Matrix Rounding, Evolutionary Algorithms, and Hole Detection

Dissertation zur Erlangung des Grades
des Doktors der Ingenieurwissenschaften (Dr.-Ing.)
der Naturwisschaftlich-Technischen Fakultäten
der Universität des Saarlandes

vorgelegt von

Dipl.-Inf. Christian Klein

Saarbrücken
Dezember 2013

Tag des Kolloquiums:
23. Juni 2014

Dekan der Naturwissenschaftlich–Technischen Fakultät I:
Prof. Dr. Markus Bläser

Vorsitzender des Prüfungsausschusses:
Prof. Dr. Markus Bläser

Berichterstatter:
Prof. Dr. Benjamin Doerr
Prof. Dr. Dr. h.c. mult. Kurt Mehlhorn
Prof. Dr. Carsten Witt

Akademischer Mitarbeiter:
Dr.-Ing. Michael Kerber

Meinen Großeltern

Acknowledgements

My thesis advisor Benjamin Doerr has provided me with great support and guidance during my PhD studies. Doing research with Benjamin was very inspiring and he has helped me to discover and work on many interesting topics. I am also deeply grateful to Kurt Mehlhorn who gave me the opportunity to work in the inspiring research environment of his algorithms and complexity department at the Max-Planck-Institut für Informatik in Saarbrücken, Germany. To both Kurt and Benjamin I am also very thankful for their patience and encouragement. Stefan Funke also helped a great deal during my research and it has been a pleasure working with him.

My research profited greatly from working with many coauthors to which I am very thankful for their collaboration. In particular, I want to mention Benjamin Doerr, Tobias Friedrich, Stefan Funke, Edda Happ and Ralf Osbild for their permission to include our joint work in my thesis.

Furthermore I would like to thank all the colleagues I met during my time at the institute. You were a great group of people and made my time in Saarbrücken both very fruitful and enjoyable.

I also wish to thank Eric Furmanek for helping me with my English.

Finally, this thesis would not have been possible without Silke. Thank you for your love and for always being there for me!

Abstract

In this thesis we study three different topics from the field of algorithms and data structures. First, we investigate a problem from statistics. We give two randomised algorithms that can round matrices of fractional values to integer-valued matrices. These matrices will exhibit only small rounding errors for sums of initial row or column entries. Both algorithms also round each entry up with probability equal to its fractional value. We give a derandomisation of both algorithms.

Next, we consider the analysis of evolutionary algorithms (EAs). First, we analyse an EA for the Single Source Shortest Path problem. We give tight upper and lower bounds on the optimisation time of the EA. For this, we develop some new techniques for such analyses. We also analyse an EA for the All-Pairs Shortest Path problem. We show that adding crossover to this algorithm provably decreases its optimisation time. This is the first time that the usefulness of crossover has been shown for a non-constructed combinatorial problem.

Finally, we examine how to retrieve the implicit geometric information hidden in the communications graph of a wireless sensor network. We give an algorithm that is able to identify wireless nodes close to a hole in this network based on the connectivity information alone. If the input fulfils several preconditions, then the algorithm finds a node near each boundary point and never misclassifies a node.

Zusammenfassung

Diese Dissertation befasst sich mit drei Bereichen aus dem Gebiet der Algorithmen und Datenstrukturen. Zuerst betrachten wir ein Problem aus der Statistik. Wir präsentieren zwei randomisierte Algorithmen, die Matrizen von Kommazahlen in ganzzahlige Matrizen runden. Die berechneten Matrizen haben einen kleinen Rundungsfehler in allen Summen zusammenhängender Zeilen- oder Spaltenelemente. Außerdem ist die Wahrscheinlichkeit, dass eine Zahl aufgerundet wird, gleich ihrem Nachkommawert. Für beide Algorithmen zeigen wir derandomisierte Varianten.

Dann beschäftigen wir uns mit der Analyse Evolutionärer Algorithmen (EAs). Zuerst analysieren wir einen EA für das Single Source Shortest Path Problem. Wir zeigen scharfe obere und untere Schranken für die Optimierungszeit dieses EAs. Dazu entwickeln wir neue Techniken für solche Analysen. Außerdem untersuchen wir einen EA für das All-Pairs Shortest Path Problem. Wir zeigen, dass Rekombination die Optimierungszeit dieses EAs beweisbar beschleunigt. Dies ist das erste kombinatorische Problem, für das der Nutzen von Rekombination gezeigt werden konnte.

Abschließend untersuchen wir, wie man implizite geometrische Informationen im Kommunikationsgraphen eines Sensornetzes finden kann. Wir entwickeln einen Algorithmus, der Knoten nahe eines Loches im Netzwerk anhand der Konnektivitätsinformation identifizieren kann. Unter gewissen Bedingungen findet der Algorithmus einen Knoten nahe aller Randpunkte und markiert auch nur solche Knoten.

Contents

Chapter 1

Introduction

This introduction gives a short overview on the research I conducted during my PhD studies. I had the opportunity to work on three different topics during my time in the algorithms and complexity department at the Max-Planck-Institut für Informatik, namely randomised rounding, evolutionary algorithms and wireless sensor networks. For the sake of clarity, this thesis only contains the main results of my research. In the following overview, a short summary of each omitted publication is given after discussing the results from which this thesis was distilled.

Randomised Rounding

If a problem has several possible solutions for a given input, then the use of randomised algorithms might be of interest to ensure that the output is chosen randomly from all valid solutions. This is the case for the following problem that is motivated by a scenario from statistics. Given a table of statistical data, one wants to round the data entries to hide small entries that may otherwise allow the identification of single individuals. Obviously, there are several possible ways to round such a table. It is often desirable to use a randomised algorithm to ensure that the output is chosen such that each table entry is rounded up with probability equal to its fractional value. This ensures that the expected value of any sum of entries in the rounded table is equal to their sum in the original table. In statistics this is known as "unbiased rounding", while it is known as "randomised rounding" in computer science.

Of course, we want to ensure that a rounding of a table is still as close as possible to the original data. Especially, it is desirable that the rounding process does not change the row and column totals. Such roundings are known as "controlled roundings".

In Chapter 2 we give two algorithms to compute such unbiased controlled roundings. Furthermore, our algorithms will ensure that the rounding errors for

each sum over the first i elements of any row or column is less than one. Since many statistical attributes are linearly ordered, this allows to answer queries about a range of entries.

Our first algorithm works on tables where each entry is a binary number. If the fractional part of each number has at most ℓ bits, it will compute such a randomised rounding in time $O(mn\ell)$ if the table has $m \times n$ entries.

However, if one wants to round to multiples of ten, the numbers in the input might not have a finite binary representation. For this case we give a second algorithm that allows us to deal with arbitrary rational numbers. If all numbers have a common denominator q which has factorisation $q = \prod_{i=1}^{\ell} q_i$, then our algorithm will compute a rounding in expected time $O(mn \sum_{i=1}^{\ell} q_i^2)$.

For both algorithms we also give a derandomised variant. Although these variants will of course no longer compute an unbiased rounding, they have the same run time as their randomised counterparts.

The first algorithm is joint work with Benjamin Doerr, Tobias Friedrich and Ralf Osbild. It was published in [DFKO06b]. The algorithm for rational numbers was published in [DK06] and is joint work with Benjamin Doerr.

Further Work on Randomised Rounding

We also studied a simpler version of the rounding problem, which is however not discussed in this thesis. Again considering a real-valued matrix, we studied how to compute a rounding such that the rounding errors of all column sums and all initial row sums are less than one. In other words, we drop the constraint on the error for initial column sums. Another interpretation of this relaxed problem is that we are given a sequence of n-dimensional vectors which we have to round while not changing the norm of each of them. Hence, it is an extension of the classical problem of rounding a sequence of numbers to the n-dimensional case. This problem can be used to model various scheduling problems like the maximum deviation just-in-time scheduling problem, where balanced schedules on machines with negligible switch-over-costs are sought. In [DFKO06a], we show that this problem can be solved in linear time by a one-pass algorithm.

Evolutionary Algorithms

Randomness is also used as the main engine to power many meta-heuristics that can be used as generic frameworks to solve specific problems. A special variant of those meta-heuristics are so called “evolutionary algorithms”. These utilise an abstract form of the principles of evolution to tackle hard problems. A set (called population) of possible solutions (called individuals) is repeatedly modified by applying variation operators (i.e., mutation and crossover) and selection. Selection

is based on the fitness of each individual which is normally given by a so-called fitness function. While meta-heuristics can often be beaten by a problem-specific algorithm, they are easy to adapt to new problems. Designing a custom-tailored algorithm, on the other hand, often requires a thorough understanding of the underlying problem.

The theoretical analysis of the optimisation time[1] of such algorithms is still in its infancy. Also, the ramifications of using different variants of crossover, mutation and selection (or not using them at all) on the optimisation time are still not fully understood.

In Chapter 3 we discuss some of our contributions to the above questions. We give a tight analysis of the optimisation time of an evolutionary algorithm for the single source shortest path problem. The algorithm itself was proposed in earlier work by Scharnow, Tinnefeld and Wegener. However, they only give an upper bound on the optimisation time. We improve on their result to show a tight upper bound and a matching lower bound for this problem. In doing so, we develop several interesting techniques for the analysis of evolutionary algorithms. This analysis is joint work with Benjamin Doerr and Edda Happ and was published in [DHK07, DHK11].

Furthermore, we tackle the problem of whether crossover is a useful ingredient in evolutionary algorithms. While practitioners like to add a pinch of crossover to their algorithms, there have been very few theoretical justifications to do so. The only papers that showed that crossover might improve the optimisation time consider synthetic problems that are constructed to show just that. We give the first analysis of a non-artificial problem that profits from crossover. Namely, we will show that an evolutionary algorithm for the all pairs shortest path problem can be sped up by adding a crossover operator. This is joint work with Benjamin Doerr and Edda Happ and was published in [DHK08, DHK12].

Further Work on Evolutionary Algorithms

Apart from the two results discussed above, we published several more results on evolutionary algorithms that are not discussed in this thesis.

Together with Benjamin Doerr and Tobias Storch we gave a new representation for individuals in problems that have cyclic permutations as solutions. We analysed two variants of evolutionary algorithms for the Eulerian cycle problem based on this representation. Our analysis shows that, compared to previous solutions, the new representation indeed improves the optimisation time of both algorithms. This has been published in [DKS07].

[1] Roughly speaking, the optimisation time of an evolutionary algorithm is the number of times that variation and selection is applied until an optimal solution is found.

Furthermore we studied the effect of adding more fitness functions to a multi-objective optimisation problem. We analyse how such additional objectives can both improve or degrade the optimisation time. This result has been published in [BFH+07, BFH+09]. It is joint work with Dimo Brockhoff, Tobias Friedrich, Nils Hebbinghaus, Frank Neumann and Eckart Zitzler.

In [HJKN08] we analyse the effects of using different selection methods in evolutionary algorithms. For linear pseudo-Boolean functions it is known that elitist selection (always selecting the fittest individual) leads to an $O(n \log n)$ optimisation time. However, for fitness-proportional selection where each individual is selected with probability proportional to its fitness, we show that an exponential number of steps is needed to find the optimum. In particular, we can prove that all solutions generated by the algorithm during an exponential number of steps differ with high probability in linearly many bits from the optimal solution. This is joint work with Edda Happ, Daniel Johannsen and Frank Neumann.

Another interesting question arising in evolutionary computation is for which problems randomised local search (RLS) and the (1 + 1)-EA perform equivalently. In the classic bit-string model RLS flips one random bit per step while the (1 + 1)-EA flips every bit with probability $\frac{1}{n}$. Hence, in expectation, the (1 + 1)-EA behaves like RLS. It has long been known that for some problems the (1 + 1)-EA is exponentially faster than RLS, as the later can get stuck in local optima. The (1 + 1)-EA, however, seems to be at least as powerful as RLS. Indeed, the only known examples where RLS outperforms the (1 + 1)-EA only work if the start point lies in a very specific part of the search space. Most of the search space, however, consists of a deceptive area that leads to a local optimum that is far away from the global optimum. RLS, started in the right region, will follow a narrow path to the optimum without a chance to find the deceptive area. The (1 + 1)-EAon the other hand, can (and likely will) leave the path and enter the deceptive area, where it is lured to a local optimum. Of course this example has a fatal flaw. If RLS starts in the deceptive area it will be trapped forever, while the (1 + 1)-EA will, albeit after exponentially many steps, find the optimum. Hence, one might suppose that under reasonable conditions the (1 + 1)-EA will never be worse than RLS. In [DJK08] we show that this is not the case. We present an unimodal function (i.e., a function having no local optima) which RLS optimises in expected polynomial time while the (1 + 1)-EA needs a weakly exponential number of steps. Also, this example needs no artificial constraints on the start points. This is joint work with Thomas Jansen and Benjamin Doerr.

Wireless Sensor Networks

Finally, we study a computational geometry problem arising in the field of wireless sensor networks. Here, one typically has many very simple sensor nodes that

are randomly distributed over some area. To cover large areas, these nodes must be cheap and energy-efficient. Hence, they only have a very limited communications range and also lack a power-hungry GPS unit.

A major challenge is to compute the topology of such networks. Due to the lack of absolute positioning information, the only information available is the communication graph of the underlying network. In Chapter 4 we give a linear-time algorithm that can identify the boundary of holes in the communication graph from connectivity information alone. The algorithm can be implemented in a distributed and localised way. Also, our algorithm does not need the nodes to be uniformly distributed but works with arbitrary node distributions. We prove the correctness of our algorithm (i.e. that it only finds nodes close to a boundary and that it finds a node close to each boundary point) for well-formed holes and a sufficiently high node density. While this theoretical analysis turns out to be quite conservative, our actual implementation of the algorithm shows that it works well under less stringent conditions.

This result is joint work with Stefan Funke and has been published in [FK06].

Chapter 2

Matrix Rounding

Rounding an $m \times n$ matrix while keeping the errors in all rows, columns and the whole matrix small is a classical problem. It has applications in hypergraph colouring, in scheduling and in statistics.

In this Chapter, we present two algorithms that round a real matrix to an integer matrix. The roundings are such that the rounding errors in all initial intervals of rows and columns, as well as in the whole matrix, are less than one.

Often, it is also desirable to round each entry randomly such that the probability of rounding it up equals its fractional part. This is known as unbiased rounding in statistics and as randomised rounding in computer science. Both our algorithms compute an unbiased rounding.

We present a highly efficient algorithm to compute rounding in Section 2.3. This algorithm will work for matrices where each number has a finite binary representation. It can compute such roundings in time $O(mn \log(mn))$.

However, the input may not always consist of such numbers. Especially in statistics, one may want to round fractional numbers where the denominators are multiples of 3 or 10. In Section 2.4, we show how to compute unbiased roundings for matrices of arbitrary rational numbers. The algorithm needs expected time $O(mnq^2)$, where q is the common denominator of the matrix entries. We also show that, if the denominator can be written as $q = \prod_{i=1}^{\ell} q_i$ for some integers q_i, the expected runtime can be reduced to $O(mn \sum_{i=1}^{\ell} q_i^2)$. Our algorithm can be derandomised efficiently using the method of conditional probabilities.

Finally, we give a lower bound on the error in arbitrary intervals in Section 2.5.

Section 2.3 is based on joint work with Benjamin Doerr, Tobias Friedrich and Ralf Osbild published in [DFKO06b]. Section 2.4 is based on joint work with Benjamin Doerr published in [DK06]. The lower bound given in Section 2.5 is based on joint work with Benjamin Doerr, Tobias Friedrich and Ralf Osbild published in [DFKO06a].

2.1 Introduction

In this Chapter, we analyse a rounding problem with connections to different areas in discrete mathematics, computer science and statistics. We give several algorithms that efficiently round a matrix to an integer one in such a way that rounding errors in intervals of rows and columns as well as in the whole matrix are small. Furthermore, those algorithms will compute a so called randomised or unbiased rounding. This means that each matrix entry is rounded up with probability equal to its fractional value.

This result extends the famous rounding Lemma of Baranyai [Bar75] and several results on controlled rounding in statistics to small rounding errors in all initial segments of rows and columns. While the existence of such roundings can also be derived from Knuth's two-way rounding result [Knu95] as well as from the work of Asano, Katoh, Obokata and Tokuyama [AKOT03] on rounding 2-laminar systems, both of these works use a formulation as flow problem and thus are less efficient than our approach.

2.1.1 Our Contribution

We now state the main theorems of this chapter. For this we introduce the following notation. Let $a, b \in \mathbb{R}$ be real numbers. Then we define $[a, \dots, b] := \{z \in \mathbb{Z} \mid a \leq z \leq b\}$. For $x \in \mathbb{R}$ we define $\lfloor x \rfloor := \max\{z \in \mathbb{Z} \mid z \leq r\}, \lceil x \rceil := \min\{z \in \mathbb{Z} \mid z \geq r\}$ and $\{x\} := x - \lfloor x \rfloor$. For $q \in \mathbb{N}$ let $\frac{1}{q}\mathbb{Z} := \{\frac{p}{q} \mid p \in \mathbb{Z}\}$.

The algorithms presented by us compute controlled roundings which fulfil the additional constraint that all rounding errors in initial row and column intervals are less than one. This is formalised by the following definition.

Definition 2.1 (Locally Consistent Controlled Rounding)**.** *Let $X \in \mathbb{R}^{m \times n}$ be a real-valued matrix. A matrix $Y \in \mathbb{Z}^{m \times n}$ is called* locally consistent controlled rounding *if*

$$\forall b \in [1, \dots, n],\ i \in [1, \dots, m] : \left| \sum_{j=1}^{b} (x_{ij} - y_{ij}) \right| < 1, \tag{2.1}$$

$$\forall b \in [1, \dots, m],\ j \in [1, \dots, n] : \left| \sum_{i=1}^{b} (x_{ij} - y_{ij}) \right| < 1, \tag{2.2}$$

$$\left| \sum_{i=1}^{m} \sum_{j=1}^{n} (x_{ij} - y_{ij}) \right| < 1 \tag{2.3}$$

holds.

The randomised variants of our algorithms have the additional property that each matrix entry is rounded up with probability equal to its fractional value. This is known as *randomised rounding* in computer science [Rag88] and as *unbiased rounding* in statistics [Cox87, Fel75]. Here, a controlled rounding is computed such that the expected values of each matrix entry (including the totals) equals its fractional value in the original matrix.

Definition 2.2 (Randomised Rounding)**.** *Let $x \in \mathbb{R}$. A random variable y is called* randomised rounding *of x, denoted by $x \approx y$, if*

$$\begin{aligned} \Pr(y = \lfloor x \rfloor + 1) &= \{x\} \\ \Pr(y = \lfloor x \rfloor) &= 1 - \{x\}. \end{aligned}$$

For a matrix $X \in \mathbb{R}^{m \times n}$, we call a $\mathbb{Z}^{m \times n}$-valued random variable Y randomised rounding *or* unbiased rounding *of X if*

$$\forall i \in [1, \ldots, m]\,,\ j \in [1, \ldots, n] : x_{ij} \approx y_{ij}$$

holds.

Note that if $x \approx y$, then $\Pr(|y - x| < 1) = 1$ and $\mathbb{E}(y) = x$. In fact, the converse holds as well. The randomised algorithms discussed in this chapter will adhere to the following definition.

Definition 2.3 (Randomised Locally Consistent Controlled Rounding)**.** *Let $X \in \mathbb{R}^{m \times n}$ be a real-valued matrix. We call an integer-valued random variable $Y \in \mathbb{Z}^{m \times n}$* randomised locally consistent controlled rounding *or* unbiased locally consistent controlled rounding *of X if*

$$\begin{aligned} \forall b \in [1, \ldots, n]\,,\ i \in [1, \ldots, m] : \sum_{j=1}^{b} x_{ij} &\approx \sum_{j=1}^{b} y_{ij}, \\ \forall b \in [1, \ldots, m]\,,\ j \in [1, \ldots, n] : \sum_{i=1}^{b} x_{ij} &\approx \sum_{i=1}^{b} y_{ij}, \\ \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} &\approx \sum_{i=1}^{m} \sum_{j=1}^{n} y_{ij} \end{aligned}$$

holds.

With these definitions we can now characterise our algorithms. Their behaviour is summarised by the following four theorems.

Theorem 2.1. *Let $X \in \mathbb{R}^{m\times n}$ be a real-valued matrix. Assume that the fractional part of each matrix entry has binary length at most ℓ. Then, a randomised locally consistent controlled rounding $Y \in \mathbb{Z}^{m\times n}$ of X can be computed in time $O(mn\ell)$.*

The algorithm that yields this theorem can also be modified to calculate an, albeit not randomised, rounding of arbitrary matrices.

Theorem 2.2. *For all $X \in \mathbb{R}^{m\times n}$, a locally consistent controlled rounding $Y \in \mathbb{Z}^{m\times n}$ can be computed in time $O(mn \log(mn))$.*

Note that Theorem 2.1 only works for matrices of numbers with finite binary expansion. Hence, it cannot round non-binary fractions, as is often required in applications. We will give another algorithm to compute randomised roundings of such matrices. For this algorithm, the following variant of Theorem 2.1 holds.

Theorem 2.3. *For all $X \in \frac{1}{q}\mathbb{Z}^{m\times n}$, a randomised locally consistent controlled rounding $Y \in \mathbb{Z}^{m\times n}$ can be computed in expected time $O(mn \sum_{i=1}^{\ell} p_i^2)$, where $q = \prod_{i=1}^{\ell} p_i$, $p_i \in \mathbb{N}$ is a factorisation of q.*

The result above can be derandomised using the method of conditional probabilities. This leads to a deterministic algorithm having the same asymptotic runtime.

