases, but the general case remains open.

**Open Problem 1.** *Resolve Conjecture 1 (page 48).*

Then we defined and analysed the *reverse-binary* schedules and proved that these have a performance that is within a constant factor of optimal. An improvement in this factor (either from a better schedule or from a better lower bound) would be interesting.

**Open Problem 2.** *Give better deterministic splitline schedules.*

These results are for one-dimensional localisation; in higher dimensions we have shown how to localise in a hypercube spanned by base stations at its vertices. It seems that $d+1$ base stations should suffice for localisation in a $d$-dimensional simplex, but we have not investigated this. For high $d$, this would significantly reduce the number of base stations required in comparison to our hypercube approach, which uses $2^d$. For the important cases of $d \leqslant 3$ the difference is not large, but still it is a natural objective to minimise the number of base stations.

**Open Problem 3.** *Give more efficient splitline constructions in higher dimensions.*

Next we considered a problem in the scheduling of wireless transmissions. In Chapter 4 we have provided an exact exponential-time algorithm for the Link Independent Set (Abstract) problem. Its branching rules are valid for arbitrary *gain matrices*, but designed to take advantage of the structure available in geometric instances. We have proved a worst-case bound of $\mathcal{O}^*(1.888^n)$ on the runtime of the algorithm, but only under an assumption on the input. In Chapter 5 we have experimentally demonstrated that this assumption is reasonable for random geometric instances and that, in fact, the effective branching factor of the algorithm is expected to be much better than 1.888 on such instances. This state of affairs suggests two further questions.

**Open Problem 4.** *Give a better bound on the worst-case runtime of Algorithm 1 (page 75): experiments suggest that there is room for improvement.*

**Open Problem 5.** *In the analysis of the worst-case runtime of Algorithm 1, reduce the reliance on structural assumptions about the input.*

For random geometric instances we considered the *density* of the instance, that is, how many nodes are expected in a unit area. Further experiments showed that depending on this density of the instance, the algorithm performs well for different reasons: either the branching and reduction rules, the usage of branch and bound, or both. We have shown that the density is also a meaningful parameter for the problem in structural terms (as opposed to just algorithmic): experiments as well as theory tell us that the size of the optimal link independent set is closely tied the density of the instance.

After these results for physical-model networks, we turned our attention to graphs. In Chapter 6 we have introduced a model of recoverable routing in the presence of node failures. We call the resulting planning problem Recoverable path. The model is based on the concept of *backup nodes*. For every node in the network, we assign a backup node: the latter will take over in case the former fails. The motivation for the Recoverable path problem *robust recoverability*. Choosing a solution that is feasible for any failure scenario is overly cautious—in our particular case it would only allow paths of one hop. Unrestricted replanning in case of a failure, however, can be too costly in terms of computational power or the information available. We therefore calculate a route that is *recoverable*: in case of a failure, the backup nodes can be used to fix the route easily and locally.

We have resolved the basic algorithmic and complexity-related questions about Recoverable path: some variants are polynomial-time solvable and others are $\mathcal{NP}$-complete. For the polynomially-solvable cases we have given an $\mathcal{O}(n^4)$-time algorithm. This stands in stark contrast to the $\mathcal{NP}$-completeness of the other cases and there are no unreasonable constant factors involved in the runtime. However, one might hope for a lower exponent.

**Open Problem 6.** *Solve* Recoverable path *with many-to-many backup in* $o(n^4)$ *time.*

We have given non-trivial exponential-time algorithms for the hard cases of the problem. The runtime for one problem variant in particular is worse than the others. This seems inevitable with the employed techniques, but begs for improvement.

**Open Problem 7.** *Solve* Recoverable path *with one-to-one backup in* $o(4^n)$ *time.*

Then we looked at a variant of the problem where the path is given and we are merely asked to pick which node is backed up by which other node. We call this problem Backup assignment. Again we see that the problem has variants that are polynomial-time solvable and variants that are $\mathcal{NP}$-complete. We have given algorithms for all variants.

We finished the chapter by making some preliminary remarks about parameterised versions of the hard Backup assignment variant. We posed the general question and briefly discussed possible parameterisations.

**Open Problem 8.** *Give fixed parameter tractable algorithms for reasonable parameterisations of* Backup assignment *with injective backup.*

**Open Problem 9.** *Resolve the complexity of* Backup assignment *with injective backup parameterised by* $\mu_{max}$.

**Open Problem 10.** *Resolve the complexity of* Backup assignment *with injective backup parameterised by* $\sum_{v \in V} \Big(\mu(v) - 1\Big)$.

In Chapters 7 and 8, finally, we have looked at a problem of energy-efficient data gathering in wireless sensor networks. We have formalised a version of this problem and call it *energy-constrained flow*: the measurements taken by the sensors are relayed to a central base station, where these relay nodes have limited battery power. We have studied, in particular, the integer version of the problem, where we consider the flow of measurements to consist of packets that cannot be split.

We have proved $\mathcal{NP}$-hardness and $\mathcal{APX}$-hardness in various settings. Among other results, we have shown that the problem is strongly $\mathcal{NP}$-complete on geometric configurations on a line. On networks of bounded treewidth greater than two, the problem is only weakly $\mathcal{NP}$-complete. For treewidth two, we have given a polynomial-time algorithm if the source and the sink are connected.

After the mostly-negative results of Chapter 7, we presented a column generation algorithm for the fractional relaxation of energy-constrained flow. We have implemented this column-generation approach and showed that it outperforms the normal linear-programming formulation. We demonstrated experimentally that this approach also leads to an effective heuristic for the integer problem.

One of the variants of algorithm uses an *alternating* objective function during the column generation procedure.

**Open Problem 11.** *Is alternating column generation, which alternates between two objective functions on the same set of variables, useful beyond the specific problem studied here?*

We finished the thesis with two algorithms *with advantage*. For the energy-constrained flow problem, such algorithms with advantage can be converted into approximation algorithms. In this way, we have improved the best known approximation factor when restricted to $st$-planar networks. In the case of general networks, our algorithm with advantage does not improve the best known approximation factor: it has factor $\mathcal{O}(\log n/\log\log n)$ instead of constant. However, our algorithm is simpler and more efficient.

**Open Problem 12.** *Give improved approximation algorithms for energy-constrained flow.*

In this thesis we have shown that understanding the structure of wireless sensor networks leads to many challenging questions in algorithmic design, requiring a blend of techniques ranging from network analysis to probability theory and modern complexity theory. Computational experiments have proven to be an indispensable tool for discovering new results; we believe that many of the open problems mentioned in this section will benefit from the same methodology.

# Bibliography

[Abr70] N. Abramson. The ALOHA system: another alternative for computer communications. In *Proceedings of the November 17-19, 1970, Fall Joint Computer Conference (AFIPS '70)*, pages 281–285, New York, NY, USA, 1970. ACM.

[ACG$^+$99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, Berlin, Germany, 1999.

[ACPS93] S. Arnborg, B. Courcelle, A. Proskurowski, and D. Seese. An algebraic theory of graph reduction. *Journal of the ACM (JACM)*, 40:1134–1164, 1993.

[AD09] M. Andrews and M. Dinitz. Maximizing capacity in arbitrary wireless networks in the SINR model: Complexity and game theory. In *IEEE International Conference on Computer Communications (INFOCOM 2009)*, pages 1332–1340. IEEE, 2009.

[ADHT11] J. M. van den Akker, T.C. van Dijk, J.A. Hoogeveen, and T. Toorop. Optimizing wireless sensor network flow by column generation. In *24th Conference of the European Chapter on Combinatorial Optimization (ECCO)*. 2011.

[AEK$^+$12] C. Avin, Y. Emek, E. Kantor, Z. Lotker, D. Peleg, and L. Roditty. SINR diagrams: Convexity and its applications in wireless networks. *Journal of the ACM (JACM)*, 59(4):18:1–18:34, 2012.

[AK09] I. Amundson and X. D. Koutsoukos. A survey on localization for mobile wireless sensor networks. In *Mobile Entity Localization and Tracking in GPS-less Environnments*, pages 235–254, Berlin, 2009. Springer.

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Upper Saddle River, NJ, USA, 1993.

[AYB13] A. A. Abbasi, M. F. Younis, and U. A. Baroudi. Recovering from a node failure in wireless sensor-actor networks with minimal topology changes. *IEEE Transactions on Vehicular Technology*, 62(1):256–271, 2013.

[BH06] A. Björklund and T. Husfeldt. Inclusion–exclusion algorithms for counting set partitions. In *Proceedings of the 47th Annual Symposium on Foundations of Computer Science (FOCS '06)*, pages 575–582, Los Alamitos, CA, USA, 2006. IEEE.

[BH14] M. H. L. Bodlaender and M. M. Halldórsson. Beyond geometry: Towards fully realistic wireless models. *Computing Research Repository (CoRR)*, abs/1402.5003, 2014.

[BHE00] N. Bulusu, J. Heidemann, and D. Estrin. GPS-less low cost outdoor localization for very small devices. *IEEE Personal Communications Magazine*, 7(5):28–34, 2000.

[BK96] H. L. Bodlaender and T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21:358–402, 1996.

[Bod93] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–23, 1993.

[Bod98] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209:1–45, 1998.

[BT05] J. Bachrach and C. Taylor. *Handbook of Sensor Networks*, chapter Localization in Sensor Networks, pages 277–310. John Wiley & Sons, Inc., 2005.

[BTvDvL08] H. L. Bodlaender, R. B. Tan, T. C. van Dijk, and J. van Leeuwen. Integer maximum flow in wireless sensor networks with energy constraint. In *Algorithm Theory – SWAT 2008*, pages 102–113. Springer Berlin Heidelberg, Germany, 2008.

[BvAdF01] H. L. Bodlaender and B. van Antwerpen-de Fluiter. Reduction algorithms for graphs of small treewidth. *Information and Computation*, 167:86–119, 2001.

[CK05] C. Chekuri and S. Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 35(3):713–728, 2005.

[CLRS09] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, USA, 3rd edition, 2009.

[CT00a] J.-H. Chang and L. Tassiulas. Energy conserving routing in wireless ad-hoc networks. In *Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2000)*, pages 22–31. IEEE, 2000.