Theorem 2.4. *For all $X \in \frac{1}{q}\mathbb{Z}^{m\times n}$, a locally consistent controlled rounding $Y \in \mathbb{Z}^{m\times n}$ can be computed in time $O(mn \sum_{i=1}^{\ell} p_i^2)$, where $q = \prod_{i=1}^{\ell} p_i$, $p_i \in \mathbb{N}$ is a factorisation of q.*

A Lower Bound

We also present a non-trivial lower bound for the error in arbitrary intervals. Earlier works only regarded errors in initial intervals $[1, \ldots, t]$. For upper bounds, a triangle inequality argument extends any upper bound for initial intervals to twice this bound for arbitrary intervals. For lower bounds, things are more complicated. In particular, an example of Brauner and Crama [BC04] showing a lower bound of $1 - \frac{1}{m}$ for initial intervals yields no better bound for arbitrary intervals. We present a family of $3 \times n$-matrices such that any rounding contains a non-initial interval having an error of at least $1.5 - \varepsilon$.

As a corollary, this also yields an error of $0.75 - \varepsilon$ for initial intervals of $3 \times n$-matrices.

2.1.2 Related Work

Baranyai's Rounding Lemma and Applications in Statistics

Baranyai [Bar75] used a weaker version of Theorem 2.2 to obtain his famous results on colouring and partitioning complete uniform hypergraphs. He showed that any matrix can be rounded in a way that the errors in all rows, all columns and the whole matrix are less than one. In other words, the entries of the original matrix are rounded such that the row sums, column sums and the total matrix sum of the rounded matrix differ by less than one from the corresponding sums in the unrounded matrix. Baranyai used a formulation as flow problem to prove this statement. This yields a worse runtime than the bound in Theorem 2.2. However, algorithmic issues were not the focus of his work.

With statistics applications in mind, Baranyai's result was independently obtained by Causey, Cox and Ernst [CCE85] and, in a slightly weaker form, by Bacharach [Bac66]. In statistics, there are two applications for such rounding results (cf. [CE82]). Note first that instead of rounding to integers, our results also apply to rounding to multiples of any other base (e.g., whole multiples of one percent). This can be used in statistics to improve the readability of data tables. A second reason to apply such rounding procedures is confidentiality protection [WW01]. Frequency counts that directly or indirectly disclose small counts may permit the identification of individual respondents. This risk can be reduced by rounding all numbers to multiples of a small integer, e.g., 10. However, in both applications one wants to keep rounding errors in columns and rows small. This allows to use the rounded matrix to obtain information on the row and column totals.

Our result allows to retrieve further reliable information from the rounded matrix, namely also on the sums of consecutive elements in rows or columns. Such queries may occur if there is a linear ordering on statistical attributes. Here is an example. Let x_{ij} be the number of people in country i that are j years old. Say Y is such that $\frac{1}{1000}Y$ is a rounding of $\frac{1}{1000}X$ as in Theorem 2.3. Now $\sum_{j=20}^{40} y_{ij}$ is the number of people in country i that are between 20 to 40 years old, apart from an error of less than 2000. Note that such guarantees are not provided by the results of Baranyai [Bar75], Bacharach [Bac66] and Causey, Cox and Ernst [CCE85].

Also, our result is highly efficient. Baranyai, who was not interested in algorithmic issues, as well as both Bacharach and Causey, Cox and Ernst used a reduction of the rounding problem to a flow or transportation problem. Though such problems can be solved relatively efficiently, our almost linear time solution clearly beats their runtimes.

There exist various extensions to the controlled rounding problem. Kelly, Assad and Golden [KAG90] show that three-dimensional controlled rounding is

NP-hard. They also show that such a rounding does not always exist. Various heuristics to compute controlled roundings of linked or multi-dimensional tables have been studied (cf. [KGAB90, RGK97, SGLY+04]). Hell, Kirkpatrick and Li [HKL96] studied the problem of rounding symmetric matrices in such a way that symmetry is preserved.

Knuth's Two-way Rounding

Knuth [Knu95] showed how to round a sequence $x_1, \ldots, x_n \in \mathbb{R}$ of n real numbers to $y_i \in \{\lfloor x_i \rfloor, \lceil x_i \rceil\}$ such that for two given permutations $\sigma_1, \sigma_2 \in S_n$, we have $|\sum_{i=1}^{k}(x_{\sigma_1(i)} - y_{\sigma_1(i)})| \leq \frac{n}{n+1}$ and $|\sum_{i=1}^{k}(x_{\sigma_2(i)} - y_{\sigma_2(i)})| \leq \frac{n}{n+1}$ for all k. Knuth's proof uses integer flows in a certain network [FF62]. On account of this his worst-case runtime is super-linear.

One application mentioned in [Knu95] is that of matrix rounding. For this, first assume that the input matrix has integral row and column sums. As we will see later this is no real restriction. Then apply Knuth's algorithm using a permutation that enumerates the x_{ij} row by row, and a permutation that enumerates the x_{ij} column by column. Obviously, the rounding errors in all initial intervals of the first row and column will be smaller than one. Because of the integrality of the row and column sums the error occurring after rounding any number of rows or columns will be zero, hence all initial intervals of any row or column will exhibit an error smaller than one.

Rounding 2-Laminar Families and Halftoning

Motivated by the digital halftoning problem from image processing, Asano, Katoh, Obokata and Tokuyama [AKOT03] showed that rounding with small errors with respect to a 2-laminar family of sets is always possible. A family $\mathcal{S}$ of sets is called *laminar*, if $S_1 \subseteq S_2$ or $S_1 \cap S_2 = \emptyset$ holds for all $S_1, S_2 \in \mathcal{S}$. It is called 2-laminar, if it is the union of two laminar families of sets.

Asano et al. show that a 2-laminar set system is unimodular. A set system (i.e. a hypergraph) is called unimodular if it has a totally unimodular incidence matrix, which means that the determinant of each square submatrix is $-1, 0$, or 1. In consequence, if $\mathcal{S}$ is 2-laminar, then any family of numbers $x_1, \ldots, x_n$ can be rounded to integers $y_1, \ldots, y_n$ such that $|\sum_{s\in S} x_s - \sum_{s\in S} y_s| < 1$ for all $S \in \mathcal{S}$. They also show that such a rounding problem can be solved in time $O(n^2 \log^3(n))$ via convex-cost flow algorithms. Note that in general unimodular set systems admit roundings with all errors at most $1 - \frac{1}{n+1}$. This was shown in [Doe04] and [BH04], but both works don't seem to yield more efficient solutions for 2-laminar rounding problems.

Since the set of all initial segments of rows and columns of a two-dimensional grid is a 2-laminar family, this result yields the existential statement of Theorem 2.2. However, the run-time of this approach is larger by more than a linear factor. Also, our approach only uses elementary combinatorial arguments.

2.2 Preliminaries

2.2.1 Simple Extensions

We now provide an easy extension of the results stated in the introduction. Also, we will show that it suffices if we can compute roundings of matrices having integral row and column sums. This will help us simplify our main proofs.

We immediately obtain rounding errors of less than two in arbitrary intervals in rows and columns. This is supplied by the following lemma.

Lemma 2.1. *Let Y be a rounding of a matrix X such that the errors $|\sum_{j=1}^{b}(x_{ij}-y_{ij})|$, $i \in [1,\dots,m]$, $b \in [1,\dots,n]$, in all initial intervals of rows are at most d. Then the errors in arbitrary intervals of rows are at most $2d$. That is, for all $i \in [1,\dots,m]$ and all $1 \le a \le b \le n$ it holds that*

$$\left|\sum_{j=a}^{b}(x_{ij}-y_{ij})\right| \le 2d.$$

This also holds for column intervals, i.e., if the errors $|\sum_{i=1}^{b}(x_{ij}-y_{ij})|$ in all initial intervals of columns are at most d', then the errors $|\sum_{i=a}^{b}(x_{ij}-y_{ij})|$ in arbitrary intervals of columns are at most $2d'$.

Proof. Let $i \in [1,\dots,m]$ and $1 \le a \le b \le n$. Then

$$\begin{aligned}\left|\sum_{j=a}^{b}(x_{ij}-y_{ij})\right| &= \left|\sum_{j=1}^{b}(x_{ij}-y_{ij}) - \sum_{j=1}^{a-1}(x_{ij}-y_{ij})\right| \\ &\le \left|\sum_{j=1}^{b}(x_{ij}-y_{ij})\right| + \left|\sum_{j=1}^{a-1}(x_{ij}-y_{ij})\right| \le 2d.\end{aligned}$$

□

The following lemma shows how computing roundings for matrices having arbitrary row and column sums can be reduced to computing roundings of matrices having integral row and column sums.

Lemma 2.2. *Assume that for any $X \in \mathbb{R}^{m\times n}$ with integral column and row sums a locally consistent controlled rounding $Y \in \mathbb{Z}^{m\times n}$ can be computed in time $T(m,n)$. Then for all $\tilde{X} \in \mathbb{R}^{m\times n}$ with arbitrary column and row sums a locally consistent controlled rounding $\tilde{Y} \in \mathbb{Z}^{m\times n}$ can be computed in time $T(m+1,n+1) + O(mn)$.*

Proof. Given an arbitrary matrix $\tilde{X} \in \mathbb{R}^{m\times n}$, we add an extra row taking what is missing towards integral column sums and add an extra column taking what is missing towards integral row sums. Hence, let $X \in \mathbb{R}^{(m+1)\times(n+1)}$ be such that

$$\begin{aligned}
x_{ij} &= \tilde{x}_{ij} && \text{for all } i \in [1,\dots,m]\,,\ j \in [1,\dots,n]\,,\\
x_{m+1,j} &= \left\lceil \sum_{i=1}^{m} \tilde{x}_{ij} \right\rceil - \sum_{i=1}^{m} \tilde{x}_{ij} && \text{for all } j \in [1,\dots,n]\\
x_{i,n+1} &= \left\lceil \sum_{j=1}^{n} \tilde{x}_{ij} \right\rceil - \sum_{j=1}^{n} \tilde{x}_{ij} && \text{for all } i \in [1,\dots,m+1]\,.
\end{aligned}$$

Clearly, X has integral row and column sums. Therefore, it can be rounded to a locally consistent controlled rounding $Y \in \mathbb{Z}^{(m+1)\times(n+1)}$ in time $T(m+1,n+1)$.

Observe that for any $X \in \mathbb{R}^{(m+1)\times(n+1)}$ with integral row and column sums the total matrix error is zero. This stems from the fact that the total error for $\tilde{X}$ is the rounding error of the lower right entry of X, as the following calculation shows.

$$\begin{aligned}
\left|\sum_{i=1}^{m}\sum_{j=1}^{n}(x_{ij}-y_{ij})\right| &= \Bigg|\sum_{i=1}^{m+1}\sum_{j=1}^{n+1}(x_{ij}-y_{ij}) - \sum_{i=1}^{m+1}(x_{i,n+1}-y_{i,n+1})\\
&\qquad - \sum_{j=1}^{n+1}(x_{m+1,j}-y_{m+1,j}) + (x_{m+1,n+1}-y_{m+1,n+1})\Bigg|\\
&= \left|0-0-0+(x_{m+1,n+1}-y_{m+1,n+1})\right|\\
&\le |x_{m+1,n+1}-y_{m+1,n+1}| < 1.
\end{aligned}$$

The above calculation uses the fact that the row and column sums of X are integral and hence the rounding error for a whole row or a whole column must be zero. Since we remove both the last row and the last column from X to get $\tilde{X}$, we have to add $(x_{m+1,n+1} - y_{m+1,n+1})$ again as it was subtracted twice. This is the only part of the sum which is non-zero. But by definition of a rounding, the error in a single entry must be less than 1.

By setting $\tilde{y}_{ij} = y_{ij}$ for all $i \in [1,\dots,m]$ and $j \in [1,\dots,n]$, we obtain the desired rounding $\tilde{Y} \in \mathbb{Z}^{m\times n}$. □

2.2.2 Random Walks

We need some well known facts about one-dimensional random walks with absorbing barriers. These are random processes that take random steps back and forward along a discrete line, until the first or last point of the line is reached. More precisely, consider a set of $n+1$ vertices labelled v_0 to v_n. From vertex $v_i, i \in [1, \ldots, n-1]$, one can either take a step to vertex v_{i+1} or v_{i-1}, both with probability $\frac{1}{2}$. After reaching one of the endpoints v_0 or v_n the process stops and no further steps can be taken.

We are interested in the probability $\Pr(v_i \nearrow v_n)$ that a random walk starting from vertex v_i will reach v_n instead of v_0.

Lemma 2.3. $\Pr(v_i \nearrow v_n) = \frac{i}{n}$.

Proof. From the definition of random walks we obtain the following system of linear equations

$$\begin{aligned}
\Pr(v_n \nearrow v_n) &= 1 \\
\Pr(v_{n-1} \nearrow v_n) &= \frac{1}{2}\Pr(v_{n-2} \nearrow v_n) + \frac{1}{2}\Pr(v_n \nearrow v_n) \\
&\cdots \\
\Pr(v_1 \nearrow v_n) &= \frac{1}{2}\Pr(v_0 \nearrow v_n) + \frac{1}{2}\Pr(v_2 \nearrow v_n) \\
\Pr(v_0 \nearrow v_n) &= 0.
\end{aligned}$$

It can easily be checked that this system has the unique solution $\Pr(v_i \nearrow v_n) = \frac{i}{n}$. □

From this Lemma it follows immediately that

$$\Pr(v_i \nearrow v_0) = 1 - \Pr(v_i \nearrow v_n) = \frac{n-i}{n}.$$

Next, we consider the expected number of steps $\mathbb{E}(\text{Steps}(v_i))$ that a random walk starting in vertex v_i needs to reach either vertex v_0 or v_n.

Lemma 2.4. $\mathbb{E}(\text{Steps}(v_i)) = i(n-i)$.

Proof. The random walk needs one step to reach either v_{i-1} or v_{i+1} from vertex v_i for $i \in [1, \ldots, n-1]$. Since both vertices are reached with equal probability $\frac{1}{2}$, we get the following system of linear equations:

$$\begin{aligned}
\mathbb{E}(\text{Steps}(v_0)) &= \mathbb{E}(\text{Steps}(v_n)) = 0 \\
\mathbb{E}(\text{Steps}(v_i)) &= 1 + \frac{1}{2}\mathbb{E}(\text{Steps}(v_{i-1})) + \frac{1}{2}\mathbb{E}(\text{Steps}(v_{i+1})), \qquad i \in [1, \ldots, n-1]\,.
\end{aligned}$$

Again, it is easy to check that this system has the unique solution $\mathbb{E}(\text{Steps}(v_i)) = i(n-i)$. □

2.3 Bit-wise Rounding

We now present an algorithm to compute locally consistent controlled roundings. For this, we first study a simpler problem, namely that of computing locally consistent controlled roundings of half-integral matrices. We call a matrix X half-integral, if it only has entries 0 or $\frac{1}{2}$, i.e. if $X \in \{0, \frac{1}{2}\}^{m \times n}$ holds.

For such matrices our rounding problem turns out to be much simpler. In fact, it can be solved in linear time.

2.3.1 Half-Integral Rounding

We will now consider the rounding problem for half-integral matrices. According to Lemma 2.2 it suffices to consider matrices with integral row and column sums.

Here is an outline of our approach. For each row and column, we consider the sequence of its $\frac{1}{2}$-entries and partition them into disjoint pairs of neighbours. From the two $\frac{1}{2}$s forming such a pair, exactly one is rounded to 1 and the other to 0. Thus, if such a pair is fully contained in an initial interval, it does not contribute to the rounding error.

To make the idea precise, assume some row contains exactly $2K$ entries of value $\frac{1}{2}$. For each $k \in [1, \dots, K]$ we call the $(2k-1)$-th and $(2k)$-th $\frac{1}{2}$-entry of this row a *row pair*. The $\frac{1}{2}$s of a row pair are mutually referred to as *row neighbours*. Similarly, we define *column pairs* and *column neighbours*. Figure 2.1(a) shows a half-integral matrix together with row and column pairs indicated by boxes.

Our solution makes use of an auxiliary graph $\mathcal{G}_X$ which contains the necessary information about row and column neighbours. Each $\frac{1}{2}$-entry is represented by a vertex that is labelled with the corresponding matrix indices. Each pair is represented by an edge connecting the vertices that correspond to the paired $\frac{1}{2}$s. Figure 2.1(b) shows the auxiliary graph that belongs to the matrix of Figure 2.1(a).

We collect some properties of this auxiliary graph.

Lemma 2.5. *Let $X \in \{0, \frac{1}{2}\}^{m \times n}$ be a matrix with integral row and column sums.*

a) Every vertex of $\mathcal{G}_X$ has degree 2.

b) $\mathcal{G}_X$ is a disjoint union of even cycles.

c) $\mathcal{G}_X$ is bipartite.

Proof. a) Because of the integrality of the row and column sums, the number of $\frac{1}{2}$-entries in each row and column is even. Hence each $\frac{1}{2}$-entry has a row and a column neighbour. In consequence, each vertex is incident with exactly two edges.

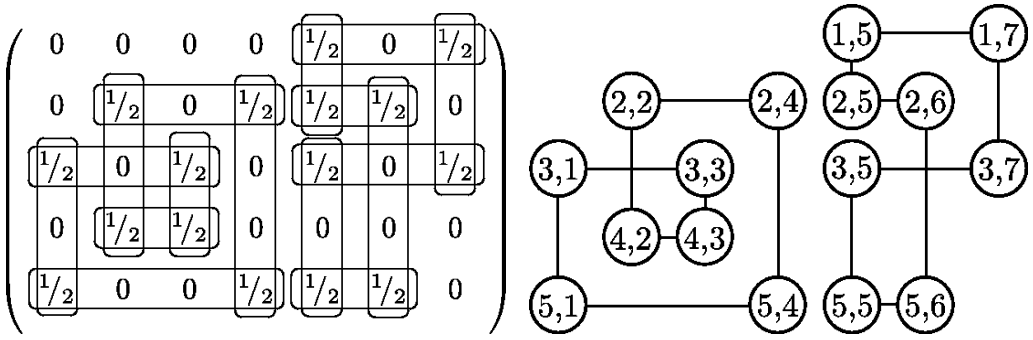

Figure 2.1: Example for the construction of an auxiliary graph. (a) Input matrix X with its row and column pairs. (b) Auxiliary graph $\mathcal{G}_X$. Vertices are labelled with matrix indices and edges connect vertices of row and column pairs.

b) The edge sequence of a path in $\mathcal{G}_X$ corresponds to an alternating sequence of row and column pairs. Therefore any cycle in $\mathcal{G}_X$ consists of an even number of edges. Since each vertex has degree two, $\mathcal{G}_X$ is a disjoint union of cycles.

c) Clearly, every even cycle is bipartite. □

With this result, we are able to find the desired roundings.

Lemma 2.6. *Let $X \in \{0, \frac{1}{2}\}^{m\times n}$ and let $V_0 \dot\cup V_1$ be a bipartition of $\mathcal{G}_X$. Define $Y = (y_{ij}) \in \{0,1\}^{m\times n}$ by*

$$y_{ij} = \begin{cases} 0, & \text{if } x_{ij} = 0 \\ 0, & \text{if } x_{ij} = \frac{1}{2} \text{ and } (i,j) \in V_0 \\ 1, & \text{if } x_{ij} = \frac{1}{2} \text{ and } (i,j) \in V_1. \end{cases}$$

Then Y has the property that

$$\forall b \in [1,\dots,n],\ i \in [1,\dots,m] : \left|\sum_{j=1}^{b}(x_{ij} - y_{ij})\right| \le \frac{1}{2}, \tag{2.4}$$

$$\forall b \in [1,\dots,m],\ j \in [1,\dots,n] : \left|\sum_{i=1}^{b}(x_{ij} - y_{ij})\right| \le \frac{1}{2}. \tag{2.5}$$

Proof. Because 0s of X are maintained in Y, it suffices to consider $\frac{1}{2}$-entries to determine the rounding error in initial intervals. Since the rounded values for the $(2k-1)$-th and $(2k)$-th $\frac{1}{2}$-entry sum up to 1 by construction, there is no error in initial intervals that contain an even number of $\frac{1}{2}$s, and an error of $\frac{1}{2}$ if they contain an odd number of $\frac{1}{2}$s. □

After these considerations, we are able to present an algorithm that solves the problem in two steps. First we compute the auxiliary graph and afterwards the output matrix. To construct $\mathcal{G}_X$, we transform the input matrix X column by column from left to right. Generating the labelled vertices is trivial. The column neighbours can be detected by just numbering the $\frac{1}{2}$-entries within a column from top to bottom. Assume there are $2k$ such entries. Then we insert an edge between the vertices with number $2i - 1$ and $2i$ for $i \in [1, \dots, k]$. The strategy to detect row neighbours is the same but we need more information. Therefore we store for each row the parity of its $\frac{1}{2}$-entries so far and, if the parity is odd, further a pointer to the last occurrence of $\frac{1}{2}$ in this row. Then, if the current $\frac{1}{2}$ is an even occurrence, we have a pointer to the preceding $\frac{1}{2}$, and are able to insert an edge between the corresponding vertices in $\mathcal{G}_X$.

The output matrix Y can be computed from X as follows. Every 0 in X is kept and every $\frac{1}{2}$-sequence that corresponds to a cycle in $\mathcal{G}_X$ is substituted by an alternating 0-1-sequence. By Lemma 2.5, this is always possible. It does not matter which of the two alternating 0-1 sequences we choose.

The graph $\mathcal{G}_X$ can be realised with adjacency lists (the vertex degree is always 2). The additional information per row can be realised by a simple pointer-array of length m (a special nil-value indicates even parity).

Since the runtime of each step is bounded by the size of the input matrix, the entire algorithm takes time $O(mn)$. In addition to the constant amount of data we store for each of the m rows, we store all k entries of value $\frac{1}{2}$ in the auxiliary graph. This leads to a total space consumption of $O(m + k)$.