[CT00b] J.-H. Chang and L. Tassiulas. Fast approximate algorithms for maximum lifetime routing in wireless ad-hoc networks. In *Networking 2000*, pages 702–713, Germany, 2000. Springer Berlin.

[CT04] J. Chang and L. Tassiulas. Maximum lifetime routing in wireless sensor networks. *IEEE/ACM Transaction on Networking*, 12(4):609–619, 2004.

[DF13] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer-Verlag London, 2013.

[DKQW10] Y. Ding, N. Krislock, J. Qian, and H. Wolkowicz. Sensor network localization, Euclidean distance matrix completions, and graph realization. *Optimization and Engineering*, (1):45–66, 2010.

[DN05] H. A. David and H. N. Nagaraja. *Order Statistics*. John Wiley & Sons, Inc., 2005.

[Fen94] P. M. Fenwick. A new data structure for cumulative frequency tables. *Software: Practice and Experience*, 24:327–336, 1994.

[FKKO05] P. Floréen, P. Kaski, J. Kohonen, and P. Orponen. Exact and approximate balanced data gathering in energy-constrained sensor networks. *Theoretical Computer Science*, 344(1):30–46, 2005.

[FL06] R. Fan and N. Lynch. Gradient clock synchronization. *Distributed Computing*, 18(4):255–266, 2006.

[GÁB$^+$14] H. Gudmundsdottir, E. I. Ásgeirsson, M. H. L. Bodlaender, J. T. Foley, M. M. Halldórsson, G. M. Järvelä, H. Úlfarsson, and Y. Vigfusson. Measurement based interference models for wireless scheduling algorithms. *Computing Research Repository (CoRR)*, abs/1401.1723, 2014.

[GDP04] S. R. Gandham, M. Dawande, and R. Prakash. An integral flow-based energy-efficient routing algorithm for wireless sensor networks. In *Proceedings of IEEE Wireless Communications and Networking Conferrence (WCNC)*, pages 2341–2346. IEEE, 2004.

[GDPV03] S. R. Gandham, M. Dawande, R. Prakash, and S. Venkatesan. Energy-efficient schemes for wireless sensor networks with multiple mobile base stations. In *Proceedings of the IEEE Global Communications Conference (GLOBECOM '03)*, pages 377–381, 2003.

[Gil61] E. N. Gilbert. Random plane networks. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):533–543, 1961.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, NY, USA, 1979.

[GK98] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS '98)*, pages 300–309, Los Alamitos, CA, USA, 1998. IEEE.

[GK00] P. Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2):388–404, 2000.

[GKL03] A. Gupta, R. Krauthgamer, and J. R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *Proceedings of the 44th Annual Symposium on Foundations of Computer Science (FOCS '03)*, pages 534–543. IEEE, 2003.

[GKW$^+$02] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker. Complex behavior at scale: An experimental study of low-power wireless sensor networks. Technical report, UCLA Computer Science Department, 2002.

[GWHW09] O. Goussevskaia, R. Wattenhofer, M.M. Halldórsson, and E. Welzl. Capacity of arbitrary wireless networks. In *IEEE International Conference on Computer Communications (INFOCOM 2009)*, pages 1872–1880. IEEE, 2009.

[HCB00] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Hawaii International Conference on System Sciences*, pages 3005–3014. IEEE, 2000.

[HHB$^+$03] T. He, C. Huang, B. M. Blum, J. A. Stankovic, and T. Abdelzaher. Range-free localization schemes for large scale sensor networks. In *Proceedings of the 9th annual international conference on Mobile computing and networking (MobiCom '03)*, pages 81–95, New York, NY, USA, 2003. ACM.

[HL08] Q.-S. Hua and F. C.M. Lau. Exact and approximate link scheduling algorithms under the physical interference model. In *Proceedings of the 5th international workshop on Foundations of mobile computing (DIALM-POMC '08)*, pages 45–54, New York, NY, USA, 2008. ACM.

[HM11] M. Halldórsson and P. Mitra. Nearly optimal bounds for distributed wireless scheduling in the sinr model. In *Automata, Languages and Programming*, pages 625–636. Springer Berlin Heidelberg, Germany, 2011.

[HN79] H. Hitotumatu and K. Noshita. A technique for implementing backtrack algorithms and its application. *Information Processing Letters*, 8(4):174–175, 1979.

[HP06] B. Hong and V. K. Prasanna. Maximum lifetime data sensing and extraction in energy constrained networked sensor systems. *Journal of Parallel and Distributed Computing*, 66(4):566–577, 2006.

[HW09] M. Halldórsson and R. Wattenhofer. Wireless communication is in APX. In *Automata, Languages and Programming*, pages 525–536. Springer Berlin Heidelberg, Germany, 2009.

[KBK$^+$10] A. Knobbe, H. Blockeel, A. Koopman, T. Calders, B. Obladen, C. Bosma, H. Galenkamp, E. Koenders, and J. Kok. Infrawatch: Data management of large systems for monitoring infrastructural performance. In *Advances in Intelligent Data Analysis IX*, pages 91–102. Springer Berlin Heidelberg, Germany, 2010.

[KDN03] K. Kalpakis, K. Dasgupta, and P. Namjoshi. Efficient algorithms for maximum lifetime data gathering and aggregation in wireless sensor networks. *Computer Networks*, 42(6):697–716, 2003.

[KNMW05] F. Kuhn, T. Nieberg, T. Moscibroda, and R. Wattenhofer. Local approximation schemes for ad hoc and sensor networks. In *Proceedings of the 2005 joint workshop on Foundations of mobile computing (DIALM-POMC '05)*, pages 97–103, New York, NY, USA, 2005. ACM.

[Knu00] D. E. Knuth. Dancing links. *Millennial perspectives in Computer Science*, pages 187–214, 2000.

[Kur08] P. Kurp. Green computing. *Communications of the ACM (CACM)*, 51(10):11–13, 2008.

[LEH04] T. Li, A. Ekpenyong, and Y.-F. Huang. A location system using asynchronous distributed sensors. In *Proceedings of the 23th IEEE International Conference on Computer Communications (INFOCOM 2004)*. IEEE, 2004.

[Max13] Max_87. Desktop CPU comparison guide rev. 14.0. `http://www.techarp.com/showarticle.aspx?artno=337&pgno=0`, 2013.

[Met93] B. Metcalfe. Wireless computing will flop—permanently. *InfoWorld*, 15(33):48, 1993.

[MU05] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[MWW06] T. Moscibroda, R. Wattenhofer, and Y. Weber. Protocol Design Beyond Graph-Based Models. In *5th Workshop on Hot Topics in Networks (HotNets '06)*, 2006.

[NRR10] A. Nanda, A. K. Rath, and S. K. Rout. Node sensing & dynamic discovering routes for wireless sensor networks. *Computing Research Repository (CoRR)*, abs/1004.1678, 2010.

[NSB03] R. Nagpal, H. Shrobe, and J. Bachrach. Organizing a global coordinate system from local information on an ad hoc sensor network. In *Proceedings of the 2nd international conference on Information processing in sensor networks (IPSN '03)*, pages 333–348, Germany, 2003. Springer Berlin Heidelberg.

[Nut08] Z. Nutov. Approximating maximum integral flows in wireless sensor networks via weighted-degree constrained k-flows. In *Proceedings of the Fifth International Workshop on Foundations of Mobile Computing (DIALM-POMC '08)*, pages 29–34, New York, NY, USA, 2008. ACM.

[OK04] F. Ordonez and B. Krishnamachari. Optimal information extraction in energy-limited wireless sensor networks. *IEEE Journal on Selected Areas in Communications*, 22(6):1121–1129, 2004.

[PCV04] M. Patel, R. Chandrasekaran, and S. Venkatesan. Efficient minimum-cost bandwidth-constrained routing in wireless sensor networks. In *Proceedings of the International Conference on Wireless Networks (ICWN '04)*, pages 447–453, Athens, GA, USA, 2004. CSREA Press.

[Ram97] S. Ramachandramurthi. The structure and number of obstructions to treewidth. *SIAM Journal on Disrete Mathematics*, 10:146–157, 1997.

[RDBL12] M. Radi, B. Dezfouli, K. A. Bakar, and M. Lee. Multipath routing in wireless sensor networks: survey and research challenges. *Sensors*, 12(1):650–685, 2012.

[RMR$^+$06] C. Reis, R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan. Measurement-based models of delivery and interference in static wireless networks. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '06)*, pages 51–62, New York, NY, USA, 2006. ACM.

[RN10] S.J. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Prentice Hall series in artificial intelligence. Prentice Hall, 2010.

[Sax79] J. B. Saxe. Embeddability of weighted graphs in k-space is strongly NP-hard. *Proceedings of the 17th Allterton Conference in Communications, Control and Computing*, pages 480–489, 1979.

[SL07] M. Singh and L. C. Lau. Approximating minimum bounded degree spanning trees to within one of optimal. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC '07)*, pages 661–670, New York, NY, USA, 2007. ACM.

[SSGE04] A. Savvides, M. Srivastava, L. Girod, and D. Estrin. *Wireless Sensor Networks*, chapter Localization in Sensor Networks, pages 327–349. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[Tic98] Walter F. Tichy. Should computer scientists experiment more? *Computer*, 31(5):32–40, 1998.

[VKV$^+$11] U. Vespier, A. Knobbe, J. Vanschoren, S. Miao, A. Koopman, B. Obladen, and C. Bosma. Traffic events modeling for structural health monitoring. In *Proceedings of the 10th international conference on Advances in intelligent data analysis X (IDA'11)*, pages 376–387, Berlin, Heidelberg, 2011. Springer.

[WC06] Y.-H. Wang and C.-F. Chao. Dynamic backup routes routing protocol for mobile ad hoc networks. *Information Sciences*, 176(2):161–185, 2006.

[WZW06] N. Wang, N. Zhang, and M. Wang. Wireless sensors in agriculture and food industry—recent development and future perspective. *Computers and Electronics in Agriculture*, 50(1):1–14, 2006.

[XCN05] Y. Xue, Y. Cui, and K. Nahrstedt. Maximizing lifetime for data aggregation in wireless sensor networks. *Mobile Networks and Applications*, 10:853–864, 2005.