Summarising the above, we obtain the following lemma.

Lemma 2.7. *Let $X \in \{0, \frac{1}{2}\}^{m \times n}$. Then a rounding $Y \in \{0, 1\}^{m \times n}$ satisfying the inequalities* (2.4) *and* (2.5) *from Lemma 2.6 can be computed in time $O(mn)$.*

2.3.2 Binary Rounding

We now discuss how we can use the algorithm for half-integral matrices discussed in the last section to compute roundings of generic matrices. The following rounding method was introduced by Beck and Spencer [BS84] in 1984. They used it to prove the existence of two-colourings of $\mathbb{N}$ having small discrepancy in all arithmetic progressions of arbitrary length and bounded difference.

Assume that we have some numbers in $[0, 1]$ which have to be rounded. Assume that they all have a finite binary expansion. Then we repeat rounding the last digit (to zero or twice its value) and thus reduce the length of the binary expansion until we obtain numbers in $\{0, 1\}$. To round a single digit, we only need to understand the corresponding rounding problem for half-integral numbers. The finally

resulting rounding errors are at most twice the ones incurred by the half-integral roundings.

If some numbers do not have a finite binary expansion, one can use a sufficiently large finite length approximation. To get rid of additional errors caused by this, we invoke a slight refinement of the binary rounding method. In [Doe04] it was proven that the extra factor of two can be reduced to an extra factor of $2(1 - \frac{1}{2r})$, where r is the number of rounding errors we want to keep small.

In our setting, the number of rounding errors is the number of all initial row and column intervals, i.e., $r = 2mn$.

Lemma 2.8. *Assume that for any $X \in \{0, \frac{1}{2}\}^{m\times n}$ a rounding $Y \in \{0,1\}^{m\times n}$ can be computed in time T (which is $\Omega(mn)$) that satisfies*

$$\forall b \in [1,\dots,n],\ i \in [1,\dots,m] :\ \left|\sum_{j=1}^{b}(x_{ij} - y_{ij})\right| \le D,$$

$$\forall b \in [1,\dots,m],\ j \in [1,\dots,n] :\ \left|\sum_{i=1}^{b}(x_{ij} - y_{ij})\right| \le D.$$

Then for all $\ell \in \mathbb{N}$ and $X \in [0,1)^{m\times n}$ a rounding $Y \in \{0,1\}^{m\times n}$ such that

$$\forall b \in [1,\dots,n],\ i \in [1,\dots,m] :\ \left|\sum_{j=1}^{b}(x_{ij} - y_{ij})\right| \le 2(1 - \frac{1}{4mn})D + 2^{-\ell}b,$$

$$\forall b \in [1,\dots,m],\ j \in [1,\dots,n] :\ \left|\sum_{i=1}^{b}(x_{ij} - y_{ij})\right| \le 2(1 - \frac{1}{4mn})D + 2^{-\ell}b$$

can be computed in time $O(\ell\, T)$.

To give an outline of the underlying algorithm we now give a brief sketch of the main ideas found in [Doe00].

Let $X \in [0,1)^{m\times n}$. Assume first that all entries of $X := X^{(\ell)}$ have binary length ℓ, that is, $2^\ell X$ is an integral matrix. Let $\ell_0 := \lfloor \log_2(2mn) \rfloor + 1$.

Having defined $X^{(i)}$ with binary length i for some $i > 1$, we obtain $X^{(i-1)}$ as follows. Let $X^{(i)} = X' + 2^{-i+1}X''$ with $X' \in [0,1]^{m\times n}$ having binary length $i-1$ and $X'' \in \{0, \frac{1}{2}\}^{m\times n}$. Using the assumptions of the lemma, we compute a rounding $Y'' \in \{0,1\}^{m\times n}$ of X'' such that

$$\forall b \in [1,\dots,n],\ i \in [1,\dots,m] :\ \left|\sum_{j=1}^{b}(x''_{ij} - y''_{ij})\right| \le D, \tag{2.6}$$

$$\forall b \in [1,\dots,m],\ j \in [1,\dots,n] :\ \left|\sum_{i=1}^{b}(x''_{ij} - y''_{ij})\right| \le D. \tag{2.7}$$

This takes time T. Let $X^{(i-1)} = X^{(i)} + 2^{-i+1}(Y'' - X'')$. Then $X^{(i-1)}$ has binary length $i-1$. In particular, $Y := X^{(0)}$ is a rounding of $X = X^{(\ell)}$. Note that by construction, $Y - X = \sum_{i=1}^{\ell} 2^{-i+1}(X^{(i-1)} - X^{(i)})$. In consequence we can bound the rounding errors in initial intervals of rows by

$$\sum_{j=1}^{b}(y_{ij} - x_{ij}) = \sum_{i=1}^{\ell} 2^{-i+1} \sum_{j=1}^{b} \left(x_{ij}^{(i-1)} - x_{ij}^{(i)}\right),$$

which leads to an estimate of $2D$ for all these rounding errors. For initial intervals of columns the same argument holds.

The key observation leading to the improvement is the following. Say we have computed Y'' as above in some iteration. Define $\tilde{Y}''$ by $2X'' - Y''$, that is, in $\tilde{Y}''$ each $\frac{1}{2}$-entry of X is rounded to the opposite value of the one in Y''. Note that this $\tilde{Y}''$ in place of Y'' still satisfies (2.6) and (2.7). However, as $(\tilde{Y}'' - X'') = -(Y'' - X'')$, all rounding errors have the opposite sign.

We may use this observation in the last ℓ_0 iterations to ensure that for each initial interval of a row or column, the corresponding sums in $X^{(\ell_0)} - X^{(\ell)}$, $X^{(\ell_0-1)} - X^{(\ell_0)}$, …, $X^{(0)} - X^{(1)}$ do not all have the same sign (where we distinguish the three signs positive, zero, and negative). This is done by choosing Y'' or $\tilde{Y}''$ in that way that at least half of the still critical initial intervals receive such a sign change.

The worst thing that can happen now is that the corresponding sums of some initial interval in $X^{(\ell_0-1)} - X^{(\ell_0)}, \ldots, X^{(0)} - X^{(1)}$ all equal D or $-D$, but are zero in $X^{(\ell_0)} - X^{(\ell)}$. This leads to a rounding error of

$$\sum_{i=1}^{\ell_0} 2^{-i+1} D = 2\left(1 - 2^{-\ell_0}\right) \leq 2\left(1 - \frac{1}{4mn}\right).$$

If the binary length of X is larger than ℓ we arbitrarily round its entries to numbers having binary length ℓ. For an initial interval of length b, this inflicts an additional error of at most $2^{-\ell}b$.

By combining the binary rounding method from Lemma 2.8 and the half-integral rounding method from Lemma 2.7, we obtain the following result.

Theorem 2.5. *For all $\ell \in \mathbb{N}$ and $X \in [0,1)^{m\times n}$ a rounding $Y \in \{0,1\}^{m\times n}$ such that*

$$\forall b \in [1,\ldots,n],\ i \in [1,\ldots,m]\colon \left|\sum_{j=1}^{b}(x_{ij} - y_{ij})\right| \leq 1 - \frac{1}{4mn} + 2^{-\ell}b,$$

$$\forall b \in [1,\ldots,m],\ j \in [1,\ldots,n]\colon \left|\sum_{i=1}^{b}(x_{ij} - y_{ij})\right| \leq 1 - \frac{1}{4mn} + 2^{-\ell}b$$

can be computed in time $O(\ell mn)$.

Note that we can round any matrix $X \in \mathbb{R}^{m\times n}$ by simply ignoring the integral part of the matrix. Choosing $\ell > \log_2(4mn \max\{m,n\})$ in the theorem above yields Theorem 2.2 in the introduction.

2.3.3 Unbiased Binary Rounding

In this section we give a randomised algorithm that computes a randomised locally consistent controlled rounding satisfying Theorem 2.1. First observe that the $\{0, \frac{1}{2}\}$ case has a very simple randomised solution. Whenever it has to round a cycle, it chooses one of the two alternating 0-1-sequences for each cycle uniformly at random. Then, each $x_{ij} = \frac{1}{2}$ is rounded up with probability $\frac{1}{2}$.

Now consider the output of the bitwise rounding algorithm using this randomised rounding variant for the half-integral case as subroutine. We adapt the methods of [Doe06] to show that this algorithm computes an unbiased controlled rounding.

Theorem 2.6. *Let $X \in [0,1)^{m\times n}$ be a matrix containing entries with binary representation of length at most ℓ. Let Y be a random variable modelling the output of the randomised algorithm. Then Y is a randomised locally consistent controlled roundingof X.*

Proof. We prove $Y \approx X$ by induction. For $\ell = 1$ it is clear that $\Pr(y_{ij} = 1) = x_{ij}$. If $\ell > 1$, write $x_{ij} = x'_{ij} + \frac{1}{2}x''_{ij}$, where $x'_{ij} \in \{0, \frac{1}{2}\}$ and $x''_{ij} \in [0,1)$ has bit-length $\ell - 1$. Let y''_{ij} be the rounding computed for x''_{ij}. Then $\Pr(y''_{ij} = 1) = x''_{ij}$ by induction. Now the algorithm will round $\tilde{x}_{ij} := x'_{ij} + \frac{1}{2}y''_{ij} \in \{0, \frac{1}{2}, 1\}$ to y_{ij}. If $y''_{ij} = 1$, then $\tilde{x}_{ij}$ will be rounded up with probability 1 if $x'_{ij} = \frac{1}{2}$ and with probability $\frac{1}{2}$ otherwise. If, on the other hand, $y''_{ij} = 0$, then $\tilde{x}_{ij}$ will be rounded up with probability x'_{ij}. Thus

$$\Pr(y_{ij} = 1) = x''_{ij}(\frac{1}{2} + x'_{ij}) + (1 - x''_{ij})x'_{ij} = x'_{ij} + \frac{1}{2}x''_{ij} = x_{ij}.$$

To prove that for any initial column interval a randomised rounding is computed, observe that $s_y := \sum_{j=1}^{b} y_{ij}$ is a rounding of $s_x := \sum_{j=1}^{b} x_{ij}$ by Lemma 2.8 for $i \in [1,\dots,m]$ and $b \in [1,\dots,n]$. Furthermore it holds that $E(s_y) = \sum_{j=1}^{b} E(y_{ij}) = s_x$ by linearity of expectation. But also

$$E(s_y) = \Pr(s_y = \lfloor s_x \rfloor)\lfloor s_x \rfloor + \Pr(s_y = \lfloor s_x \rfloor + 1)(\lfloor s_x \rfloor + 1),$$

which is only possible if $s_y \approx s_x$. The proof for initial row intervals is analogous. Hence the computed rounding is a randomised locally consistent controlled rounding according to Lemma 2.3. □

2.4 Rational Rounding

We will now give a different algorithm for computing randomised locally consistent controlled roundings. While the algorithm that we studied in the previous section can handle binary data, applications may require to round fractions that don't have a finite binary representation. This is especially useful in statistics. For confidentiality protection, one often wants to round to multiples of 3. If the goal is to improve the readability of tabular data, on the other hand, rounding to multiples of 10 may be desirable.

2.4.1 The Algorithm

Index Intervals

What properties does a locally consistent controlled rounding Y of X fulfilling the inequalities of Definition 2.1 have? Substituting $b = n$ in inequality (2.1), we can deduce that the ith row of Y must contain exactly $\sum_{j=1}^{n} x_{ij}$ many 1-entries. To fulfil the inequality for $b \neq n$, there must be $\lfloor \sum_{j=1}^{b} x_{ij} \rfloor$ or $\lceil \sum_{j=1}^{b} x_{ij} \rceil$ many 1-entries in column 1 to b of the ith row of Y. Inequality (2.2) gives analogous statements for columns. This observation suggests that we should put one 1 in each interval bounded by two positions where the integral part of the partial row (resp. column) sum increases. This motivates the following definition.

Definition 2.4. *The* kth index interval of the ith row of X *is defined as*

$$I_i^X(k) := \left\{ j \in [1, \ldots, n] \mid x_{ij} \neq 0 \wedge \sum_{\ell=1}^{j} x_{i\ell} > k - 1 \wedge \sum_{\ell=1}^{j-1} x_{i\ell} < k \right\}.$$

The kth index interval of the jth column of X *is defined analogous.*

Observe that the sum over all entries of an index interval is at least one. Because of this, each index interval consists of at least two non-zero elements. If the sum is more than one, then the interval shares an entry with a neighbouring interval. The following example shows a row of values and the corresponding index intervals.

$$\overbrace{0.2 \quad 0.7 \quad 0}^{I(1)}.\overbrace{8 \quad 0}^{I(2)}.\overbrace{6 \quad 0.4 \quad 0.3}^{I(3)} \qquad \overbrace{0.5 \quad 0.4 \quad 0.1}^{I(4)}$$

The idea now is to "concentrate the total value of all entries" of an index interval into a single entry until it has value 1. For this, observe what happens if we pick two non-zero entries in the same row index interval and modify one by $+\frac{1}{q}$ and the other by $-\frac{1}{q}$. Obviously this doesn't change any of the partial sums left of the first or right of the second entry. In particular, the total sum of this row stays unchanged. The same holds for columns.

```
ROUND(X ∈ (1/q ℤ ∩ [0, 1))^{m×n})
 1  t = 0
 2  X^(0) = X
 3  Compute row and column index intervals of X^(0)
 4  while X^(t) ∉ {0, 1}^{m×n} do
 5      C = FINDCYCLE(X^(t))
 6      Choose a ∈ {+1/q, −1/q}
 7      X^(t+1) = alternatingly augment X^(t) along C by ±a
 8      t = t + 1
 9      Update row and column index intervals of X^(t).
10  return Y := X^(t)
```

Algorithm 2.2: The rounding algorithm.

The Algorithm

The algorithm now iteratively modifies the matrix until all elements are 0 or 1. In each step it first constructs a cycle in the current matrix that alternatingly pairs two directly adjacent fractional elements in the same row interval resp. column interval[1]. This way each element of the cycle has one horizontal and one vertical neighbour in the cycle. How to construct such cycles will be discussed later. The algorithm then traverses this cycle and alternatingly adds $\frac{1}{q}$ and subtracts $\frac{1}{q}$ to each cycle entry.

The current matrix is stored in a two-dimensional doubly linked list where every non-integral entry has a pointer to the next non-integral entry in each direction. For the cycle finding step the algorithm must keep track of the index intervals of the current matrix $X^{(t)}$. To do this, the fractional parts $s_{ij}^{col} := \{\sum_{k=1}^{i} x_{kj}\}$ and $s_{ij}^{row} := \{\sum_{k=1}^{j} x_{ik}\}$ of the partial row and column sums of each entry $x_{ij}, i \in [1, \ldots, m], j \in [1, \ldots, n]$ are computed for the initial matrix and updated during the augmentation step. With these values the algorithm can decide if the neighbour of an entry belongs to the same index interval or not, based on the value of the neighbour entry and on the fractional part of the current entry. Whenever two neighbouring elements inside the same index interval are augmented, only their partial sums change, hence the cost of an update is linear in the size of the current cycle.

If an augmentation changes an element to 0 or 1, it is removed from the data

[1]There is a special case where this is not true, as we will see later.

structure. Also, when updating the intervals, such element are ignored. By disregarding entries changed to 1, the corresponding row and column sums decrease by 1 and thus also the number of intervals decreases by 1 if this happens. Since the fractional part of an element that changes to 0 or 1 is also 0, this does not change the values s_{ij}^{col} or s_{ij}^{row} for any other element.

Runtime and Unbiasedness

For the moment let us assume that the call in line 5 of the algorithm always returns a cycle and takes time proportional to the cycle size. Does the algorithm terminate? As we will see, this depends on how we choose a in line 6. Each of the two possible choices corresponds to one of the two possible augmentations along the cycle. Either we start by adding $+\frac{1}{q}$ to the first element on the cycle, then $-\frac{1}{q}$ to the second and so on, or we start by adding $-\frac{1}{q}$ then $+\frac{1}{q}$ and so on. If one of this possibilities is chosen uniformly at random, we have the following theorem.

Theorem 2.7. *Assume that in line 6 of Algorithm 2.2, the value a is chosen independently at random such that* $\Pr(a = \frac{1}{q}) = \Pr(a = -\frac{1}{q}) = \frac{1}{2}$. *Then the following holds.*

a) The algorithm terminates in expected time $O(mnq^2)$.

b) Each $x_{ij}, i \in [1, \ldots, n], j \in [1, \ldots, n]$ *is rounded to one with probability* x_{ij}.

Proof. a) Consider an entry $x_{ij}, i \in [1, \ldots, m], j \in [1, \ldots, n]$ of the cycle. Each time the algorithm considers this entry, it will either add or subtract $\frac{1}{q}$ from it. Both augmentations happen with probability $\frac{1}{2}$. But this is equivalent to doing a random walk on a line with $q + 1$ elements, starting from position $q \cdot x_{ij}$. From Lemma 2.4 it follows that the element becomes 0 or 1 after an expected number of $O(q^2)$ augmentations. As soon as this happens, x_{ij} will no longer belong to any index interval, and hence will no longer be chosen during the cycle construction. Since the matrix has mn entries, the first claim follows. Claim b) follows immediately from Lemma 2.3. □

Finding Cycles

We now specify the function FindCycle used by the algorithm to find a cycle along which it can round. As we will see, the fact that we aim at low errors in all initial intervals (and not only whole rows and columns) imposes some subtle additional difficulties.

First an arbitrary non-integral matrix entry a_1 is chosen as current entry. Then, alternatingly pick a non-integral entry directly adjacent to the current entry in the

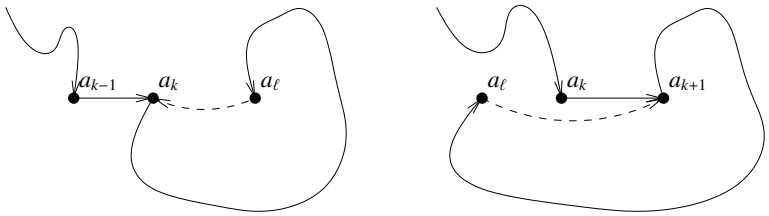

Figure 2.3: The two possibilities during cycle construction.

same row interval resp. in the same column interval as new current entry. This way, a sequence $(a_1, \ldots, a_\ell)$ of matrix entries is constructed. Since each index interval contains at least two fractional entries, the cycle construction routine can not fail to construct a cycle C as long as the matrix is not integral. The algorithm stops as soon as an element already picked before, say $a_k, k \in [1, \ldots, \ell - 1]$, can be chosen as current element. Assume that a_k and a_ℓ share a row interval[2]. By construction, either a_{k-1} or a_{k+1} will also be an element of this row. If a_{k-1} is an element of this row, $C := (a_k, \ldots, a_\ell)$ is a cycle alternatingly pairing row and column elements sharing common intervals as needed by the main algorithm. However, if a_{k+1} is an element of this row, the above cycle would contain two successive edges pairing row entries, namely (a_ℓ, a_k) and (a_k, a_{k+1}).

In this case, the cycle $C := (a_{k+1}, \ldots, a_\ell)$ is chosen instead which again alternatingly pairs row and column elements (See Figure 2.3 for details.). As this cycle now contains an edge pairing an element to its neighbours neighbour, the algorithm has to modify one additional partial sum during the augmentation, namely the one of a_k by $\pm\frac{1}{q}$ depending on how the pair (a_ℓ, a_{k+1}) is augmented. Observe that if a_k belongs to two overlapping index intervals, then a_ℓ and a_{k+1} belong to different intervals. As we will see in the analysis, this will not influence the correctness of the algorithm.

We finally argue that FindCycle has an amortised runtime of $\Theta(|C|)$, where $|C|$ is the length of the cycle computed. Because augmenting along C only changes the local structure between two paired elements, the remaining elements of the sequence that were not chosen for C still alternatingly connect entries of the same row resp. column interval. Hence, the next time FindCycle is called, it can reuse the part of the sequence not used to construct the cycle. Thus, over the whole algorithm, each element is touched during cycle construction as often as it is part of a cycle.

[2]Again the same holds for columns.

Correctness

We will now show that our algorithm indeed computes a randomised locally consistent controlled rounding. In the following we only consider rows. The arguments for columns are analogous. Hence, let $(x_1^{(t)}, \dots, x_n^{(t)}) := (x_{i1}^{(t)}, \dots, x_{in}^{(t)})$ be the elements of the ith row of $X^{(t)}$ for an arbitrary $i \in [1, \dots, m]$. Let $I^{(t)}(1), \dots, I^{(t)}(k)$ be the $k := \sum_{j=1}^{n} x_j^{(t)}$ index intervals of this row. For $\ell \in [1, \dots, k]$ we write $L(I^{(t)})(\ell) := \min(I^{(t)}(\ell))$ and $R(I^{(t)})(\ell) := \max(I^{(t)}(\ell))$ for the position of the leftmost and rightmost entry of the ℓth interval. If $L(I^{(t)})(\ell)$ (resp. $R(I^{(t)})(\ell)$) does not belong to two intervals, we call it *proper*. If both $L(I^{(t)})(\ell)$ and $R(I^{(t)})(\ell)$ are proper, we call the corresponding index interval $I^{(t)}(\ell)$ *proper*.

The *interior of an interval* $I^{(t)}(\ell)^\circ$ is defined as the set of all elements that only belong to this interval. Hence, $I^{(t)}(\ell)^\circ = I^{(t)}(\ell)$ if and only if the interval is proper.

By the definition of index intervals, $R(I^{(t)})(\ell)$ is proper if and only if the partial sum up to this entry is integral. $L(I^{(t)})(\ell)$ is proper if and only if $R(I^{(t)})(\ell - 1)$ is proper. Hence, $I^{(t)}(\ell)$ is proper if the sum over all entries in $I^{(t)}(\ell)$ is 1.

In the special case where we constructed a cycle pairing two entries $x_a^{(t)}, x_b^{(t)}$ from neighbouring row intervals, those intervals share a common element $x_j^{(t)} \neq 0$, $j \in [a, \dots, b]$. Augmenting along this cycle introduces no inconsistencies in the columns, as all other pairs of entries are taken from a common interval. Modifying $x_a^{(t)}, x_b^{(t)}$ to, say, $x_a^{(t)} + \frac{1}{q}, x_b^{(t)} - \frac{1}{q}$, can, for the analysis, be viewed as modifying $x_a^{(t)} + \frac{1}{q}, x_j^{(t)} - \frac{1}{q}$ and $x_j^{(t)} + \frac{1}{q}, x_b^{(t)} - \frac{1}{q}$ independently. Since $x_j^{(t)}$ is a non-zero multiple of $\frac{1}{q}$ shared by both intervals, this is always possible.

First we show that as long as no element of an interval is set to one, the interval will only contract.

Lemma 2.9. *Let $I^{(t)}(\ell)$ be the ℓth interval at time t. Assume that no entry of $I^{(t)}(\ell)$ changes to* 1. *Let $I^{(t+1)}(\ell')$ be an interval at time $t+1$ that intersects $I^{(t)}(\ell)$.*

a) If $R(I^{(t)})(\ell)$ is proper, then $R(I^{(t+1)})(\ell')$ is proper and $R(I^{(t)})(\ell) \geq R(I^{(t+1)})(\ell')$.

b) If $L(I^{(t)})(\ell)$ is proper, then $L(I^{(t+1)})(\ell')$ is proper and $L(I^{(t)})(\ell) \leq L(I^{(t+1)})(\ell')$.

c) If $L(I^{(t)})(\ell)$ and $L(I^{(t+1)})(\ell')$ are not proper, then $L(I^{(t)})(\ell) = L(I^{(t+1)})(\ell')$.

d) If $L(I^{(t)})(\ell)$ is not proper, but $L(I^{(t+1)})(\ell')$ is proper, then $R(I^{(t+1)})(\ell' - 1) \leq R(I^{(t)})(\ell) = L(I^{(t)})(\ell) \leq L(I^{(t+1)})(\ell')$.

Proof. First observe that if $L(I^{(t)})(\ell)$ (resp. $R(I^{(t)})(\ell)$) is proper, it can only be paired with an element to its right (left). Since augmenting a pair does not change the partial sum of the right element of the pair, the first statement follows. For the second statement observe that the partial sum up to $L(I^{(t)})(\ell)$ has the same fractional value as this element. Hence before it can be made small enough to

belong to the $(\ell - 1)$th interval, it will be zero, since all changes are done in steps of $\frac{1}{q}$. Statement c) follows from the fact that a shared element always has value larger than $\frac{1}{q}$. Since the augmentation only changes each value by at most $\frac{1}{q}$, this also proves d). □

Lemma 2.10. *Let $I^{(t)}(\ell)$ be the ℓth interval at time t and let $x_a^{(t)}$ be an element of $I^{(t)}(\ell)$ that changes to* 1*. If $x_a^{(t)}$ is shared with $I^{(t)}(\ell + 1)$, both intervals will merge. Otherwise $I^{(t)}(\ell)$ vanishes and both the rightmost border of the interval to the left as well as the leftmost border of the interval to the right become proper.*

Proof. Surely $x_a^{(t)} = \frac{q-1}{q}$ or it could not change to 1. First, assume that $x_a^{(t)}$ is shared with $I^{(t)}(\ell + 1)$. Then the partial sum for $x_a^{(t)}$ must have fractional value smaller than $x_a^{(t)}$, hence the intervals will merge.

Now assume that $x_a^{(t)}$ is an inner element of $I^{(t)}(\ell)$. Then the partial sum for $x_a^{(t)}$ must either be $\frac{q-1}{q}$ and $L(I^{(t)})(\ell) = a$ is proper, or it must be integral and $R(I^{(t)})(\ell) = a$ is proper. Note that in both cases the other border of $I^{(t)}(\ell)$ is the only non-integral element of this interval. If this element is also proper, the lemma obviously holds. Hence assume it is a shared entry (and thus has value at least $\frac{2}{q}$). If $a = L(I^{(t)})(\ell)$, then the augmentation will cause the partial sum for $x_a^{(t)}$ to become integral, making $R(I^{(t)})(\ell)$ the proper left border of $I^{(t)}(\ell + 1)$. Otherwise the augmentation will cause the partial sum for $L(I^{(t)})(\ell)$ to become integral, making it the proper right border of $I^{(t)}(\ell - 1)$. □

Lemma 2.11. *Let Y be a rounding of X as computed by Algorithm 2.2. Then for each row there exists a bijective mapping between elements rounded to* 1 *and index intervals of this row in X, mapping each element to an interval containing it. The same holds for columns.*

Proof. Let $K := (I_a, \ldots, I_b)$ be a maximum collection of neighbouring intervals in an arbitrary row of X such that $I_j \cap I_{j+1} \neq \emptyset$ for $j \in [a, \ldots, (b - 1)]$. In other words, exactly $L(I_a)$ and $R(I_b)$ are proper. Clearly, it suffices to prove the lemma for such subcollections.

First assume that at time t no element is changed to 1. If none of the borders shared by intervals in K become proper, then nothing changes according to Lemma 2.9. Otherwise, K decomposes into smaller collections of intervals which can be treated separately.

Now assume that at time t an inner element $x_j^{(t)}$ of a current interval changes to 1. By Lemma 2.10, this interval was obtained by merging $d - c + 1$ neighbouring intervals $I_c, \ldots, I_d, a \leq c \leq d \leq b$ of the initial collection. This means that their $d - c$ shared entries were set to 1 during the algorithm. Hence we can assign 1 to the interval of the initial collection containing $x_j^{(t)}$, and the remaining $d - c$ 1s to the other intervals. Since by Lemma 2.10 the borders of the neighbours of this

ComputeRounding($X \in \frac{1}{q_1 \cdot q_2}\mathbb{Z}^{m\times n}$)
1 Compute $X' \in \frac{1}{q_1}\mathbb{Z}^{m\times n}, X'' \in \frac{1}{q_2}\mathbb{Z}^{m\times n}$ such that $X = X' + \frac{1}{q_1}X''$
2 $Y'' = \text{Round}(X'') \in \{0,1\}^{m\times n}$
3 $\tilde{X} = X' + \frac{1}{q_1}Y'' \in \frac{1}{q_1}\mathbb{Z}^{m\times n}$
4 $Y = \text{Round}(\tilde{X}) \in \{0,1\}^{m\times n}$
5 **return** $Y \in \{0,1\}^{m\times n}$

Algorithm 2.4: The factor rounding algorithm.

interval become proper in this case, we get two smaller subcollections which can be treated separately. □

Theorem 2.8. *If Y is a rounding of X computed by Algorithm 2.2, then*

$$\forall b \in [1,\dots,n],\ i \in [1,\dots,m] : \left|\sum_{j=1}^{b}(x_{ij} - y_{ij})\right| < 1,$$

$$\forall b \in [1,\dots,m],\ j \in [1,\dots,n] : \left|\sum_{i=1}^{b}(x_{ij} - y_{ij})\right| < 1.$$

Proof. Let $b \in [1,\dots,n]$ and $i \in [1,\dots,m]$. If $x_{ib} = 0$ then $y_{ib} = 0$. Hence it suffices to regard the case $x_{ib} \neq 0$. Let $\ell \in \mathbb{N}$ be maximal such that x_{ib} is contained in the ℓth interval of the ith row of X. By definition, this means that $\ell - 1 < \sum_{j=1}^{b} x_{ij} \leq \ell$. By Lemma 2.11, $\ell - 1 \leq \sum_{j=1}^{b} y_{ij} \leq \ell$ holds. If $\sum_{j=1}^{b} x_{ij} < \ell$, this shows the theorem. In the other case, x_{ib} must be the last element of the ℓth interval, hence $\sum_{j=1}^{b} y_{ij} = \ell$. For columns, the proof is analogous. □

2.4.2 Speeding it up

Assume that q has a non-trivial factorisation $q = q_1 \cdot q_2$ with $q_1, q_2 \in \mathbb{N}_{\geq 2}$. Then we can exploit this fact to severely the runtime given in Theorem 2.7. Our approach resembles the one used in [Doe06] for powers of 2.

Lemma 2.12. *Let $X \in \frac{1}{q}\mathbb{Z}^{m\times n}$ be a rational matrix with $q = q_1 q_2$ and $q_1, q_2 \in \mathbb{N}$.*

Then Algorithm 2.4 will compute an unbiased rounding Y of X satisfying

$$\forall b \in [1,\dots,n]\,,\ i \in [1,\dots,m] \quad : \quad \left|\sum_{j=1}^{b}(x_{ij}-y_{ij})\right| < 1,$$

$$\forall b \in [1,\dots,m]\,,\ j \in [1,\dots,n] \quad : \quad \left|\sum_{i=1}^{b}(x_{ij}-y_{ij})\right| < 1.$$

Proof. First note that each matrix entry $x_{ij}, i \in [1,\dots,m]\,, j \in [1,\dots,n]$ is decomposed into $x'_{ij} \in \frac{1}{q_1}\mathbb{Z}$ and $x''_{ij} \in \frac{1}{q_2}\mathbb{Z}$. To show unbiasedness observe that in line 2 an unbiased rounding $y''_{ij} \in \{0,1\}$ of x''_{ij} is computed according to Theorem 2.7. In other words, $\tilde{x}_{ij}$ computed in line 3 will have value $x'_{ij} + \frac{1}{q_1}$ with probability x'', and value x'_{ij} otherwise. From line 4 it follows that

$$\begin{aligned}\Pr(y_{ij}=1) = \Pr(\tilde{x}_{ij} \nearrow 1) &= x''_{ij}Pr((x'_{ij}+\frac{1}{q_1}) \nearrow 1) + (1-x''_{ij})Pr(x'_{ij} \nearrow 1)\\ &= x''_{ij}(x'_{ij}+\frac{1}{q_1}) + (1-x''_{ij})x'_{ij}\\ &= \frac{1}{q_1}x''_{ij} + x'_{ij} = x_{ij}.\end{aligned}$$

Hence the algorithm computes an unbiased rounding of X.

To see that the rounding computed in Figure 2.4 is a controlled rounding satisfying our additional constraints, let $s_{ij}(X) := \sum_{k=1}^{i} x_{kj}$ for $i \in [1,\dots,m]\,, j \in [1,\dots,n]$ be the sum over the first i elements of the jth column of X. In line 2 the algorithm computes a controlled rounding Y'' of X'' satisfying our additional constraints. Hence

$$\left|s_{ij}(X''-Y'')\right| \le 1 - \frac{1}{q_2}.$$

A similar statement holds for Y and $\tilde{X}$ in line 4, namely

$$\left|s_{ij}(\tilde{X}-Y)\right| \le 1 - \frac{1}{q_1}.$$

Together with the triangle inequality these two error bounds yield

$$\begin{aligned}\left|s_{ij}\,(X-Y)\right| &= \left|s_{ij}\left(X' + \frac{1}{q_1}X'' - \frac{1}{q_1}Y'' + \frac{1}{q_1}Y'' - Y\right)\right|\\ &\le \left|s_{ij}\left(\tilde{X}-Y\right)\right| + \frac{1}{q_1}\left|s_{ij}\,(X''-Y'')\right|\\ &\le 1 - \frac{1}{q_1} + \frac{1}{q_1}\left(1-\frac{1}{q_2}\right) = 1 - \frac{1}{q}.\end{aligned}$$

Hence the error in all initial column intervals is at most $1 - \frac{1}{q}$. The proof for the error in initial row intervals and in single elements is analogous. □

Now let $q = \prod_{i=1}^{\ell} q_i,\ q_i \in \mathbb{N}$ be a factorisation of the denominator of X. Then the algorithm in Figure 2.4 can be applied recursively to get the main result as stated in Theorem 2.3.

Now consider the case of half-integral matrices $X \in \{0, \frac{1}{2}\}^{m \times n}$. Here an augmentation of an element by $\pm\frac{1}{2}$ will always change the element to either 0 or 1. Hence Algorithm 2.2 will run in *deterministic* time $O(mn)$ for this special case. Using this observation and choosing $q = 2^{\ell}$ we get another proof of Theorem 2.1 for unbiased rounding of matrices of ℓ-bit numbers.

Corollary 2.1. *Let $X \in [0, 1)^{m \times n}$. Assume that each entry of X has binary length at most ℓ. Then a randomised locally consistent controlled rounding Y of X can be computed in time $O(mn\ell)$.*

2.4.3 Derandomisation

We will now show how to derandomise Algorithm 2.2. For this we use the *method of conditional probabilities*. An introduction to this technique can be found in the paper by Spencer [Spe94].

First observe that by Lemma 2.4 the expected number $\mathbb{E}(\text{Steps}(X))$ of augmentations needed to round a given matrix $X \in (\frac{1}{q}\mathbb{Z} \cap [0, 1))^{m \times n}$ is

$$\mathbb{E}(\text{Steps}(X)) = \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij}(q - x_{ij}) = O(mnq^2).$$

The derandomisation now works as follows. At the beginning of Algorithm 2.2 the expected number of augmentations $\mathbb{E}(\text{Steps}(X))$ is computed. Each time one of the two possible ways to augment along a cycle C in line 6 of the algorithm must be chosen we don't choose randomly. Instead, the augmentation for which the algorithm would need the fewer number of expected steps if it would continue choosing randomly is picked. By Lemma 2.4 it follows that

$$\mathbb{E}(\text{Steps}(X)) = |C| + \frac{1}{2}\mathbb{E}(\text{Steps}(X - X_C)) + \frac{1}{2}\mathbb{E}(\text{Steps}(X + X_C)),$$

where X_C is the matrix for one of the two possible augmentations along C. From this formula it follows that $\mathbb{E}(\text{Steps}(X - X_C))$ and $\mathbb{E}(\text{Steps}(X + X_C))$ cannot both be larger than $\mathbb{E}(\text{Steps}(X)) - |C|$. Hence, each time the algorithm augments along a cycle C, $\mathbb{E}(\text{Steps}(X))$ decreases by at least $|C|$, since the augmentation giving the smaller expected value is picked.

Calculating $\mathbb{E}(\text{Steps}(X))$ for the input matrix needs time $O(mn)$. Deciding which augmentation to use for cycle C in step t can be done in time $O(|C|)$ while constructing the cycle. The value $\mathbb{E}(\text{Steps}(X^{(t)}))$ can be derived from the expected value of the previous matrix $\mathbb{E}(\text{Steps}(X^{(t-1)}))$ in $O(|C|)$ time while doing the actual augmentation. This gives the following theorem.

Theorem 2.9. *For all $X \in \frac{1}{q}\mathbb{Z}^{m\times n}$ a locally consistent controlled rounding $Y \in \mathbb{Z}^{m\times n}$ can be computed in time $O(mnq^2)$.*

Together with Lemma 2.12, this yields Theorem 2.4 from the introduction.

2.5 A Lower Bound

As we have seen in the previous sections, it is always possible to compute a locally consistent controlled rounding of a rational matrix. Such a rounding exhibits an error of less than one in initial intervals and, by the triangle inequality, of less than two in arbitrary intervals. But what is the minimum error we have to tolerate when computing a locally consistent controlled rounding? We now give a lower bound on the errors in initial and arbitrary intervals.

In [SY93], Steiner and Yeomans show that the error in initial intervals may be arbitrary close to one. More precisely, for every $n \in \mathbb{N}$ there exists a $n \times n$ matrix that has an error of $1 - \frac{1}{n}$ in at least one initial interval. To see this simply consider the matrix $((\frac{1}{n})_{i,j})_{i=1\ldots n, j=1\ldots n}$. Obviously, all locally consistent controlled roundings of this matrix are permutations of the identity matrix. Hence, there is one row that has $n-1$ zeroes followed by a one. The initial interval containing the first $n-1$ entries of this row has an error of

$$\left|\sum_{i=1}^{n-1}\left(\frac{1}{n} - 0\right)\right| = 1 - \frac{1}{n}.$$

This example gives the same lower bound for arbitrary intervals. However, one might conjecture that investigating arbitrary intervals should create larger errors. As we show next, this is indeed the case.

We now give an improved lower bound on the errors in arbitrary intervals. In particular, we give a $3 \times n$ matrix such that any locally consistent controlled rounding of this matrix contains an interval with error $1.5 - \epsilon$. As a corollary, this gives an error of $0.75 - \epsilon$ for initial intervals. While this is not as good as the above example by Steiner and Yeomans, it needs only 3 rows instead of n.

Theorem 2.10. *For any $\epsilon > 0$ there exists an $n \in N$ and a matrix $X \in \mathbb{R}^{3\times n}$ such that for any locally consistent controlled rounding $Y \in \mathbb{Z}^{3\times n}$ there are $i \in \{1, 2, 3\}$*

and $1 \leq a \leq b \leq n$ *with*

$$\left| \sum_{j=a}^{b} \left(x_{ij} - y_{ij} \right) \right| \geq 1.5 - \epsilon.$$

Proof. Let $\epsilon' := \frac{1}{4}\epsilon$ and $n > \frac{1.5}{\epsilon'^2}$. Set $X \in \mathbb{R}^{3\times n}$ with

$$X := \begin{pmatrix} 1-\epsilon' & 1-\epsilon' & \cdots & 1-\epsilon' \\ \epsilon' - \epsilon'^2 & \epsilon' - \epsilon'^2 & \cdots & \epsilon' - \epsilon'^2 \\ \epsilon'^2 & \epsilon'^2 & & \epsilon'^2 \end{pmatrix}$$

Assume that there is a locally consistent controlled rounding $Y \in \mathbb{Z}^{3\times n}$ of X that exhibits a rounding error of less than $1.5 - \epsilon$ in all intervals. The row sum of the third row of X is at least $n\epsilon'^2 > \frac{1.5}{\epsilon'^2}\epsilon'^2 = 1.5$, hence there is at least one column j having a one in the third row. Since all column sums are 1, the first and second entry of column j must hence be zero. Let $p \geq 0$ and $q \geq 0$ be the number of consecutive columns of Y of the form $(1,0,0)^T$ to the left and right of the jth column. In other words, the matrix Y has the form

$$Y = \begin{pmatrix} & 0 & 1\ldots1 & 0 & 1\ldots1 & 0 & \\ \cdots & ? & 0\ldots0 & 0 & 0\ldots0 & ? & \cdots \\ & ? & \underbrace{0\ldots0}_{p \text{ times}} & \underbrace{1}_{j\text{th column}} & \underbrace{0\ldots0}_{q \text{ times}} & ? & \end{pmatrix}.$$

Now consider the interval $[(j-p-1)..(j+q+1)]$ in the first row. It has a rounding error of

$$\begin{aligned} \left| \sum_{\ell=(j-p-1)}^{(j+q+1)} (x_{1\ell} - y_{1\ell}) \right| &= (p+q+3)(1-\epsilon') - (p+q) \\ &= 3(1-\epsilon') - (p+q)\epsilon'. \end{aligned}$$

By assumption, this must be less than $1.5 - \epsilon$. Hence we can bound $p + q$ by

$$p + q > \frac{1}{\epsilon'}\left(3(1-\epsilon') - 1.5 + 4\epsilon'\right) = \frac{1.5}{\epsilon'} + 1.$$

With this the rounding error of the interval [(j-p)..(j+q)] in the second row can be bound by

$$\begin{aligned} \left| \sum_{\ell=(j-p)}^{(j+q)} (x_{2\ell} - y_{2\ell}) \right| &= (p+q+1)(e' - e'^2) \\ &> (\frac{1.5}{\epsilon'} + 2)(\epsilon' - \epsilon'^2) \\ &= 1.5 + 0.5\epsilon' - 2\epsilon' \\ &> 1.5 - 4\epsilon', \end{aligned}$$

which contradicts the assumption □

Corollary 2.2. *For any* $\epsilon > 0$ *there exists a matrix such that any locally consistent controlled rounding of that matrix contains an initial interval with rounding error at least* $0.75 - \epsilon$.

Chapter 3

Evolutionary Algorithms

An interesting class of randomised algorithms are those that are inspired by various principles found in nature. Among them are ant colony optimisation algorithms, simulated annealing and evolutionary algorithms. Since it is simple to adapt them to new problems, such nature-inspired randomised search heuristics are often used to tackle complex or hard problems. Due to this there exists a large body of experimental work on such heuristics.

To gain a deeper insight into how nature-inspired algorithms work they are also often studied for simple and well-known problems. But even those studies are often based on experimental results alone. The last decade, however, produced a growing interest in a theory-founded understanding of these algorithms.

In this chapter, we will study evolutionary algorithms for the single source shortest path and the all-pairs shortest path problem.

Section 3.2 discusses some preliminaries that we will use in the analysis of the algorithms. Also, we exhibit two simple Chernoff type inequalities that are used throughout the analysis. The first is for sums of independent geometrically distributed random variables. The second can be used for sequences of random variables that are not independent, but show a desired behaviour independent of the outcomes of the previous random variables.

In Section 3.3, we analyse an evolutionary algorithm for the single source shortest path problem. This algorithm was proposed by Scharnow, Tinnefeld and Wegener. We prove a bound on the optimisation time of this algorithm that holds with high probability. We also show that this bound is tight by constructing a class of worst-case graphs for the algorithm. To obtain such sharp bounds, we develop a new technique that overcomes the coupon collector behaviour of previously used arguments.

In Section 3.4, we study the all-pairs shortest path problem. For this problem we show that a natural evolutionary algorithm is significantly faster with a crossover operator than without. This is the first theoretical analysis proving the

usefulness of crossover for a non-artificial problem. Previous studies that tried to prove the usefulness of crossover considered simple problems especially tailored to exhibit this behaviour.