[Xia13] G. Xia. The stretch factor of the Delaunay triangulation is less than 1.998. *SIAM Journal on Computing*, 42(4):1620–1659, 2013.

[XTW10] X. Xu, S. Tang, and P.-J. Wan. Maximum weighted independent set of links under physical interference model. In *Proceedings of the 5th International Conference on Wireless Algorithms, Systems, and Applications (WASA '10)*, pages 68–74. Springer Berlin Heidelberg, Germany, 2010.

[ZS03] G. Zussman and A. Segall. Energy efficient routing in ad hoc disaster recovery networks. *Ad Hoc Networks*, 1(4):405–421, 2003.

# Index

# Samenvatting in het Nederlands

Draadloze technologie heeft de wereld veroverd. Veel mensen hebben een smartphone die verbonden is met mobiel Internet en veel thuisnetwerken zijn tegenwoordig draadloos. Draadloze technologie bestaat natuurlijk al langer dan dat het breed wordt toegepast in consumentenprodukten—op de tijdschaal van de informatica: *veel* langer. Oorspronkelijk ging het echter vooral om uitzonderlijke oplossingen voor uitzonderlijke problemen. Tegenwoordig zijn draadloze verbindingen eerder de standaard dan een uitzondering, of in ieder geval zijn ze serieus te overwegen in vrijwel alle toepassingen. Dit maakt bestaande gevallen aangenamer (bijvoorbeeld het draadloze thuisnetwerk), maar het heeft ook volledig nieuwe toepassingsgebieden geopend, bijvoorbeeld sensornetwerken, *ambient intelligence* en een *Internet of Things*. Deze draadloze wereld biedt ongekende mogelijkheden en stelt de informatica voor vele nieuwe onderzoeksvragen. We belichten enkele van deze vragen en geven aan hoe we er in dit proefschrift op in gaan.

In dit proefschrift hebben we in het bijzonder aandacht voor sensornetwerken. Dat is een algemene term die veel omvat en verschillend opgevat kan worden, maar de algemene eigenschappen kunnen als volgt geduid worden. Een sensornetwerk bestaat uit een groot aantal kleine apparaten, elk uitgerust met sensoren, een radio en een batterij. De individuele apparaten worden doorgaans *sensor nodes* genoemd en samen verzamelen ze gegevens over hun omgeving.

Bij een sensornetwerk ligt de nadruk op andere punten dan bij een doorsnee draadloos netwerk. Zo wordt doorgaans verondersteld dat de hardware goedkoop is, en niet altijd betrouwbaar. Tevens worden sensornetwerken vaak toegepast op onherbergzame of gevaarlijke plekken, waar het opstellen van het netwerk en het onderhoud ervan moeilijk zijn. Vanwege deze twee aspecten is er vaak aandacht voor fouttolerantie: het kan efficiënt zijn om met zo goedkoop mogelijke hardware een systeem te maken dat de problemen met individuele sensor nodes fouttolerant opvangt in het netwerk. Ook energiezuinigheid is van groot belang: een sensor node is effectief buiten werking zodra zijn batterij leeg is.

De theorie van netwerken is grotendeels gebaseerd op grafen: dat is het standaardmodel voor communicatie in bedrade netwerken. Grafen, zo is in vele studies gebleken, zijn het 'juiste' model. De eigenschappen van draadloze netwerken laten zich echter niet zo makkelijk vangen in graafgebaseerde modellen; het is allesbehalve duidelijk wat het 'correcte' model voor draadloze netwerken is. Gemeengoed bij de nieuwe modellen die onderzocht worden is een sterke focus op de fysische eigenschappen van radiosignalen. In Deel I van dit proefschrift kijken we naar twee van zulke *fysische modellen*: we geven een nieuw model voor lokalisatie, en nieuwe resultaten in het bekende *Signal to Interference plus Noise Ratio (SINR)* model.

**Lokalisatie.** Fysische modellen zijn van groot belang bij *lokalisatie*, een probleem dat bijzonder relevant is voor sensornetwerken: waar ben ik? De positie van een sensor node is namelijk vaak van belang, om te registreren waar de metingen genomen zijn. GPS is in veel gevallen niet geschikt voor toepassing in sensornetwerken en ook lokalisatie door middel van *graafrealisatie* is vaak niet praktisch toepasbaar in deze context: de daarvoor benodigde afstandsmetingen zijn zelden met de vereiste nauwkeurigheid te verkrijgen. Daarom is er interesse in zogenaamde afstandsvrije lokalisatie *(range-free localisation)*: lokalisatie zonder dat er expliciet afstanden gemeten worden tussen de sensor nodes.

In Hoofdstuk 3 introduceren we een nieuwe model voor afstandsvrije lokalisatie. De lokalisatie vindt plaats in relatie tot een aantal basisstations, die (effectief) niet beperkt zijn in hun energieverbruik en beschikken over een krachtige radiozender. Het lokalisatiesysteem is vervolgens ontworpen om zo simpel mogelijk te zijn voor de sensor nodes: die zijn immers het zwaarst onderhevig aan beperkingen (functionaliteit, batterijcapaciteit, foutgevoeligheid).