This Chapter is based on joint work with Benjamin Doerr and Edda Happ. The results have been published in [DHK07, DHK11, DHK08, DHK12].

3.1 Introduction

The paradigm of evolutionary computation is to use principles inspired by natural evolution, e.g., mutation, crossover and selection, to build algorithms. Genetic algorithms (GA), genetic programming (GP) and evolution strategies (ES) are prominent examples. Together with related approaches like randomised local search (RLS), the Metropolis algorithm [MRR+53], and simulated annealing [KGJV83] they all belong to a class of algorithms known as randomised search heuristics.

Whereas early hopes that these ideas might make notoriously hard problems become tractable did not fulfil, randomised search heuristics nowadays are frequently used as a generic way to obtain algorithms. Naturally, such generic approaches cannot compete with a custom-tailored algorithm. Practitioners still like to use them, because they are easy and cheap to implement, need fewer analysis of the problem to be solved, and can be reused easily for related problems.

Typically, evolutionary algorithms have a collection ("population") of solution candidates ("individuals"), which they try to improve gradually. Improvements may be generated by applying different variation operators, most notably mutation and crossover, to certain individuals. The quality of solutions is measured by a so-called fitness function. Based on this fitness value, a selection procedure may replace some individuals by fitter ones. The cycle of variation and replacement is repeated until a solution of sufficient fitness is found. See, e.g., [For93] for a short introduction to genetic algorithms.

One strength of this general approach is that each component can be adapted to the particular problem under consideration. This adaptation can be guided by an experimental evaluation of the actual behaviour of the algorithms or by previously obtained experience. Also, not all evolutionary algorithms need to have all components described above. For example, the simple variants of randomised local search and the (1+1) evolutionary algorithm have a population size of only one, and consequently, no crossover operator.

Using such evolutionary approaches has proven to be extremely successful in practice (see, e.g., the Proceedings of the annual ACM Genetic and Evolutionary Computation Conferences (GECCO)). In contrast to this, a theoretical understanding of such methods is still in its infancy. One reason for this is that genetic al-

gorithms can be seen as non-linear (namely quadratic) dynamical systems, which are inherently more powerful, but "usually impossible to analyze" (c.f. [RRS95]).

Rigorous run-time analyses of evolutionary algorithms for classical algorithmic problems became a hot topic in the last years. The aim is to obtain a theoretically founded understanding of the basic principles of evolutionary computation. As mentioned above, we are in the situation that we have seen many highly successful applications of evolutionary algorithms, but hardly can explain why they are so successful in practice. Such an understanding, however, would be very helpful in the future design of such algorithms.

Theoretical run-time analyses started on artificial problems like maximising simple pseudo-boolean functions $f : \{0,1\}^n \to \mathbb{R}$, e.g., the number of ones (ONEMAX$(x) := \sum_{i=1}^{n} x_i$), the number of leading ones (LO$(x) := \max\{i \in \mathbb{N}_0 \mid \forall j \leq i : x_j = 1\}$) or monotone linear functions and polynomials [Weg01, DJW02, WW05]. Analysing such artificial problems already lead to some insight on how evolutionary algorithms work and on how to analyse them.

More recently, the focus moved on to classical problems from computer science. In one of the first papers in this direction, [STW04] proposed and analysed evolutionary algorithms for sorting and shortest path problems. Other results on evolutionary algorithms for combinatorial optimisation problems include papers on the problem of computing Eulerian cycles [Neu04, DHN06, DKS07, DJ07], minimum spanning trees [NW04, NW05], maximal matchings [GW03] and partitions [Wit05].

For such problems, the design and analysis of evolutionary algorithms becomes more interesting. One simple reason is that we typically have the choice between several natural representations of individuals, fitness functions and variation operators.

To avoid misunderstandings, let us stress that the focus of this line of research is not to find superior algorithms for the particular underlying optimisation problem. Since these are classical and important problems, they have been investigated thoroughly and hence very good custom-tailored algorithms already exist. The focus rather is to analyse how such problems can be tackled with generic approaches, to understand how their components like particular representations or variation operators work, and, finally, to develop methods to analyse evolutionary algorithms. More information on theoretical analyses of bio-inspired randomised search heuristics can be found in the recent books [NW10, AD11].

Surprisingly, even for extremely simple evolutionary algorithms on simple problems a tight analysis of their run-time behaviour can be very complicated. Note that we only aim at determining its order of magnitude, that is, we ignore constant factors and lower order terms. Also, we only regard the optimisation time, that is, the number of fitness evaluations needed to find the desired solution.

A classical example for the difficulties one faces when analysing evolutionary

algorithms is the $O(n \log(n))$ bound for the optimisation time of a simple $(1 + 1)$ evolutionary algorithm maximising a monotone linear function on $\{0, 1\}^n$. Here, classical methods from the analysis of randomised algorithms lead to a highly technical proof [DJW02]. Subsequent efforts put into this problem resulted in the so-called drift analysis becoming a major tool in the run-time analysis of evolutionary algorithms (see [HY04] for the first use and [Jäg08, DJW12, DG10] for recent simplifications).

The Single-Source Shortest Path Problem

The first work on evolutionary algorithms for combinatorial problems is the analysis of the single source shortest path problem by Scharnow, Tinnefeld and Wegener [STW04]. Naturally, analyses are even harder for problems that carry a richer structure. Hence it is not surprising that their analysis is only tight for certain instances. As we shall see in this work, for many graphs the optimisation time needed to solve the single source shortest path problem is better than what is proven in [STW04].

They proposed a natural $(1 + 1)$ evolutionary algorithm for the problem of finding shortest paths from a single node ("source") to all other vertices in a graph with edge weights. For a graph on n vertices they show an upper bound of $O(n^3)$ for the expected optimisation time. This bound is tight if (and only if, as we shall see) the graph and edge weights are such that there is a vertex such that all shortest paths to the source contain $\Omega(n)$ edges.

The proof given by [STW04] for the single-source shortest path problem reveals an in fact stronger upper bound of $O(n^2 \sum_{i=1}^{n} \log(n_i + 1))$, where n_i is the number of vertices for which a shortest path to the source with the minimum number of edges consists of exactly i edges. Since $\sum_{i=1}^{n}(n_i + 1) = 2n$, this yields a bound of $O(n^2 \ell \log(\frac{2n}{\ell}))$, where ℓ is the smallest integer such that any vertex can be reached from the source via a shortest path having at most ℓ edges.

Crossover and Evolutionary Algorithms

The paradigm of evolution-inspired computing suggests to use both a mutation operator and a crossover operator. Mutation means that a new individual is generated by slightly altering a single parent individual, whereas the crossover operator generates a new individual by recombining information from two parents. Most evolutionary algorithms used in practice use both a mutation and a crossover operator.

In contrast to this, there is little evidence for the need of crossover. In fact, early work in this direction suggests the opposite. In [MHF93], Mitchell, Holland and Forrest experimentally compared the run-time of a simple genetic algorithm

(using crossover) and several hill-climbing heuristics on so-called *royal road functions*. According to Holland's [Hol75] *building block hypothesis*, these functions should be particularly suited to be optimised by an algorithm employing crossover. The experiments conducted in [MHF93], however, clearly demonstrated that this advantage does not exist. In fact, an elementary randomised hill-climbing heuristic (repeated mutation and survival of the fitter one of parent and offspring) was found to be far superior to the genetic algorithm.

The first theoretical analysis indicating that crossover can be useful was given by Jansen and Wegener [JW99] in 1999 (see also [JW02]). For $m < n$, they defined a pseudo-Boolean *jump* function $j_m : \{0,1\}^n \to \mathbb{R}$ such that (more or less) $j_m(x)$ is the number of ones in the bit-string x if this is at most $n - m$ or equal to n, but small otherwise. A typical mutation based evolutionary algorithm (flipping each bit independently with probability $\frac{1}{n}$) will easily find an individual x such that $j_m(x) = n - m$, but will need expected time $\Omega(n^m)$ to flip the remaining m bits (all in one mutation step). However, if we add the uniform crossover operator (here, each bit of the offspring is randomly chosen from one of the two parents) and use it sufficiently seldom compared to the mutation operator, then the run-time reduces to $O(n^2 \log n + 2^{2m} n \log n)$. While the precise computations are far from trivial, this behaviour stems naturally from the definition of the jump function.

The work of Jansen and Wegener [JW99, JW02] was subsequently extended by different authors in several directions [SW04, JW05], partly to overcome the critique that in the first works the crossover operator necessarily had to be used very sparingly. While these works enlarged the theoretical understanding of different crossover operators, they could not resolve the feeling that all these pseudo-Boolean functions were artificially tailored to demonstrate this particular phenomenon. In [JW05], the authors state that "It will take many major steps to prove rigorously that crossover is essential for typical applications."

The only two works (that we are aware of) that address the use of crossover for other problems than maximising a pseudo-Boolean function are "Crossover is Provably Essential for the Ising Model on Trees" [Sud05] by Sudholt and "The Ising Model on the Ring: Mutation Versus Recombination" [FW04] by Fischer and Wegener. They show that crossover also helps when considering a simplified Ising model on special graph classes, namely rings and trees. The simplified Ising model, however, is equivalent to looking for a vertex colouring of a graph such that all vertices receive the same colour. While it is interesting to see that evolutionary algorithms have difficulties addressing such problems, proving "rigorously that crossover is essential for typical applications" remains an open problem.

3.1.1 Our Contribution

We give a tight analysis of the algorithm for the single-source shortest path (SSSP) problem proposed in [STW04]. This leads to an improved upper bound for the expected optimisation time of $O(n^2 \max\{\ell, \log(n)\})$. In addition, we show that this bound not only holds in expectation, but is fulfilled with high probability. That is, for any constant $c > 0$ the implicit constants in the optimisation time bound can be chosen such that after that many iterations, the optimum is found with probability at least $1 - n^{-c}$. The bound on the optimisation time is tight for all ℓ. For all values of ℓ we present a problem instance such that all shortest paths have length at most ℓ, but the optimisation time is $\Omega(n^2 \max\{\ell, \log(n)\})$ with high probability.

To prove strong bounds like this, we develop several tools that might see further applications in the future. To prove the upper bound, we closely analyse how nodes become connected to the source via shortest paths. The speed of growth of such shortest paths (note that they do not have to be unique) displays a strong concentration behaviour. Although we use a union bound argument over all paths needed, it is still strong enough to obtain bounds that hold with high probability. To show the lower bound, we prove a Chernoff type strong concentration bound for sums of geometrically distributed random variables. To the best of our knowledge, such bounds have not been published so far in a mathematics or computer science journal. For both the upper and the lower bound, we encounter a situation where a sequence of random variables is not independent, but has the property that each member of the sequence has a desired property for all possible outcomes of the previous random variables. For such situations, we prove Chernoff type concentration bounds. We are optimistic that all these tools see more applications in the near future.

Apart from the analysis of the SSSP problem we also present the first non-artificial problem for which crossover provably reduces the order of magnitude of the optimisation time. This problem is the all-pairs shortest path (APSP) problem, that is, the problem to find, for all pairs of vertices of a directed graph with edge lengths, the shortest path from the first vertex to the second. This is one of the most fundamental problems in graph algorithms, see for example the books by Mehlhorn and Näher [MN99] or Cormen et al. [CLRS09].

There are two classical algorithms for this problem. The Floyd-Warshall algorithm [Flo62, War62] has a cubic runtime and is quite easy to implement. In contrast, Johnson's algorithm [Joh77] is more complicated, but has a superior runtime on sparse graphs. Since the problem is NP-hard [GJ79] if negative cycles exist and simple paths are sought, we will always assume that all weights are non-negative.

We present a natural evolutionary algorithm for the APSP problem. It has a population consisting of at most one path for every pair of vertices (connecting the

first to the second vertex). Initially, it contains all paths consisting of one edge. A mutation step consists of taking a single path from the population uniformly at random and adding or deleting a (Poisson distributed) random number of times an edge at one of its endpoints. The newly generated individual replaces an existing one (connecting the same vertices) if it is not longer. Hence, our fitness function (which is to be minimised) is the length of the path.

We analyse this algorithm and prove that, in the worst case, it has with high probability an optimisation time of $\Theta(n^4)$, where n is the number of vertices of the input graph.

We then state three different crossover operators for this problem. They all take two random individuals from the population and try to combine them to form a new one. In most cases, of course, this will not generate a path. In this case, we define the fitness of the new individual to be infinite (or some number larger than n times the longest edge). Again, the new individual replaces one having the same endpoints and not smaller fitness.

Using an arbitrary constant crossover rate for any of these crossover operators, we prove an upper bound of $O(n^{3.5}\sqrt{\log n})$ for the expected optimisation time. Hence, for the APSP problem, crossover leads to a reduction of the optimisation time. While the improvement of order $n^{0.5-\epsilon}$ might not be too important, this work solves a long-standing problem in the theory of evolutionary computation. It justifies to use both a mutation and crossover operator in applications of evolutionary computation.

We employ similar methods as in the SSSP case to obtain a tight analysis for the APSP problem. Furthermore, we propose another interesting tool for the analysis of evolutionary algorithms. A classical problem in the analysis of such algorithms is that the mutation operator may change an individual at several places (multi-bit flips in the bit-string model). Hence, unlike for the heuristic of randomised local search, with evolutionary algorithms we cannot rely on the fact that our offspring is in a close neighbourhood of the original search point. While this is intended from the view-point of algorithm design (to prevent being stuck in local optima), this is a major difficulty in the theoretical analysis of such algorithms. Things seem to become even harder if we do not use bit-strings as representations for the individuals. We overcome these problems via what we call *c-trails*. These are hypothetical ways of how to move from one individual to another using simple mutations only.

3.2 Preliminaries

3.2.1 Evolutionary Algorithms

We now give a framework for the evolutionary algorithms studied by us in this chapter. The algorithms we consider repeatedly apply variation and replacement to a set of individuals. We study both an algorithm that only uses mutation and an algorithm that uses both mutation and crossover. Both algorithms share the following common framework.

First, the population of initial *individuals* is initialised. How this is done and how big this initial population is depends on the concrete algorithm.

In evolutionary computation, new individuals are generated by *variation operators*, namely by mutation or crossover (or both). In our case it is decided randomly with a certain fixed probability if a mutation or a crossover step should be done. If a mutation step is done, the algorithm picks an individual uniformly at random from the population and applies the mutation operator to it to generate a new individual. If a crossover step is done, the algorithm picks two individuals uniformly at random from the population and applies a crossover operator to generate a new individual. Note that we may choose a crossover probability of zero, giving an algorithm where only mutation is applied.

The next step is then *(selection for) replacement*. The aim is to prevent the population from growing too big as well as to get rid of individuals that are not considered to be useful solution candidates anymore. Typically, replacement is guided by a *fitness function* assigning each individual a non-negative *fitness*. Strict replacement operators, like truncation, eliminate the unfittest individuals. There are also other replacement operators like fitness proportionate (also called roulette-wheel) or tournament selection that favour fitter individuals more moderately. The first can lead to a faster fitness increase in the population, the latter has the advantage of a higher degree of diversity in the population.

We will only consider selection by truncation. During the replacement step it is checked if there is an individual in the population that is comparable to the new individual. The fitness of both individual is then compared. If the old individual is not fitter than the new individual, it will be replaced by the new individual. If no comparable individual exists and the new individual does not have infinite fitness, it will also be added to the population. Hence the population size can increase to a certain maximum size in this step. This will indeed happen in the case of the all-pairs shortest path problem studied by us. For the single source shortest path problem, on the other hand, our population will always contain exactly one individual.

Variation and replacement are then repeated as long as wanted. Algorithm 3.1 illustrates this procedure.

ALGORITHM FRAMEWORK FOR $(\leq \mu + 1)$-EA AND $(\leq \mu + 1)$-GA

Input: Crossover Probability $p_\otimes \in [0, 1]$
// Initialisation:
1 $\mathcal{I}$ = Set of initial Individuals.
2 **repeat**
3 $x = 1$ with probability $p_\otimes$
4 **if** $x = 1$
// Crossover:
5 I_1 = RANDOMLYCHOOSEFROM($\mathcal{I}$)
6 I_2 = RANDOMLYCHOOSEFROM($\mathcal{I}$)
7 I' = CROSSOVER(I_1, I_2)
8 **else**
// Mutation:
9 I' = RANDOMLYCHOOSEFROM($\mathcal{I}$)
10 s = Pois($\lambda = 1$)
11 **for** $i = 0$ **to** s **do**
12 I' = MUTATION(I')
// Selection:
13 **forall** I **in** $\mathcal{I}$ **do**
14 **if** I and I' are comparable and FITTNESS(I') $\geq$ FITTNESS(I)
15 $\mathcal{I} = \mathcal{I} \setminus \{I\}$
16 $\mathcal{I} = \mathcal{I} \cup \{I'\}$
17 **forever**

Algorithm 3.1: Pseudo-Code for the two algorithms studied by us. If $p_\otimes$ is a constant greater than zero, both mutation and crossover are used and the resulting algorithm will be called $(\leq \mu + 1)$-GA. For $p_\otimes = 0$, only mutation is used as variation operator. We call the resulting algorithm $(\leq \mu + 1)$-EA. μ is the maximum population size that can occur.

Note that this framework is already somewhat tailored to our problem. In general there are evolutionary algorithms that not only select for replacement but also use a specialised selection step to choose the individuals for the variation step. Also, if the population can have a certain maximum size greater one, one often already initialises the population to this size and assume that all individuals are comparable.

If only mutation is used, we get an algorithm that strongly resembles the algo-

rithm called $(\mu+1)$-EA by various authors [JDJW05, Wit06, OHY07]. Indeed, for the SSSP problem where we will deal with mutation only and have a population of size one, we get the classical (1 + 1)-EA. For the APSP problem the population size will not be fixed but can grow up to some value μ. We call the mutation-only algorithms $(\leq \mu + 1)$-EA in that case.

The variant using crossover will be called $(\leq \mu + 1)$-GA to differentiate the cases with and without crossover. This can be justified since it strongly resembles a classical genetic algorithm. We do see that these names are not ideal in the sense that usually evolutionary algorithms are a more general concept, including genetic algorithms, genetic programming and evolution strategies.

The *optimisation time* of an evolutionary algorithm is the number of evaluations of the fitness function the algorithm executes until it finds an optimal solution. This is the usual complexity measure used when analysing evolutionary algorithms. Observe that the time needed by the algorithm for "bookkeeping", i.e. the creation of new individuals by mutation or crossover or the execution of the fitness function is usually disregarded.

A *mutation* of an individual changes it slightly at some random positions. For the classical case of bit-strings of length n, mutation is often performed by flipping each bit of an individual independently with probability $\frac{1}{n}$. As this might be infeasible for more complex representations, this behaviour must be simulated.

In [STW04], Scharnow, Tinnefeld and Wegener propose the following method to do this. First, a number s is chosen at random according to a Poisson distribution $\text{Pois}(\lambda = 1)$ with parameter $\lambda = 1$. An individual is then mutated by applying an *elementary mutation operator* $(s+1)$ times. The use of the Poisson distribution is motivated by the fact that it is the limit of the binomial distribution for n trials with probability $\frac{1}{n}$ each.

A *crossover* of two individuals combines parts of them to a new individual. We will only consider the so-called 1-point crossover. If the individuals are bit-strings of length n it is defined as follows. First pick a random position between 1 and n. Then merge the initial part of the first individual up to the chosen position with the ending part of the second individual starting from the chosen position.

3.2.2 Graphs and Shortest Paths

As both problems studied by us deal with shortest paths, we will now fix some notation.

Let $G = (V, E)$ be a directed graph with $n := |V|$ vertices and $m := |E|$ edges. Let $w\colon E \to \mathbb{N}$ be a function that assigns to each edge $e \in E$ a weight $w(e)$.

A *walk* from u to v is a sequence $u = v_0, v_1, \ldots, v_k = v$ of vertices such that $(v_{i-1}, v_i) \in E$ for all $i \in [1, \ldots, k]$. The walk is called *path* if it contains each vertex at most once. We will usually describe a walk by the sequence $(e_1, \ldots, e_k)$,

$e_i = (v_{i-1}, v_i)$ of edges it traverses. The weight of a walk is defined as the sum of the weights of all its edges.

The Single Source Shortest Path Problem

Given a vertex $s \in V$ called "source", the *Single Source Shortest Path Problem* (SSSP problem) is the problem of finding a shortest path from s to all other vertices $v \in V$. A simple implementation of Dijkstra's famous algorithm [Dij59] solves the problem in time $O(n^2)$.

When analysing the SSSP problem we can easily restrict ourselves to the complete (bi-directed) graph K_n. For this we simply allow the weight $w(e) = \infty$ for previously not existing edges e. A problem instance then is given by the distance matrix $D = (d_{ij})_{1 \le i,j \le n}$ of the graph, where $d_{ij} = w((i, j)) \in \mathbb{N} \cup \{\infty\}$.

The All-Pairs Shortest Path Problem

The *All-Pairs Shortest Path Problem* (APSP problem) is defined as follows. Given a weighted graph $G = (V, E)$, compute a shortest path from every vertex $u \in V$ to every other vertex $v \in V$.

In general, a pair of vertices may be connected by more than one shortest path, and these different paths may consist of different numbers of edges. In our analysis of the APSP problem we will often deal with all shortest paths having at most a given number of edges. To ease the language, we introduce the following notation.

Definition 3.1. *Let $G = (V, E)$ be a graph and let $\ell \in \mathbb{N}$. We define*

$$V_\ell^2 := \{(u, v) \in V^2 \mid u \neq v \text{ and there exists a shortest path from } u \text{ to } v \text{ consisting of at most } \ell \text{ edges}\}.$$

3.2.3 Chernoff Bounds

As mentioned earlier, the optimisation time of an evolutionary algorithm not only depends on the input but also on the random decisions done by the algorithm. Thus it can be modelled as a random variable. To estimate the optimisation time one can bound the expected value of this random variable. However, depending on the variance of this variable, the expected value alone may not be as useful as it seems on first glance. Hence we not only want to bound the expected value of the random variable, but are also interested in giving a bound that holds "with high probability". The following definition formalises this idea.

Definition 3.2. *Let $X(n)$ be a random variable depending on n. We say that $X(n) \in O(f(n))$ holds* with high probability (w.h.p.) *if for any $c \in N$ there exists a constant c' such that*

$$\Pr[X(n) \leq c' f(n)] \geq 1 - \frac{1}{n^c}.$$

We say that $X(n) \in \Omega(f(n))$ holds with high probability (w.h.p.) *if for any $c \in N$ there exists a constant c' such that*

$$\Pr[X(n) \geq c' f(n)] \geq 1 - \frac{1}{n^c}.$$

In other words, the probability that X differs by more than a constant factor from f can be made arbitrarily small. To achieve such bounds we will mainly use so-called *Chernoff Bounds*. Those allow us to bound the probability that the sum of some independent random variables will vary too widely from the expected value. The following Theorem summarises some Chernoff Bounds that we will use in this Chapter. Those are classical results from probability theory and can be found in most textbooks on the topic, for example in [AS08].

Theorem 3.1 (Chernoff Bound)**.** *Let $X_1, \ldots, X_t$ be mutually independent 0-1-random variables with $\Pr[X_i = 1] = p$ and $\Pr[X_i = 0] = 1 - p$ for all i. Let $X := \sum_{i=1}^{t} X_i$. Then the following holds.*

a) For all $\delta \in (0, 1]$ it holds that

$$\Pr\left[X < (1-\delta)\mathbb{E}[X]\right] \leq \exp\left(-\frac{\mathbb{E}[X]\delta^2}{2}\right).$$

a') For all $\alpha < 1$ it holds that

$$\Pr\left[X < \alpha\mathbb{E}[X]\right] \leq \exp\left(-\frac{(1-\alpha)^2\mathbb{E}[X]}{2}\right).$$

b) For all $\delta > 0$ it holds that

$$\Pr\left[X \geq (1+\delta)\mathbb{E}[X]\right] \leq \exp\left(-\frac{\min\{\delta, \delta^2\}\mathbb{E}[X]}{3}\right).$$

c) For all $\beta > 1$ it holds that

$$\Pr\left[X \geq \beta\mathbb{E}[X]\right] < \left(\exp(\beta - 1)\beta^{-\beta}\right)^{\mathbb{E}[X]}.$$

Observe that the Chernoff bounds a) and a') are basically the same, just substitute $\alpha = 1 - \delta$. However, we will need both variants of this bound and by listing them here we can save some calculations later.

While the above bounds allow us to handle sums of $0-1$ variables, we will later also need a bound for sums of geometrically distributed variables. For this we prove the following Chernoff Bound.

Theorem 3.2. *Let $X_i, i \in [1, \dots, n]$, be independent geometrically distributed random variables with $\Pr[X_i = j] = (1-p)^{j-1}p$ for all $j \in \mathbb{N}$ where $p \in (0,1)$. Let $X := \sum_{i=1}^{n} X_i$, and and $\delta > 0$. Then it holds that*

$$\Pr[X \geq (1+\delta)\mathbb{E}[X]] \leq \exp\left(-\frac{\delta^2}{2}\frac{(n-1)}{1+\delta}\right).$$

Proof. Let $Y_1, Y_2, \dots$ be an infinite sequence of identically distributed independent, biased, binary random variables such that Y_i is one with probability $\Pr[Y_i = 1] = p$ and zero with probability $\Pr[Y_i = 0] = 1 - p$. Note that the random variable "smallest j such that $Y_j = 1$" has the same distribution as each X_i. In consequence, X has the same distribution as "smallest j such that exactly n of the variables $Y_1, \dots, Y_j$ are one". In particular it holds that

$$\forall j \in \mathbb{N} : \Pr[X \geq j] = \Pr\left[\sum_{i=1}^{j-1} Y_i \leq n-1\right].$$

This manipulation reduces our problem to the analysis of independent Bernoulli trials and will enable us to use the classical Chernoff bounds.

The expected value of each X_i is $\mathbb{E}[X_i] = \frac{1}{p}$, thus $\mathbb{E}[X] = \frac{n}{p}$. Let $Y := \sum_{i=1}^{\lceil (1+\delta)\mathbb{E}[X]-1 \rceil} Y_i$. By the above,

$$\Pr[X \geq (1+\delta)\mathbb{E}[X]] = \Pr[Y \leq n-1].$$

The expected value of Y is bounded by

$$\mathbb{E}[Y] = \lceil (1+\delta)\mathbb{E}[X] - 1 \rceil p \geq (1+\delta)n - p > (1+\delta)(n-1).$$

Now let $\delta' := 1 - \frac{n-1}{\mathbb{E}[Y]}$. Then $0 < \delta' \leq 1$ and $\Pr[Y \leq n-1] = \Pr[Y \leq (1-\delta')\mathbb{E}[Y]]$. Hence we can apply the Chernoff bound from Theorem 3.1a) to get

$$\begin{aligned}
\Pr[X \geq (1+\delta)\mathbb{E}[X]] &= \Pr[Y \leq (1-\delta')\mathbb{E}[Y]] \\
&\leq \exp\left(-\frac{1}{2}\mathbb{E}[Y](1 - \frac{n-1}{\mathbb{E}[Y]})^2\right) \\
&\leq \exp\left(-\frac{1}{2}\mathbb{E}[Y](1 - \frac{1}{1+\delta})^2\right) \\
&\leq \exp\left(-\frac{1}{2}(n-1)(1+\delta)(\frac{\delta}{1+\delta})^2\right).
\end{aligned}$$

□

The Chernoff Bounds allow us to deal with sums of independent random variables. However, this is not enough. In our proofs we will often encounter sums of dependent random variables. For those the above Theorems cannot be applied directly.

To still be able to apply Chernoff Bounds we use the following Lemma. It allows us to approximate sums of dependent random variables by sums of independent random variables. Those can then be handled using Chernoff Bounds.

Lemma 3.1. *For $t \in \mathbb{N}$ let $X_1, \ldots, X_t \in \mathbb{N}_0$ be arbitrary random variables taking non-negative integers as values. Let $X_1^*, \ldots, X_t^* \in \mathbb{N}_0$ be mutually independent non-negative random variables such that for all $i \in [1, \ldots, t]$, X_i^* is independent of $X_1, \ldots, X_{i-1}$. Then for all $k \geq 0$ the following holds.*

a) If for all $i \in [1, \ldots, t]$, all $x_1, \ldots, x_{i-1} \in \mathbb{N}_0$ and all $m > 0$

$$\Pr[X_i = m | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}] \leq \Pr[X_i^* = m],$$

then

$$\Pr\left[\sum_{i=1}^{t} X_i \geq k\right] \leq \Pr\left[\sum_{i=1}^{t} X_i^* \geq k\right].$$

b) If for all $i \in [1, \ldots, t]$, all $x_1, \ldots, x_{i-1} \in \mathbb{N}_0$ and all $m > 0$

$$\Pr[X_i = m | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}] \geq \Pr[X_i^* = m],$$

then

$$\Pr\left[\sum_{i=1}^{t} X_i \geq k\right] \geq \Pr\left[\sum_{i=1}^{t} X_i^* \geq k\right].$$

c) If for all $i \in [1, \ldots, t]$, all $x_1, \ldots, x_{i-1} \in \mathbb{N}_0$ and all $m > 0$

$$\Pr[X_i = m | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}] \geq \Pr[X_i^* = m],$$

then

$$\Pr\left[\sum_{i=1}^{t} X_i < k\right] \leq \Pr\left[\sum_{i=1}^{t} X_i^* < k\right].$$

Proof. Define $P_j^t := \Pr[\sum_{i=1}^{j} X_i + \sum_{i=j+1}^{t} X_i^* \geq k]$ for $j \in [1, \ldots, t]$. Also define $\mathcal{X}_k^t = \{(x_1, \ldots, x_t) \in \{0,1\}^t | \sum_{i=1}^{t} x_i = k\}$.

Now consider part a) of the Lemma. Since $\Pr[X_i = m | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}] \leq \Pr[X_i^* = m]$ for all $x_1, \ldots, x_{i-1}$ and $m > 0$, it follows that

$$\Pr[X_i \geq m | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}] \leq \Pr[X_i^* \geq m]$$

holds for all non-negative m.

Comparing the outcome of the sequence of events $X_1, \ldots, X_j, X^*_{j+1}, \ldots, X^*_t$ with the outcome of $X_1, \ldots, X_{j+1}, X^*_{j+2}, \ldots, X^*_t$ for $j \in [1, \ldots, t]$ gives

$$
\begin{aligned}
P^t_{j+1} &= \Pr\left[\sum_{i=1}^{j+1} X_i + \sum_{i=j+2}^{t} X^*_i \geq k\right] \\
&= \Pr\left[\sum_{i=1}^{j} X_i + \sum_{i=j+2}^{t} X^*_i \geq k\right] + \\
&\quad \sum_{m=1}^{k}\left(\Pr\left[\sum_{i=1}^{j} X_i + \sum_{i=j+2}^{t} X^*_i = k-m\right] \cdot \Pr[X_{j+1} \geq m]\right) \\
&= \Pr\left[\sum_{i=1}^{j} X_i + \sum_{i=j+2}^{t} X^*_i \geq k\right] + \\
&\quad \sum_{m=1}^{k} \sum_{(x_1,\ldots,x_j,x_{j+2},\ldots,x_t)\in\mathcal{X}^{t-1}_{k-m}} \Pr\left[X_1 = x_1, \ldots, X_j = x_j\right] \cdot \\
&\quad \Pr\left[X_{j+1} \geq m | X_1 = x_1, \ldots, X_j = x_j\right] \cdot \prod_{i=j+2}^{t} \Pr\left[X^*_i = x_i\right] \\
&\leq \Pr\left[\sum_{i=1}^{j} X_i + \sum_{i=j+2}^{t} X^*_i \geq k\right] \\
&\quad + \sum_{m=1}^{k} \Pr\left[\sum_{i=1}^{j} X_i + \sum_{i=j+2}^{t} X^*_i = k-m\right] \cdot \Pr\left[X^*_{j+1} \geq m\right] \\
&= \Pr\left[\sum_{i=1}^{j} X_i + \sum_{i=j+1}^{t} X^*_i \geq k\right] \\
&= P^t_j.
\end{aligned}
$$

Thus, we have that

$$
\Pr\left[\sum_{i=1}^{t} X_i \geq k\right] = P^t_t \leq P^t_{t-1} \leq \cdots \leq P^t_1 \leq P^t_0 = \Pr\left[\sum_{i=1}^{t} X^*_i \geq k\right].
$$

To prove part b) observe that from $\Pr[X_i = m | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}] \geq \Pr[X^*_i = m]$ for all $x_1, \ldots, x_{i-1}$ and $m > 0$, it follows that

$$
\Pr[X_i \geq m | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}] \geq \Pr[X^*_i \geq m]
$$

holds for all non-negative m. With this the same computation as for a) holds with just the inequality reversed. Hence we get

$$\Pr\left[\sum_{i=1}^{t} X_i \geq k\right] = P_t^t \geq P_{t-1}^t \geq \cdots \geq P_1^t \geq P_0^t = \Pr\left[\sum_{i=1}^{t} X_i^* \geq k\right].$$

Inequality c) follows immediately from b) because

$$\begin{aligned}\Pr\left[\sum_{i=1}^{t} X_i < k\right] &= 1 - \Pr\left[\sum_{i=1}^{t} X_i \geq k\right] \\ &\leq 1 - \Pr\left[\sum_{i=1}^{t} X_i^* \geq k\right] \\ &= \Pr\left[\sum_{i=1}^{t} X_i^* < k\right].\end{aligned}$$

□

Later we need to bound the expected number of mutations that happen in Algorithm 3.1. However, we only need this for the special case of the $(\leq \mu + 1)$-EA, i.e. if the crossover probability $p_\otimes = 0$ is zero.

For this we use the following Chernoff bound on Poisson distributed random variables. The Lemma can be found as Theorem A.1.15 in [AS08].

Lemma 3.2. *Let P have Poisson distribution with mean λ. For $\epsilon > 0$*

$$\Pr\left[P \geq \lambda(1+\epsilon)\right] \leq \left(e^\epsilon (1+\epsilon)^{-(1+\epsilon)}\right).$$

As the number of mutations done by the $(\leq \mu + 1)$-EA is Poisson distributed, we can now easily bound the number of expected elementary mutations.

Lemma 3.3. *The expected number of elementary mutations that happen during k iterations of the $(\leq \mu + 1)$-EA is exactly $2k$. Furthermore, the probability that more than $3k$ mutations happen is less than $\left(\frac{e}{4}\right)^k$.*

Proof. By the definition of the $(\leq \mu + 1)$-EA the number of elementary mutations done in a single iteration is distributed as $\text{Pois}(\lambda = 1) + 1$. Due to the additivity of the Poisson distribution the number of elementary mutations done in k iterations has distribution $\text{Pois}(\lambda = k) + k$. Hence the expected number of elementary mutations is $2k$. Using Lemma 3.2, the probability that more than k extra mutations happen can be bound by $\Pr[\text{Pois}(\lambda = k) \geq 2k] \leq \left(\frac{e}{4}\right)^k$. □

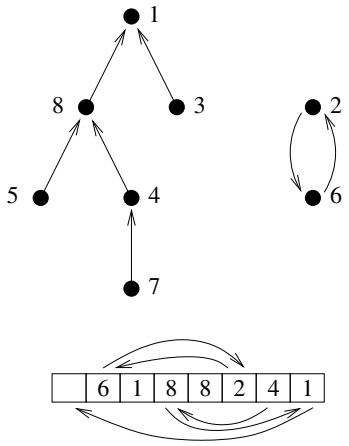

Figure 3.2: The graph representation and the vector representation of an individual not representing a tree. The fitness of vertex 7 is $d_{18} + d_{84} + d_{47}$, as represented by the arrows below the vector representation. The fitness of vertex 2 is ∞, since there is no path from vertex 2 to vertex 1, as shown by the arrows above the vector representation.

3.3 A Tight Analysis of the SSSP Problem

In this section we study a simple (1 + 1)-EA for the Single Source Shortest Path (SSSP) problem.

In [STW04], Scharnow, Tinnefeld and Wegener gave a first upper bound for this problem. We will derive tight upper and lower bounds on the optimisation time of the (1 + 1)-EA for the SSSP problem which hold both in expectation and with high probability. For this we will consider the same (1 + 1)-EA as studied in the above mentioned paper. During the analysis of the (1 + 1)-EA we will use a new technique that helps us to overcome the coupon collector behaviour of previously used arguments.

3.3.1 A Representation for the SSSP Problem

We want to analyse the (1 + 1)-EA for the SSSP problem introduced and studied in [STW04].

For this, consider the following variant of Algorithm 3.1. We set the crossover probability $p_\otimes = 0$ and consider only $\mu = 1$ individuals. This gives the classical (1 + 1) evolutionary algorithm ((1 + 1)-EA).

Note that the algorithm also works for undirected graphs. For each undirected edge $e = \{i, j\}$ simply set $d_{ij} = d_{ji} = w(e)$.

Individuals and Population

It is easy to see that we can choose shortest paths from s to any other vertex in such a way that the union of these paths forms a tree. Hence, we may represent solutions to the SSSP problem by giving for each vertex $i \in V$ its predecessor $p(i)$ on a shortest path from s to i. Without loss of generality, we will from now on always assume that the source vertex is vertex $s = 1$. Candidate solutions (i.e. individuals) for the evolutionary algorithm can then be represented as vectors of predecessors

$$\mathcal{I} = (p(2), \ldots, p(n)) \in \{1, \ldots, n\}^{n-1}.$$

Note that this representation does not imply that an individual forms a tree. See Figure 3.2 for an example.

Fitness and Selection for Replacement

To select the individuals, a multi-criteria fitness function is used. The fitness of an individual $\mathcal{I}$ is defined as $f(\mathcal{I}) := (f_2(\mathcal{I}), \ldots, f_n(\mathcal{I}))$ with

$$f_i(\mathcal{I}) := \begin{cases} \infty & \text{if } \mathcal{I} \text{ does not connect } s \text{ to } i, \\ w(P(s,i)) & \text{otherwise.} \end{cases}$$

Here, $w(P(s,i))$ is the cost of the path P from s to i induced by $\mathcal{I}$. If this path is $P = (s = v_1, v_2, \ldots, v_j = i)$ for $v_1, \ldots, v_j \in V$, then $w(P(s,i)) = d_{v_1 v_2} + \cdots + d_{v_{j-1} v_j}$. See again Figure 3.2 for an example.

The selection step accepts $\mathcal{I}'$ if $f(\mathcal{I}') \leq f(\mathcal{I})$, which is the case if $f_i(\mathcal{I}') \leq f_i(\mathcal{I})$ for all $2 \leq i \leq n$. Therefore, once we have found an optimal path for a vertex v, the (1 + 1)-EA does not accept mutations that would cause s to be connected to v using a sub-optimal path.

Mutation

At the beginning, the (1 + 1)-EA generates an initial individual $\mathcal{I}$ by assigning to each vertex $v \in V \setminus \{s\}$ a predecessor $p(v) \in V \setminus \{v\}$ uniformly at random. In the mutation step, $\mathcal{I}$ is modified to generate a new individual $\mathcal{I}'$. Then, a selection step is done replacing the individual $\mathcal{I}$ by $\mathcal{I}'$ if the fitness of $\mathcal{I}'$ is not worse than the one of $\mathcal{I}$. Mutation and selection are repeated as long as desired.

An elementary mutation of the vector $\mathcal{I}$ consists of randomly choosing a vertex $v \in V \setminus \{s\}$. Then, the predecessor $p(v)$ of v is set to a vertex chosen uniformly at random from $V \setminus \{v\}$. Obviously, there are $(n-1)^2$ possible ways to choose a vertex and its new predecessor. Hence this is the number of different elementary mutations that can be applied to an individual $\mathcal{I}$.

3.3.2 The Upper Bound

In this section we show that the optimisation time of the (1 + 1)-EA for the SSSP problem is $O(n^2 \max\{\log(n), \ell\})$ with high probability. Here, ℓ is the maximum number of edges of all shortest paths with a minimum number of edges. This leads to the following definition.

Definition 3.3 (Edge radius)**.** *Let G be a weighted graph and s a vertex of G. Then the* edge radius $\ell_G(s)$ *of s in G is the minimum number r such that for all vertices $v \in V \setminus \{s\}$, there is a shortest path from s to v having at most r edges. In other words,*

$$\ell_G(s) := \max_{v \in V} \min\{\ell(P) \mid P \text{ is a shortest path from } s \text{ to } v\},$$

where $\ell(P)$ is the number of edges of the path P.

Now we can prove the upper bound.

Theorem 3.3. *The optimisation time of the* (1 + 1)*-EA is $O(n^2 \max\{\log(n), \ell\})$ with high probability.*

This follows immediately from the following Lemma.

Lemma 3.4. *Let $\ell := \ell_G(s)$ be the edge radius of s in G, $\ell^* = \max\{\ell, \log(n)\}$, $\lambda \geq 2$ and $t = \lambda(n-1)^2\ell^*$. Then the optimisation time needed by the* (1 + 1)*-EA to find all shortest paths is less than t with probability $p > 1 - n^{1-\frac{\lambda}{8}}$.*

Proof. Because of the multi-criteria fitness function, the (1 + 1)-EA cannot replace any path in the individual $\mathcal{I}$ by a longer path. Thus, any successful mutation step that would apply more than one mutation can be simulated by a number of successful mutation steps applying a single mutation. Since the probability for such a single mutation step is constant, it suffices to assume that only single mutation steps are applied for the sake of the upper bound analysis.

The (1 + 1)-EA has to find $n - 1$ shortest paths from the source s to all other vertices vertex v. Note that there may be many different possible shortest paths for a vertex v.

Pick a vertex v and a shortest path $P := (v^1, \ldots, v^{\ell'+1})$ from $s = v^1$ to $v = v^{\ell'+1}$. Note that by the definition of the edge radius ℓ we can pick P so that it has $\ell' \leq \ell$ edges. We call a single mutation step the *j-th improvement* of P if prior to the mutation the individual $\mathcal{I}$ contains a shortest path from s to v^j for some $j \in [1, \ldots, \ell']$ and after the mutation step the predecessor of v^{j+1} is $p(v^{j+1}) = v^j$. Hence, after the j-th improvement, $\mathcal{I}$ contains a shortest path from s to v^{j+1}. Note that $\mathcal{I}$ might have already contained such a shortest path before. If the (1 + 1)-EA has performed the ℓ'-th improvement, we say that it has *followed* P. Obviously, a shortest path from s to $v = v^{\ell'+1}$ has been found by then.

Let t' be the number of steps the (1 + 1)-EA needs to follow P. Define the random variables X_i for $i \in [1, \ldots, t']$ as $X_i = 1$ if the i-th mutation step is an improvement of P and $X_i = 0$ otherwise. Then $\Pr[X_i = 1] \geq p = \frac{1}{(n-1)^2}$, since either the corresponding predecessor was already set correctly before or the i-th mutation step picks the correct vertex with probability $\frac{1}{n-1}$ and sets it to its correct predecessor with probability $\frac{1}{n-1}$. For $i > t'$, define the random variables X_i by $\Pr[X_i = 1] := p$ and $\Pr[X_i = 0] := 1 - p$. Since we only consider single mutation steps, the X_i are mutually independent. Hence, the expected value of $X := \sum_{i=1}^{t} X_i$ is $\mathbb{E}[X] \geq pt$.