De basisstations versturen een voortdurende reeks berichten en de sensor nodes stemmen hier op af als ze zich willen lokaliseren. We modelleren dit systeem en analyseren eerst het verwachte gedrag bij een probabilistisch zendschema. Dan introduceren we het concept *scheidslijnschema (splitline schedule)* dat centraal staat in de rest van het hoofdstuk. We formuleren het vermoeden dat een bepaalde klasse schema's (de *reguliere* schema's) optimaal zijn. We bewijzen dit voor een aantal gevallen, maar het algemene vermoeden blijft open. Vervolgens definiëren we het *omgekeerd binaire* schema (*reverse-binary schedule*) en bewijzen dat het op constante factoren na optimaal is.

**Zendschema's voor draadloze berichten.** In Hoofdstukken 4 en 5 bestuderen we het *Signal to Interference plus Noise Ratio* (*SINR*) model. Dit model geeft voor een gegeven verzameling radiozenders aan wiens bericht waar te ontvangen is, gebaseerd op een fysisch model van de sterkte van radiosignalen. We kijken naar een probleem met betrekking tot zendschema's voor draadloze berichten *(wireless scheduling)*. Als input krijgen we de posities van de draadloze zenders en ontvangers, en een verzameling rechtstreeks te verzenden berichten. Daarin zoeken we een *link independent set* van maximale grote: een zo groot mogelijke verzameling berichten die tegelijkertijd verzonden kan worden.

Het berekenen van een maximum link independent set is $\mathcal{NP}$-compleet. In Hoofdstuk 4 ontwerpen we een branching algoritme voor het probleem en tonen aan dat de looptijd ervan niettemin laag exponentieel blijft. In Hoofdstuk 5 beschrijven we een implementatie van het algoritme en demonstreren dat het in de praktijk goed werkt. Met deze implementatie onderzoeken we vervolgens de eigenschappen van ons algoritme en van willekeurige geometrische instanties. Daarbij blijkt de *dichtheid* van instanties van groot belang, dat wil zegggen, het verwachtte aantal zenders per eenheid oppervlak. Dit beïnvloedt welke aspecten van het algoritme effectief zijn (de specifieke branchingregels dan wel het toepassen van *branch and bound*) en blijkt tevens een betekenisvolle parameter in structurele zin. We tonen experimenteel aan dat de grootte van een optimale link independent set sterk afhangt van de dichtheid van de instantie. Hierover bewijzen we vervolgens dat erg grote link independent sets onwaarschijnlijk zijn, maar dat er met hoge kans wel een redelijk grote link independent set is.

In Deel II van dit proefschrift onderzoeken we diverse vraagstellingen voor sensornetwerken met behulp van grafentheoretische modellering. De nadruk ligt hierbij op fouttolerantie en batterijbesparende algoritmen.

**Robuste routering.** Fouttolerantie is van groot belang in sensornetwerken, omdat de hardware en omgevingsomstandigheden zorgen voor onbetrouwbaarheid in het netwerk. Het is in veel gevallen een redelijk scenario dat een sensor node het begeeft en simpelweg niet meer functioneert. In Hoofdstuk 6 kijken we daarom naar *robuust herstel (robust recovery)*. Daarbij wordt gewerkt met een herstelprocedure die eventueel optredende problemen ter plekke opvangt. Het cruciale idee van robuust herstel is vervolgens om bij de oorspronkelijke planning al rekening te houden met het gedrag van deze herstelprocedure: een extreem simpele herstelprocedure volstaat wellicht, als we er maar goed rekening mee houden bij het plannen van de basisoplossing.

We bestuderen robuust herstel voor het routeren van pakketten in een netwerk, wat we modelleren als het vinden van een toegelaten pad in een graaf. Ten behoeve van de herstelprocedure wijzen we, van tevoren, voor elke knoop een *reserveknoop (backup node)* aan. Herstel kan dan simpel en lokaal plaatsvinden, door het bericht aan de reserveknoop te sturen als het niet lukt om de eigenlijk geplande ontvanger te bereiken.

We kijken naar het vinden van een pad tussen twee gegeven knopen en het daarbij toewijzen van geldige reserveknopen. Eerst formuleren we dit routeringsprobleem exact, waarbij we succes willen garanderen bij het uitvallen van hooguit één knoop. Voor een aantal varianten geven we polynomiale algoritmen. We bewijzen dat de andere varianten $\mathcal{NP}$-volledig zijn en voor die gevallen geven we non-triviale algoritmen met exponentiële looptijd. Daarna introduceren we het *reservetoewijzingsprobleem*, waar het basispad gegeven is en we alleen nog de reserveknopen moeten toewijzen. Wederom zijn een aantal varianten polynomiaal oplosbaar (maar niet precies dezelfde varianten) en is er een $\mathcal{NP}$-volledige variant. Ook voor al deze gevallen geven we algoritmen. Tot slot leggen we de basis voor de bestudering van de geparametriseerde complexiteit van het reservetoewijzingsprobleem.

**Energiezuinige gegevensinzameling.** Als laatste thema kijken we in Hoofdstukken 7 en 8 naar energiezuinigheid. Dit is van groot belang bij sensornetwerken, aangezien de batterijen van sensor nodes vaak niet te vervangen of op te laden zijn: zodra de batterij leeg is, is de sensor node verloren. Aangezien de primaire taak van sensornetwerken er in ligt metingen te doen, doet zich de volgende vraag voor. Hoe kunnen we zo lang mogelijk (of: zo veel mogelijk) gegevens verzamelen voordat het netwerk aan energiegebrek ten onder gaat?