If the (1 + 1)-EA has not found an optimal path from s to v, it obviously has not followed P and thus $X < \ell$. Hence, the probability of not finding a shortest path from s to v in time t can be bounded by

$$\begin{aligned}\Pr\begin{bmatrix}\text{no shortest path from}\\ s \text{ to } v \text{ found in time } t\end{bmatrix} &\leq \Pr\begin{bmatrix}P \text{ not followed}\\ \text{in time } t\end{bmatrix}\\ &\leq \Pr[X < \ell]\\ &\leq \Pr[X < \ell^*].\end{aligned}$$

For $t = \lambda(n-1)^2\ell^*$ and $0 < \delta = \frac{(\lambda-1)}{\lambda} \leq 1$ we have

$$\begin{aligned}(1-\delta)\mathbb{E}[X] &\geq \left(1 - \frac{(\lambda-1)}{\lambda}\right)\frac{1}{(n-1)^2}\lambda(n-1)^2\ell^*\\ &= \ell^*.\end{aligned}$$

Hence, by Theorem 3.1a), we can bound the probability of not finding a shortest path from s to v in time $t = \lambda(n-1)^2\ell^*$ by

$$\begin{aligned}\Pr[X < \ell^*] &\leq \Pr[X < (1-\delta)\mathbb{E}[X]]\\ &\leq \exp\left(-\frac{\mathbb{E}[X]\delta^2}{2}\right)\\ &= \exp\left(-\frac{(\lambda-1)^2\ell^*}{2\lambda}\right)\\ &\leq \exp\left(-\frac{(\frac{\lambda}{2})^2\ell^*}{2\lambda}\right)\\ &\leq \exp\left(-\frac{\lambda}{8}\ell^*\right)\end{aligned}$$

for $\lambda \geq 2$.

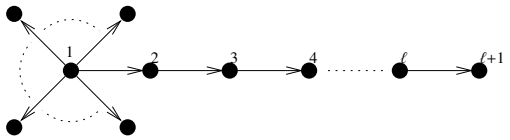

Figure 3.3: The SSSP tree of the worst-case graph $G_{n,\ell}$ with source v_1.

Using an union bound argument we get

$$\begin{aligned}
\Pr\left[\begin{array}{c}\text{not for all } v_i \text{ a shortest}\\ \text{path from } s \text{ to } v_i\\ \text{found in time } t\end{array}\right] &\leq \sum_{i=2}^{n} \Pr\left[\begin{array}{c}\text{no shortest path}\\ \text{from } s \text{ to } v_i\\ \text{found in time } t\end{array}\right]\\
&\leq \sum_{i=2}^{n} \exp\left(-\frac{\lambda}{8}\ell^*\right)\\
&\leq n\exp\left(-\frac{\lambda}{8}\log(n)\right)\\
&\leq n^{1-\frac{\lambda}{8}}.
\end{aligned}$$

Hence, by choosing λ appropriately, we can achieve a failure probability of $O(n^{-c})$ for any c. Note that we did not optimise for λ. □

3.3.3 The Lower Bound

In this section we show a lower bound that matches the upper bound presented in the previous section. For this, we will define a worst-case graph $G_{n,\ell}$ for all $n \in \mathbb{N}$ and $\ell \in [1,\ldots,n-1]$ having edge radius $\ell_{G_{n,\ell}}(s) = \ell$. We will then show that with high probability the (1 + 1)-EA has an optimisation time of at least $\Omega(n^2 \max\{\log(n), \ell\})$ for this graph.

A Worst Case Graph Class

Let $n \in \mathbb{N}$, $V = [1,\ldots,n]$. For all $\ell \in [1,\ldots,n-1]$ we must define $G_{n,\ell} = (V,E)$ such that the source of the SSSP tree to be computed is $s = 1$ and the edge radius of s is $\ell_{G_{n,\ell}}(s) = \ell$.

To enforce worst-case behaviour of the (1 + 1)-EA, we need to ensure that finding the shortest path to the vertex maximising the edge radius is hard. For this, we set the weights in such a way that $(1, 2, \ldots, \ell, \ell+1)$ is the unique shortest path from $s = 1$ to $\ell + 1$. For all other vertices k with $k > \ell + 1$, the edge (s, k) shall be the unique shortest path from s to k. Figure 3.3 shows the SSSP tree of $G_{n,\ell}$. For simplicity, we assign the weight 1 to all edges in the SSSP tree.

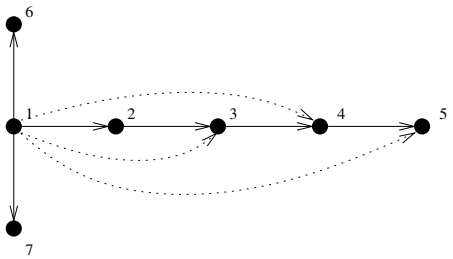

Figure 3.4: The graph $G_{7,3}$. The solid edges have weight one and form the shortest path tree. All other edges have weight 14. The dotted edges are the second-best shortest path from 1 to vertices 3, 4 and 5.

To guarantee that the optimisation time depends linearly on ℓ, the graph must contain the right weights for edges that do not belong in the SSSP tree. For each vertex $i \in [2, \ldots, \ell + 1]$ on the shortest path maximising the edge radius, we should ensure that its edge with s is not too expensive. As long as its predecessor $i - 1$ is not yet connected via the unique shortest path, this will ensure that it is cheaper to connect s and i directly than to connect s to i via $i - 1$. This will enforce that most of the edges on the shortest path are added in the order they appear in the shortest path. The formal definition of the edge weights $w(i, j), i, j \in [1, \ldots, n], i \neq j$ is then as follows.

$$w(i,j) := \begin{cases} 1, & \text{if} \quad j = i + 1 \leq \ell + 1, \\ 1, & \text{if} \quad i = 1 \wedge j > \ell + 1, \\ 2n, & \text{otherwise.} \end{cases}$$

A concrete example is given in Figure 3.4 for the graph $G_{7,3}$ with seven vertices and edge radius 3. Note that $G_{n,1}$ is the graph with edge weight 1 for each edge $(s, i), i \in [2, \ldots, n]$ and $2n$ for all other edges.

A Lower Bound

We now give a lower bound on the number of steps needed by the (1 + 1)-EA depending on n and ℓ. To prove that $\Omega(n^2 \max\{\log(n), \ell\})$ is a lower bound on the optimisation time of the (1 + 1)-EA, we first prove that $\Omega(n^2 \log(n))$ is a lower bound on the optimisation time. Observe that this bound does not depend on our special graph $G_{n,\ell}$ but holds for all graphs that have a unique SSSP tree.

Lemma 3.5. *Let $G = (V, E)$ be a graph on n vertices. Let $s \in V$ be such that the SSSP tree of G with source s is unique. Then the number of steps needed by the (1 + 1)-EA to find the SSSP tree of G with source s is $\Omega(n^2 \log(n))$ with high probability.*

Proof. Since the shortest path tree is unique, each vertex has a unique predecessor. Hence, the (1 + 1)-EA has found the optimal solution as soon as for all $v \in V \setminus \{s\}$ the predecessor pointer $p(v)$ is set to this unique predecessor. This process can be modelled as a coupon collector process.

In this proof, we say that v is *fine* after iteration t, if $p(v)$ points to the desired predecessor during any of the t iterations. Note that if the predecessor is not already connected to s via a shortest path, v might change its predecessor to a different vertex than the one in the shortest path tree. Hence, the time until all vertices are fine is clearly a lower bound for the optimisation time.

After the initialisation, each vertex is fine with probability exactly $\frac{1}{n-1}$. In consequence, with probability $1 - \exp(-\Theta(n))$, less than $\frac{n}{2}$ vertices are fine. We call the remaining $k \geq \frac{n}{2}$ vertices *interesting*.

Assume that the (1 + 1)-EA runs for $T = \frac{1}{12}(n-1)^2 \ln(\frac{n}{2})$ iterations after the initialisation. By Lemma 3.2, the algorithm applies at most $3T$ elementary mutations during these T iterations (apart from a super-exponentially small failure probability).

For an elementary mutation, the probability of choosing an interesting vertex and correctly setting its unique predecessor is $\left(\frac{k}{n-1}\right)\left(\frac{1}{n-1}\right)$. Hence, a sequence of $3T$ elementary mutations produces at most $6T\frac{k}{(n-1)^2}$ good events with probability $1 - \exp(\Theta(n \log n))$. Each of this series of good events can be viewed as the action of buying one out of k different coupons. By the coupon collector Theorem (see for example [MR95]), these $6T\frac{k}{(n-1)^2} \leq \frac{1}{2}k\ln(k)$ trials do not suffice to obtain all k coupons. This holds with probability $1 - \exp(-\Theta(n))$. □

By the above Lemma, our lower bound is tight as long as $\ell \in O(\log(n))$. To complete our claim, however, we need to prove that for larger ℓ the optimisation time linearly depends on ℓ.

Lemma 3.6. *Let $n \in \mathbb{N}$ and $\ell \in \omega(\log(n))$. Then the optimisation time of the (1 + 1)-EA on $G_{n,\ell}$ is $\Omega(n^2\ell)$ with high probability.*

Proof. The idea of this proof is similar to the one used in [DJW02] for the proof of the lower bound on the runtime of the (1 + 1)-EA on the leading ones function. To prove the claim, we analyse how long it takes until the individual $\mathcal{I}$ contains the path $P := (s = 1, 2, \ldots, \ell, \ell + 1)$. To this aim, we analyse how the length $L(\mathcal{I})$ of the longest subpath of P starting in s that is contained in $\mathcal{I}$ grows. Note that this length $L(\mathcal{I})$ never decreases, since for each vertex on P this subpath is the unique shortest path to s.

The proof basically proceeds by showing the following three claims. (i) Initially $L(\mathcal{I})$ is constant. (ii) In $\Theta(n^2\ell)$ iterations $L(\mathcal{I})$ increases at most $O(\ell)$ times. (iii) The total increase in these $O(\ell)$ relevant iterations (plus the initial constant length) is less than ℓ.

The probability that in the initial individual some vertex $i \in [2, \dots, \ell + 1]$ is already linked to $i-1$ is exactly $\frac{1}{n-1}$. Hence the probability that $L(\mathcal{I}) \geq c$ is $O(n^{-c})$ for all constants $c \in \mathbb{N}$, that is, with high probability $L(\mathcal{I})$ is initially constant.

Let t^* be the time step in which $L(\mathcal{I})$ increases to the maximal possible value of ℓ. For $i \in [1, \dots, t^*]$, we define a binary random variable X_i by $X_i = 1$ if $L(\mathcal{I})$ increases in step i. To increase $L(\mathcal{I})$, one of the $S + 1$ elementary mutations in the current step has to connect vertex $L(\mathcal{I}) + 2$ to vertex $L(\mathcal{I}) + 1$. The probability that an elementary mutation succeeds in doing so is $\frac{1}{(n-1)^2}$. By Lemma 3.3, the probability that one iteration does so is at most $\frac{2}{(n-1)^2}$. Hence we have

$$\Pr[X_i = 1] \leq p := \frac{2}{(n-1)^2}.$$

For $i > t^*$ define X_i by $\Pr[X_i = 1] := p$ and $\Pr[X_i = 0] := 1 - p$, independent of all other random variables.

Let $t := \eta(n-1)^2\ell$, where η is a constant to be chosen later. Let $X_1^*, \dots, X_t^*$ be mutually independent random variables with $\Pr[X_i^* = 1] := p$ and $\Pr[X_i^* = 0] := 1 - p$ for all i. Then

$$\Pr[X_i^* = 1] \geq \Pr[X_i = 1 | X_1 = x_1, \dots X_{i-1} = x_{i-1}]$$

holds for all $x_1, \dots, x_{i-1} \in \{0, 1\}$. For the sum $X^* := \sum_{i=1}^t X_i^*$ the expected value is

$$\mathbb{E}[X^*] = pt = \eta(n-1)^2\ell\frac{2}{(n-1)^2} = 2\eta\ell.$$

Hence, applying part a) of Lemma 3.1 and using the Chernoff bound b) from Theorem 3.1 with $\delta = 1$, we get

$$\begin{aligned}
\Pr\Big[\sum_{i=1}^{t} X_i \geq 4\eta\ell\Big] &\leq \Pr[X^* \geq 2\mathbb{E}[X^*]] \\
&\leq \exp\left(-\frac{\mathbb{E}[X^*]}{3}\right) \\
&= \exp\left(-\frac{2\eta\ell}{3}\right) \\
&= \exp\left(-\frac{2\eta}{3}\log(n)\frac{\ell}{\log(n)}\right) \\
&= n^{-\frac{2\eta}{3}\frac{\ell}{\log(n)}} \\
&= n^{-\omega(1)}.
\end{aligned}$$

In the last lines we used that since $\ell = \omega(\log(n))$ we have $\frac{\eta\ell}{\log n} = \omega(1)$ for any constant η. Since $\sum_{i=1}^t X_i$ is an upper bound on the number of improvements in the

first t iterations, this means that with high probability the (1 + 1)-EA did at most $t' := 4\eta\ell$ improvements during these t iterations.

Finally, we analyse which value of $L(\mathcal{I})$ results from these t' improvements. Clearly, each improvement increases $L(\mathcal{I})$ by at least one. However, there are two ways how an additional vertex i can be connected to the longest subpath of P starting in s. One is that an elementary mutation in the iteration causes the improvement that changes the pointer of i to its predecessor $i-1$ in P. The other is that i is coincidentally connected to $i-1$ and $i-1$ itself becomes part of the subpath by one of the two ways. We shall argue that both events happen only with a probability of at most $\frac{1}{2}$.

Suppose first that i is added to the path of interest through an elementary mutation. For this to happen (among other things), the following has to occur. Among the possibly more than one elementary mutations in the current step that connect i to some other vertex, the last one has to connect i to its predecessor in P. By definition of the mutation operator, this happens with a probability of $\frac{1}{n-1} \leq \frac{1}{2}$.

Now consider the case that $i-1$ becomes part of the subpath of interest. We argue that the probability that i is coincidentally connected to $i-1$ is at most the probability that it is pointing to s and in consequence, at most $\frac{1}{2}$. As all vertices in the initial individual have equal probability of being the predecessor of i, this obviously holds for the initial individual.

Now consider an iteration that does not result in making $i-1$ part of the subpath of interest. Fix a sequence of elementary mutations to be conducted in this iteration. Assume that at the start of the iteration vertex i has some predecessor $j \in [1,\dots,t]$ for which the path from s in $\mathcal{I}$ has length $w(s,j)$. There are two possibilities. If at the end of the iteration $i-1$ is further from s as j, i.e. $w(s,i-1) > w(s,j)$, the algorithm would not accept $i-1$ as new predecessor but it would accept s. On the other hand, if at the end of the iteration we have $w(s,i-1) \leq w(s,j)$, then changing is predecessor to $i-1$ would be acceptable. But so would be changing it to s due to the construction of the edge weights with both possibilities being equally likely. Summing over both possibilities, we see that choosing s as predecessor of i is more likely than choosing $i-1$. In consequence, the probability that i coincidentally points at $i-1$ in the iteration in which $i-1$ becomes part of the subpath of interest, is at most $\frac{1}{2}$.

Summarising, an additional vertex could be coincidentally connected with probability $p_1 \leq \frac{1}{2}$. With probability $1-p_1$ this is not the case and it may become connected by an elementary mutation which also has probability $p_2 \leq \frac{1}{2}$. Hence, the probability that an additional vertex becomes connected is at most

$$p_1 + (1-p_1)p_2 \leq \frac{1}{2}p_1 + \frac{1}{2} \leq \frac{3}{4}.$$

Let t'' be the number of improvement steps the (1 + 1)-EA performs until it

finds the optimal solution. Recall that with high probability, $t'' \leq t' = 4\eta\ell$. For $i \in [1, \ldots, t'']$ let Y_i be the random variable describing the total number of vertices which are added in the i-th improvement step. By the above arguments, independent of the outcome of previous random choices, we have

$$\Pr[Y_i = m] \leq \left(\frac{3}{4}\right)^{m-1} \frac{1}{4}$$

for all $m \geq 1$.

If $t'' < t' = 4\eta\ell$, let Y_i for $i \in [t'' + 1, \ldots, t']$ be defined by $\Pr[Y_i = m] := (\frac{3}{4})^{m-1}\frac{1}{4}$ independent from all other Y_j. Define Y_i^* for $i \in [1, \ldots, t']$ to be mutually independent random variables that are geometrically distributed with parameter $q = \frac{1}{4}$, that is, $\Pr[Y_i^* = m] := (\frac{3}{4})^{m-1}\frac{1}{4}$ for all $m \geq 1$. The expected value of Y_i^* is $\mathbb{E}[Y_i^*] = q^{-1} = 4$. Let $Y := \sum_{i=1}^{t'} Y_i$ and $Y^* := \sum_{i=1}^{t'} Y_i^*$. Then $\mathbb{E}[Y^*] = 4t'$.

Applying part a) of Lemma 3.1 and the Chernoff bound for geometrically distributed random variables from Theorem 3.2 with $\delta = 1$ and assuming $t' \geq 2$ we get

$$\begin{aligned}
\Pr[Y \geq 8t'] &\leq \Pr[Y^* \geq 8t'] \\
&= \Pr[Y^* \geq 32\eta\ell] \\
&\leq \exp\left(-\frac{(t'-1)}{4}\right) \\
&\leq \exp\left(-\frac{t'}{2 \cdot 4}\right) \\
&= \exp\left(-\frac{\eta\ell}{2}\right) \\
&= \exp\left(-\frac{\eta}{2}\log(n)\frac{\ell}{\log(n)}\right) \\
&= n^{-\frac{\eta}{2}\frac{\ell}{\log(n)}} \\
&= n^{-\omega(1)}.
\end{aligned}$$

Thus, with high probability during up to $t' = 4\eta\ell$ improvements at most $32\eta\ell$ additional vertices become part of the shortest path. Choosing $\eta = \frac{1}{64}$, we see that with high probability, $L(\mathcal{I})$ is at most $c + 32\eta\ell = c + \frac{1}{2}\ell < \ell$ after $t = \eta(n-1)^2\ell$ iterations, that is, the path P hast not been found in this time. □

Combining Lemma 3.5 and Lemma 3.6 yields the following theorem.

Theorem 3.4 (Lower Bound)**.** *The optimisation time needed by the* (1 + 1)*-EA to solve the SSSP problem is* $\Omega(n^2 \max\{\log(n), \ell\})$ *with high probability.*

3.4 Crossover and the APSP Problem

After having studied the (1 + 1)-EA for the single-source shortest path problem we will now consider the all-pairs shortest path problem. Here we will not only study a simple evolutionary algorithm but also analyse the influence of crossover on this problem. As a result we will show that adding crossover to the evolutionary algorithm for this problem indeed improves its optimisation time.

3.4.1 Components of the Genetic Algorithm

We now discuss the ingredients needed to apply our framework given in Algorithm 3.1 to the APSP problem.

Individuals and Population

As discussed in Subsection 3.2.1 genetic algorithms usually keep a population of individuals, which is gradually improved. In the APSP problem we are aiming for a population containing a shortest path for each pair of distinct vertices. Hence it makes sense to allow paths or walks in the graph $G = (V, E)$ as individuals. To have more freedom in defining the crossover operator, an individual will simply be a sequence of edges, $(e_1, \ldots, e_k), e_1, \ldots e_k \in E, k \in \mathbb{N}$. However, the replacement operator will ensure that only individuals that are walks can enter the population.

For the APSP problem, a natural choice for the initial population is the set $\mathcal{I} := \{(e) \mid e \in E\}$ of all paths consisting of one edge.

Fitness and Selection for Replacement

To apply the algorithmic framework given in Algorithm 3.1 we also need a proper fitness function and a selection mechanism for our individuals.

The natural choice for the fitness of an individual is the length of the walk it represents. This value should be minimised by the algorithm. As a result of the crossover operations discussed below, we may however generate individuals that aren't walks. For such individuals their fitness is defined as ∞. This will ensure that they are never included in the population.

In our model of the APSP problem diversity is an issue as we need to ensure that the final solution contains one path for each pair of vertices. With the fitness function chosen above, shortest paths for vertices that are far apart will always seem unfitter than those for vertices that are close. Hence we must ensure that we don't end up with a population that only contains many (near-)optimal individuals for a small set of nearby vertices. One could try to achieve this by choosing a non-strict replacement mechanism. However, for our approach we rather enforce

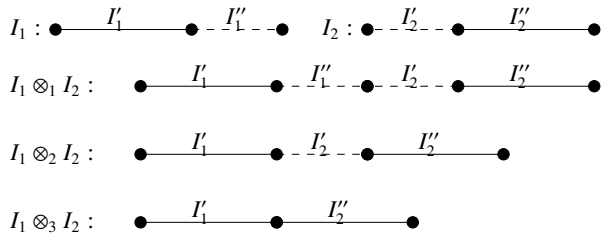

Figure 3.5: The effects of the three crossover operators.

this directly by ensuring that the algorithm will never eliminate all paths between a pair of vertices. Such an approach is called a *diversity mechanism*. Ensuring diversity this way, we can be strict in the replacement otherwise. In fact, for each pair (u, v) of vertices we eliminate all but the fittest individual connecting u to v. If a new individual with the same fitness is created for a pair of vertices, we will favour it over previously generated individuals. This is a form of *truncation selection*.

With a truncation selection operator guiding the replacement of individuals, it makes sense to select individuals as parents of mutation and crossover in a way that produces less selection pressure. We therefore choose these individuals uniformly at random from our population.

Mutation and Crossover

The last missing ingredients for Algorithm 3.1 is the definition of the variation operators for the APSP problem.