We modelleren dit als een graafprobleem: verzamel zo veel mogelijk berichten in een gegeven knoop (een basisstation), gegeven een aantal knopen die voortdurend metingen verrichten. Daarbij begint elke knoop met een hoeveelheid energie om te besteden aan het versturen of doorsturen van berichten. We noemen het probleem *energiebeperkte flow (energy-constrained flow)* vanwege de gelijkenis met het klassieke netwerkflow problem.

We bewijzen in Hoofdstuk 7 dat het maximaliseren van energiebeperkte flow sterk $\mathcal{NP}$-compleet is en zelfs $\mathcal{APX}$-moeilijk. Het probleem blijft moeilijk als we ons beperken tot geometrische netwerken met fysisch redelijke verzendkosten. Voor netwerken met begrensde boombreedte geven we een pseudopolynomial algoritme; specifiek voor boombreedte twee geven we een polynomiaal algoritme (mits de bronknoop en de doelknoop verbonden zijn).

In Hoofdstuk 8 tonen we aan dat er ondanks deze moeilijkheidsbewijzen het een en ander te bereiken is. Zo ontwikkelen we een aantal algoritmen gebaseerd op lineair programmeren en kolomgeneratie. Deze aanpak met kolomgeneratie blijkt uitstekend geschikt voor de fractionele relaxering van energiebeperkte flow. Ook als heuristiek voor het eigenlijke, geheeltallige probleem blijkt de kolomgeneratie effectiever dan normaal geheeltallig programmeren.

Tot slot geven we twee algoritmen *met voordeel (with advantage)*, wat inhoudt dat batterijcapaciteiten met een bepaalde factor overschreden mogen worden. Voor ons flowprobleem kunnen zulke algoritmen worden omgevormd tot approximatiealgoritmen. We geven zo'n algoritme met voordeel voor algemene grafen. Dat algoritme verbetert niet de best bekende approximatiefactor, maar is wel simpeler en efficienter dan bekende algoritmen. Het tweede algoritme werkt op $st$-planaire grafen en geeft daar wel een verbetering van de best bekende approximatiefactor.

In Hoofdstuk 9 bespreken we ten slotte de mogelijkheden die dit proefschrift biedt voor verder onderzoek. De verkregen resultaten laten zien dat het onderzoek naar de structuur van draadloze netwerken alleen mogelijk is door geschikte modellering en algoritmische analyse, met technieken die varieren van netwerktheorie tot kansrekening en moderne complexiteitstheorie. We besluiten het proefschrift met een lijst van twaalf open problemen voor nadere analyse.

# Acknowledgments

So this is it, then: years of work, neatly condensed in book form. In an undertaking of this size, thanks are due to a great many people:

...to Jan van Leeuwen; a list of acknowledgments for this thesis can start with no one but him. As my doctoral advisor, his influence is found throughout the thesis. More than the term *advisor* expresses, however, I consider him—to borrow a German word—my *Doktorvater*: in a broader sense than just this thesis, he contributed greatly to my outlook on (computer) science and to my standards as a researcher and as a writer.

...to the colleagues down the hall, first in the Centrumgebouw Noord and later in the Buys Ballot Laboratorium: Marjan van den Akker, Guido Diepen, Han Hoogeveen, Bart Jansen, Stefan Kratsch, Johan Kwisthout, Eelko Penninkx, Wilke Schram, Richard Tan, and in particular...

...to Johan van Rooij and Jesper Nederlof. It was a great adventure to be involved in the initial developments of the *Inclusion-Exclusion meets Measure & Conquer* research. My contributions were limited to the first paper, but it was a pleasure to see up-close how that theory developed further.

...to Hans Bodlaender, Gerard Tel and Marinus Veldhorst, who deserve special mention for sparking my interest in algorithms when I was an undergraduate student and for helping me realise that I want to do research.

...to my new colleagues in Würzburg, particularly Alexander Wolff and Jan-Henrik Haunert for their patience while I finished this thesis.

...to all my coauthors: without exception it has been a pleasure to work with them.

...to the longtime friday night gang: Joeri Noort, Remco Okhuijsen, Erik van Ommeren and later also Karin Lemmers. We were going to make a computer game, remember? What happened to *that*? On a more serious note, particular thanks go to Erik for his interest in my research and for his enthusiasm for any tasty algorithmic puzzles I encountered.

...to my fellow *Code Monkeys*: Wouter Duivesteijn, Jan-Pieter van den Heuvel and Gwyneth Ouwehand. Both as an international programming contest team and as a cover band, we may not have been in it for the prize money, but we sure had a lot of fun.

...to the various people who contributed positively to my life but do not fit the above classification, including Joost van Dongen, Hugo Duivesteijn, Bas den Heijer, Olaf Jansen, Jessie Quinlan and Wouter Slob.

...and finally to Gerri and Chris, my parents, whose support over the years has been self-evident and ever present, to the extent that I might almost forget to mention them. May we all be so lucky.

Thomas van Dijk
Würzburg
-2014-

- Chapter 5 contains joint work with Tobias Müller.
- Chapter 6 contains joint work with Marjan van den Akker, Han Hoogeveen and Erik van Ommeren.
- Chapters 7 and 8 contain joint work with Marjan van van den Akker, Hans Bodlaender, Han Hoogeveen, Jan van Leeuwen, Erik van Ommeren, Richard Tan, Tim Toorop and Ruben van der Zwaan. Part of this research was published at the 11th Scandinavian Workshop on Algorithm Theory (SWAT) in 2008 [BTvDvL08] and the 24th Conference of the European Chapter on Combinatorial Optimization (ECCO) in 2011 [ADHT11].
- I am indebted to my advisor, Jan van Leeuwen, for many helpful comments and suggestions throughout this thesis.

# Curriculum Vitae

Thomas C. van Dijk was born on January 9th 1984 in Amsterdam, The Netherlands. In 2002, he received his gymnasium diploma from the Cals College in Nieuwegein. From 2002 to 2007, he studied computer science at Utrecht University, receiving a B.Sc. degree with a minor in software engineering, and a M.Sc. degree in the Applied Computing Science program. Both degrees were awarded with distinction. His master's thesis is entitled "Fixed parameter complexity of feedback set problems" and was supervised by Dr. Hans L. Bodlaender and Dr. Gerard Tel. In 2007, he started as a Ph.D. student in the Department of Information and Computing Sciences at Utrecht University under the supervision of Prof. Dr. Jan van Leeuwen.

Since 2012, he works at Würzburg University in the Efficient Algorithms chair of Prof. Dr. Alexander Wolff.

# List of publications

- M. Chimani, T. C. van Dijk, J.-H. Haunert. **How to Eat a Graph: Computing Selection Sequences for the Continuous Generalization of Road Networks.** In: *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS 2014).*
- B. Budig, T. C. van Dijk, A. Wolff. **Matching Labels and Markers in Historical Maps: an Algorithm with Interactive Postprocessing.** In: *Proceedings of the ACM SIGSPATIAL Workshop on MapInteraction (MapInteract 2014).*
- M. A. Bekos, T. C. van Dijk, M. Fink, P. Kindermann, S. Kobourov, S. Pupyrev, J. F. Spoerhase, A. Wolff. **Improved Approximation Algorithms for Box Contact Representations.** In: *Proceedings of the European Symposium on Algorithms (ESA 2014).*
- T. C. van Dijk, A. van Goethem, J.-H. Haunert, W. Meulemans and B. Speckmann. **Map Schematization with Circular Arcs.** In: *Proceedings of the International Conference on Geographic Information Science (GIScience 2014).*
- T. C. van Dijk, A. van Goethem, J.-H. Haunert, W. Meulemans and B. Speckmann. **An Automated Method for Circular-Arc Metro Maps.** At *Schematic Mapping 2014.* Best Student Submission award.
- J. Nederlof, J. M. M. van Rooij, T. C. van Dijk. **Inclusion/Exclusion Meets Measure and Conquer.** In: *Algorithmica*, 2014.
- T. C. van Dijk, J.-H. Haunert, A. van Goethem, W. Meulemans, B. Speckmann. **Accentuating Focus Maps via Partial Schematization.** In: *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS 2013).* Best Fast-forward Presentation and Runner-up Best Poster award.
- T. C. van Dijk, K. Fleszar, J.-H. Haunert, J. F. Spoerhase. **Road Segment Selection with Strokes and Stability.** In: *Proceedings of the ACM SIGSPATIAL Workshop on MapInteraction (MapInteract 2013).*
- T. C. van Dijk, J.-H. Haunert. **A Probabilistic Model for Road Selection in Mobile Maps.** In: *Web and Wireless GIS (W2GIS 2013).* Best Short Presentation award.

- J. M. van den Akker, T. C. van Dijk, J. A. Hoogeveen, and T. Toorop. **Optimizing Wireless Sensor Network Flow by Column Generation.** In: *Proceedings of the Conference of the European Chapter on Combinatorial Optimization (ECCO 2011).*
- H. L. Bodlaender, T. C. van Dijk. **A Cubic Kernel for Feedback Vertex Set and Loop Cutset.** In: *Theory of Computing Systems*, 2010.
- J. M. M. van Rooij, J. Nederlof, T. C. van Dijk. **Inclusion/Exclusion Meets Measure and Conquer.** In: *Proceedings of the European Symposium on Algorithms (ESA 2009).*
- H. L. Bodlaender, R. B. Tan, T. C. van Dijk, J. van Leeuwen. **Integer Maximum Flow in Wireless Sensor Networks with Energy Constraint.** In: *Proceedings of the Scandinavian Workshop on Algorithm Theory (SWAT 2008)*
- T. C. van Dijk. **Kernelization for Loop Cutset.** In: *Proceedings of the Belgian-Dutch Conference on Artificial Intelligence (BNAIC 2007).*

# Colophon

This thesis was typeset with $\LaTeX\,2_\varepsilon$.

Cover illustration: *“Reception zones”* by Chris van Dijk.

ISBN: 978-90-9028684-6