Considering our individuals, it seems natural to define the following *elementary mutation* operator. Let $(u, v) \in E$ be the first edge of the individual $\mathcal{I}$ and $(u', v') \in E$ be the last edge. Pick an edge e from the set of all edges incident to u or v' uniformly at random. If this edge is (u, v) or (u', v'), remove it from the individual. Otherwise, append the edge at the corresponding end of the individual. Observe that this could erase an individual consisting of a single edge (u, v). Hence, in this case, we will pick an edge uniformly at random from the set of all edges incident to u or v except (u, v) and append it.

Since our individuals differ in length, we cannot simply apply the same definition for the 1-point crossover as in the bit-string case. Instead, we propose the following three variants of the 1-point crossover operator that combine two individuals $\mathcal{I}_1, \mathcal{I}_2$ consisting of ℓ_1 and ℓ_2 edges respectively.

The crossover operator $\otimes_1$ simply combines both individuals by appending $\mathcal{I}_2$ to $\mathcal{I}_1$. The second operator, $\otimes_2$, chooses a random number $i \in [0, \ldots, \ell_1]$ and appends $\mathcal{I}_2$ to the first i edges of $\mathcal{I}_1$. Finally, the operator $\otimes_3$ chooses two random numbers $i \in [0, \ldots, \ell_1]$ and $j \in [0, \ldots, \ell_2]$. The new individual created by this

operator consists of the first i edges of $\mathcal{I}_1$ and the last $\ell_2 - j$ edges of $\mathcal{I}_2$. In Figure 3.5 the effects of the three crossover operators are depicted.

Observe that, unlike mutation, crossover may combine two individuals representing walks to a new individual that no longer represents a walk, and hence has infinite fitness.

3.4.2 Upper Bound for the $(\leq \mu + 1)$-EA

The main ideas to prove the upper bound of $O(n^4)$ for the $(\leq \mu + 1)$-EA are as follows. Being pessimistic, we may assume that shortest paths are found exclusively by adding edges to already found shortest paths. More specifically, we assume that only a single edge is added in each iteration. Then, to find a shortest path from u to v for $(u, v) \in V_\ell^2$, it suffices that the $(\leq \mu + 1)$-EA chooses ℓ times the adequate shortest path already in the solution. Each time it has to add the appropriate edge to enlarge it. Choosing the right path has probability $O(n^{-2})$) and choosing the appropriate edge to add has probability $O(n^{-1})$. If $\ell \geq \log n$, the time needed for this is that sharply concentrated around the mean of $\Theta(\ell n^3)$, that we may use a union bound argument over all $(u, v) \in V_\ell^2$.

Lemma 3.7. *Let $\ell \geq \log(n)$. Within $O(\ell n^3)$ steps, the $(\leq \mu + 1)$-EA finds with high probability a shortest path from u to v for all $(u, v) \in V_\ell^2$.*

Proof. Let $(u, v) \in V_\ell^2$. We first analyse the probability that a shortest path from u to v is not found within a certain time. For the analysis, we fix a path $P = ((u, v_1), (v_1, v_2), \ldots, (v_{\ell'-1}, v_{\ell'} = v))$ of length $\ell' \leq \ell$. Note that P will be a technical tool only and we do not aim at finding this particular path.

In the following, we shall only consider mutation steps that perform a single elementary mutation. According to the properties of the Poisson distribution a mutation consists of a single elementary mutation with probability $\frac{1}{e}$.

We call a mutation step the *j-th pessimistic improvement* in P if the following holds. (i) The mutation creates a shortest path from u to v_{j+1} out of a shortest path from u to v_j that is already in the population. (ii) The pessimistic improvements $1, \ldots, j-1$ have already been done. Note that this implies that pessimistic improvements appear in ascending order. Obviously, when the $(\leq \mu + 1)$-EA has performed the $(\ell' - 1)$-st pessimistic improvement in P, a shortest path from u to v has been found.

Let the random variable t' denote the number of steps the $(\leq \mu + 1)$-EA executes until it performs the $(\ell' - 1)$-st pessimistic improvement in P. For $i \in [1, \ldots, t']$ define the random variable X_i by $X_i = 1$ if the i-th mutation step is a pessimistic improvement in P and $X_i = 0$ otherwise. Then

$$\Pr[X_i = 1] \geq \frac{1}{e} \frac{1}{n(n-1)^2} > \frac{1}{e} \frac{1}{n^3} =: p,$$

independent of the first $i-1$ steps, since the probability to pick the correct individual is at least $\frac{1}{n(n-1)}$ and the probability to pick the correct edge is at least $\frac{1}{n-1}$. For every $i > t'$ we independently define X_i by $\Pr[X_i = 1] = p$ and $\Pr[X_i = 0] = 1 - p$.

Let $t := e\eta\ell n^3$ for some $\eta > 2$. If the $(\leq \mu + 1)$-EA has not found a shortest path from u to v after t steps, it obviously has not performed the $(\ell' - 1)$-st pessimistic improvement in P, and thus $X := \sum_{i=1}^{t} X_i < \ell'$. For every $i \in [1, \ldots, t]$ the random variable X_i fulfils $\Pr[X_i = 1 | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}] \geq p$ for all $x_1, \ldots, x_{i-1} \in \{0, 1\}$. Define the mutually independent binary random variables X_i^* by $\Pr[X_i^* = 1] = p$ for $i \in [1, \ldots, t]$. For some vertices $u, v \in V$ define the set of all shortest paths between them as

$$\mathcal{P}_{uv} := \{P \mid P \text{ is shortest path from } u \text{ to } v\}.$$

Using Lemma 3.1 c) and the Chernoff bound from Theorem 3.1 a') with

$$\alpha := \frac{\ell'}{E[X^*]} \leq \frac{\ell}{pt} = \frac{1}{\eta}$$

the probability of not finding a shortest path from u to v in t steps can be bound by

$$\begin{aligned}
\Pr\left[\text{no } P \in \mathcal{P}_{uv} \text{ found in } t \text{ steps}\right] &\leq \Pr[X < \ell'] \\
&\leq \Pr[X^* < \ell'] \\
&= \Pr[X^* < \alpha \mathbb{E}[X^*]] \\
&\leq \exp(-\tfrac{1}{2}(1-\alpha)^2 \mathbb{E}[X^*]) \\
&\leq \exp(-\tfrac{1}{2}(1-\alpha)^2 pt) \\
&\leq \exp(-\tfrac{1}{8}\eta\ell).
\end{aligned}$$

With this inequality we can now bound the probability that the $(\leq \mu + 1)$-EA does not succeed in t steps. For this we use a simple union bound argument to bound the probability of not finding a shortest path for all vertex pairs $(u, v) \in V_\ell^2$ in t steps as

$$\begin{aligned}
&\Pr\left[\exists (u,v) \in V_\ell^2 : (\nexists P \in \mathcal{P}_{uv} : P \text{ found in } t \text{ steps})\right] \\
&\leq \sum_{(u,v) \in V_\ell^2} \Pr\left[\text{no } P \in \mathcal{P}_{uv} \text{ found in } t \text{ steps}\right] \\
&\leq n(n-1)\exp(-\tfrac{1}{8}\eta\ell) \qquad (3.1) \\
&< n^2 \exp(-\tfrac{1}{8}\eta \log(n)) \\
&= n^{2-\frac{\eta}{8}}.
\end{aligned}$$

For any constant k we can choose $\eta := 8(k+2)$. Thus, with probability $1 - O(n^{-k})$ the optimisation time is at most $e\eta\ell n^3$. Note that we did not try to optimise the constant η. □

For $\ell = n - 1$, Lemma 3.7 yields the following upper bound.

Theorem 3.5. *With high probability, the optimisation time of the* $(\leq \mu + 1)$*-EA is* $O(n^4)$.

We can also derive an expected optimisation time of $O(n^4)$ from the strong concentration bound of equation (3.1) in Lemma 3.7.

Theorem 3.6. *Let* $\ell \geq \log(n)$. *The expected number of steps until the* $(\leq \mu + 1)$*-EA finds a shortest path from* u *to* v *for all* $(u, v) \in V_\ell^2$ *is* $O(\ell n^3)$. *In particular it holds that the expected optimisation time of the* $(\leq \mu + 1)$*-EA is* $O(n^4)$.

Proof. Let t_ℓ be the number of steps until the $(\leq \mu + 1)$-EA has found a shortest path from u to v for all $(u, v) \in V_\ell^2$. In the proof of Lemma 3.7 we showed that the probability that t_ℓ is greater than $e\eta\ell n^3$ is $\Pr[t_\ell > e\eta\ell n^3] \leq n^{2-\frac{\eta}{8}}$. Let $\eta := \nu n^i$ for some $\nu \geq 24$ and $i \in \mathbb{N}$. Then,

$$\begin{aligned}\Pr[e\nu\ell n^{4+i} \geq t_\ell > e\nu\ell n^{3+i}] &\leq \Pr[t_\ell > e\nu\ell n^{3+i}] \\ &\leq n^{2-\frac{\nu n^i}{8}} \\ &\leq n^{2-3n^i}.\end{aligned}$$

For $n \geq 2$, the expected number of steps $\mathbb{E}[t_\ell]$ needed to find a shortest path from u to v for all $(u, v) \in V_\ell^2$ is thus

$$\begin{aligned}\mathbb{E}[t_\ell] &= \sum_{t'=1}^{\infty} t' \cdot \Pr[t_\ell = t'] \\ &\leq e\nu\ell n^3 + \sum_{i=0}^{\infty} \sum_{t'=e\nu\ell n^{i+3}+1}^{e\nu\ell n^{i+4}} t' \cdot \Pr[t_\ell = t'] \\ &\leq e\nu\ell n^3 + \sum_{i=0}^{\infty} e\nu\ell n^{i+4} \cdot n^{2-3n^i} \\ &= e\nu\ell n^3 (1 + \sum_{i=0}^{\infty} n^{i+3-3n^i}) \\ &\leq e\nu\ell n^3 (2 + \sum_{i=1}^{\infty} n^{-n^i}) \\ &\leq e\nu\ell n^3 (2 + 2).\end{aligned}$$

Setting $\ell = n$ we get the upper bound for the expected optimisation time. □

3.4.3 Lower Bound for the $(\leq \mu + 1)$-EA

We now give a lower bound on the runtime of the $(\leq \mu + 1)$-EA for the APSP problem. For this we construct a worst-case graph. Let K_n be the complete directed graph $K_n = ([1, \dots, n], \{(u, v) \mid u, v \in [1, \dots, n], u \neq v\})$ with edge lengths defined as

$$w(u, v) = \begin{cases} 1 & \text{if } |v - u| = 1, \\ n & \text{else.} \end{cases}$$

For two distinct vertices u, v the unique shortest path from u to v is $((u, u + 1), \dots, (v - 1, v))$ if $u < v$ and $((u, u - 1), \dots, (v + 1, v))$ otherwise. These edge lengths, together with our initialisation and selection for replacement, guarantee at any time all individuals in the population consist of a single edge or are a shortest path.

Definition 3.4. *The* distance *of two paths is the minimal number of elementary mutations needed to mutate one path into the other. A mutation step* crosses *a distance of c if the path it chooses to mutate and the one it creates have distance c.*

Note that for the graph K_n with edge lengths w the distance of two shortest paths P_1, P_2 is the size $|E(P_1) \triangle E(P_2)|$ of the symmetric difference $\triangle$ of the set of edges $E(P_1), E(P_2)$ of the two paths.

Lemma 3.8. *For any $c \in \mathbb{N}$, the probability that a mutation step crosses a distance of c is at most $\frac{4c}{e(n-2)^c} \frac{n-2}{n-3} = O(cn^{-c})$.*

Proof. Let P_1 be the shortest path the mutation step chooses for mutation and let P_2 be a shortest path that has a distance of c to P_1. Each elementary mutation of a sequence of elementary mutations applied to P_1 either decreases or increases the distance of the resulting solution to P_2. Hence a shortest path P_2 in distance c from P_1 can only be obtained via a sequence of $c + 2i$ elementary mutations for some $i \in \mathbb{N}_0$. In this case, $c + i$ of them decrease and i of them increase the distance of the intermediate solution to P_2. The probability that a certain mutation of the $c + 2i$ elementary mutations decreases this distance is at most $(n - 2)^{-1}$. This is because there are at most 2 additions/deletions that achieve the distance reduction out of at least $2(n - 2)$ possible elementary mutations.

Assume in this paragraph that our mutation consists of exactly $c + 2i$ elementary mutations. Then there are at most $\binom{c+2i}{i}$ choices for the $c + i$ ones that reduce the distance to P_2. In consequence, the probability to end up with P_2 is at most $\binom{c+2i}{i}(n - 2)^{-(c+i)}$.

It is easy to see that there are at most $2c$ shortest paths P_2 that are in distance c of P_1. Thus, the probability to end up with any shortest path P_2 in distance c of P_1 is at most $2c\binom{c+2i}{i}(n-2)^{-(c+i)}$.

Recall that the probability that our mutation consists of $c+2i$ elementary mutations is $(e(c+2i-1)!)^{-1}$. Hence the probability that a single mutation step crosses a distance c is at most

$$\begin{aligned}
\sum_{i=0}^{\infty} \frac{1}{e(c+2i-1)!}\binom{c+2i}{i}\frac{2c}{(n-2)^{c+i}} &= \frac{2c}{e(n-2)^c}\sum_{i=0}^{\infty}\frac{c+2i}{i!(c+i)!}\frac{1}{(n-2)^i} \\
&\leq \frac{4c}{e(n-2)^c}\sum_{i=0}^{\infty}\frac{1}{(n-2)^i} \\
&\leq \frac{4c}{e(n-2)^c}\frac{n-2}{n-3} \\
&= O(cn^{-c}).
\end{aligned}$$

□

Lemma 3.9. *For any constant k, there exists a constant $c := c(k)$ such that with probability $O(n^{-k})$, during its optimisation time the $(\leq \mu + 1)$-EA will only accept mutation steps that cross at most a distance of c.*

Proof. We know from Theorem 3.5 that the $(\leq \mu + 1)$-EA has with high probability an expected optimisation time of $O(n^4)$. Furthermore Lemma 3.8 tells us that a distance of c is crossed with probability $O(cn^{-c})$. Thus, the probability that during the optimisation time a step crossing a distance of c is accepted is at most $O(n^{4-c})$. Choosing c appropriately, this probability turns into $O(n^{-k})$. □

Let $P^* := ((1,2),(2,3),\ldots,(n-1,n))$ be the shortest path from vertex 1 to vertex n in K_n with edge lengths w. Consider a sequence of mutation steps (each changing at least one edge) that may create P^*. Of these steps consider the last $\lfloor \frac{n-3}{c} \rfloor$ where c is the constant from Lemma 3.9. Let the paths that are created during these steps be $P_0, P_1, \ldots P_{\lfloor \frac{n-3}{c} \rfloor} = P^*$. Since $|P^*| = n-1$ and since P_j has at most c edges more than P_{j-1}, we have that $|P_0| \geq 2$ and thus all P_j are shortest paths. Thus, these paths fulfil the requirements of the following definition.

Definition 3.5 (c-Trail)**.** *A c-*trail $T := (P_0, P_1, \ldots, P_{\lfloor \frac{n-3}{c} \rfloor})$ *of P^* is a sequence of* shortest *paths such that P_0 consists of at least 2 edges, $P_{\lfloor \frac{n-3}{c} \rfloor} = P^*$, and for all $j \in \left[1, \ldots, \lfloor \frac{n-3}{c} \rfloor\right]$, P_{j-1} and P_j have a distance of at most c.*

Since there are at most $(2c)^2$ shortest paths that have a positive distance of at most c from P_j, there are at most $(4c^2)^{\lfloor \frac{n-3}{c} \rfloor}$ such c-trails.

Theorem 3.7. *With high probability, the optimisation time of the* $(\leq \mu + 1)$*-EA on* K_n *with edge lengths* w *is* $\Omega(n^4)$.

Proof. Let c be the constant from Lemma 3.9. While finding the shortest path P^* the $(\leq \mu + 1)$-EA must perform a sequence of mutations, each creating a shortest path. By Lemma 3.9 we know that each two consecutive paths will have distance at most c with high probability. Hence the $(\leq \mu + 1)$-EA must perform all $\lfloor \frac{n-3}{c} \rfloor$ mutation steps that create P_j out of P_{j-1} for $j \in \left[1, \ldots, \lfloor \frac{n-3}{c} \rfloor\right]$ for one of the c-trails of P^*. Note that we will ignore the mutation steps leading to P_0 in this proof.

The proof will now proceed as follows. First, we will analyse the number of steps the $(\leq \mu + 1)$-EA needs to follow one particular c-trail of P^*. Then, we will prove that with high probability the $(\leq \mu + 1)$-EA will not follow any of the c-trails of P^* in less than $\Omega(n^4)$ steps.

Fix one c-trail $T = (P_0, P_1, \ldots, P_{\lfloor \frac{n-3}{c} \rfloor})$ of P^*. We call a mutation step an *improvement* in T if it creates P_j out of P_{j-1} for some $1 \leq j \leq \lfloor \frac{n-3}{c} \rfloor$. If all $\lfloor \frac{n-3}{c} \rfloor$ improvements in T have been done, we say that the $(\leq \mu + 1)$-EA has followed T.

Let the random variable t' denote the number of steps the $(\leq \mu + 1)$-EA needs to follow T. For $i \in [1, \ldots, t']$ define the binary random variables X_i by $X_i = 1$ if the i-th mutation step is an improvement in T. An improvement changes at least 1 and at most c edges of a path. In order to change $c' \in [1, \ldots, c]$ edges, the algorithm first has to pick the right individual and then change the c' edges. Picking the right individual has probability $\frac{1}{n(n-1)}$. According to Lemma 3.8 changing the c' edges happens with probability $\frac{4c'}{e(n-2)^{c'}} \frac{n-2}{n-3}$. Thus, for $n \geq 6$, $i \in [1, \ldots, t]$ and for all $x_1, \ldots, x_{i-1} \in \{0, 1\}$ we have that

$$\begin{aligned}
\Pr[X_i = 1 | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}] &\leq \sum_{c'=1}^{c} \frac{1}{n(n-1)} \frac{4c'}{e(n-2)^{c'}} \frac{n-2}{n-3} \\
&\leq \frac{4}{en(n-1)(n-2)} \cdot \frac{n-2}{n-3} \cdot \sum_{c'=0}^{c-1} \frac{c'}{(n-2)^{c'}} \\
&< \frac{4c}{en(n-1)(n-2)} \cdot \frac{n-2}{n-3} \cdot \frac{n-2}{n-3} \\
&< \frac{8c}{e(n-1)^3}.
\end{aligned}$$

For $i > t'$ define X_i by $\Pr[X_i = 1] = \frac{8c}{e(n-1)^3}$ and for $i \in [1, \ldots, t]$ define the binary random variables X_i^* by $\Pr[X_i^* = 1] = \frac{4}{e(n-1)^3}$. Let $t := \frac{1}{80c^4}(n-1)^4$. The expected value of $X^* := \sum_{i=1}^{t} X_i^*$ is

$$\mathbb{E}[X^*] = \sum_{i=1}^{t} \Pr[X_i^* = 1] = t \frac{8c}{e(n-1)^3} = \frac{n-1}{10ec^3}.$$

If the $(\leq \mu + 1)$-EA has found P^* in t steps by following the c-trail T, then obviously $X := \sum_{i=1}^{t} X_i \geq |T| = \lfloor \frac{n-3}{c} \rfloor$. Hence,

$$\Pr[P^* \text{ found in } t \text{ steps by following } T] = \Pr[X \geq |T|].$$

Let $\beta := \frac{|T|}{\mathbb{E}[X^*]}$. Then for $n \geq 5 + 2c$ it holds that

$$\beta \geq \lfloor \frac{n-3}{c} \rfloor \cdot \frac{10ec^3}{n-1} \geq \frac{n-3-c}{c} \cdot \frac{2c}{n-1} \cdot 5ec^2 \geq 5ec^2.$$

Hence, by Lemma 3.1a) and the Chernoff bound Theorem 3.1c), the probability of finding P^* in $t = \frac{1}{80c^4}(n-1)^4$ steps by following c-trail T is bounded by

$$\begin{aligned}
\Pr[X \geq |T|] &\leq \Pr[X^* \geq |T|] \\
&= \Pr[X^* \geq \beta \mathbb{E}[X^*]] \\
&< (e^{\beta-1}\beta^{-\beta})^{\mathbb{E}[X^*]} \\
&\leq (\frac{e}{\beta})^{\beta \mathbb{E}[X^*]} \\
&\leq (5c^2)^{-|T|} \\
&= (5c^2)^{-\lfloor \frac{n-3}{s} \rfloor}.
\end{aligned}$$

Since the $(\leq \mu + 1)$-EA has to follow one of the c-trails of P^* in order to find P^*, the probability that the $(\leq \mu + 1)$-EA finds P^* in $t = \frac{1}{80c^4}(n-1)^4$ steps is bounded by

$$\begin{aligned}
\Pr[P^* \text{ found in } t \text{ steps}] &\leq \sum_{T \in \mathcal{T}} \Pr[P^* \text{ found in } t \text{ steps by following } T] \\
&\leq \sum_{T \in \mathcal{T}} (5c^2)^{-\lfloor \frac{n-3}{c} \rfloor} \\
&= (\tfrac{4}{5})^{\lfloor \frac{n-3}{c} \rfloor}.
\end{aligned}$$

Here $\mathcal{T}$ denotes the set of all c-trails of P^*. In the penultimate line we used the fact that there are at most $(4c^2)^{\lfloor \frac{n-3}{c} \rfloor}$ c-trails of P^*. Since the $(\leq \mu + 1)$-EA has to find P^* to solve the APSP it needs with high probability at least $\Omega(n^4)$ steps. □

Observe that this theorem implies an expected optimisation time of $\Omega(n^4)$.

3.4.4 Analysing the $(\leq \mu + 1)$-GA

With a tight bound for the $(\leq \mu + 1)$-EA now established we next will analyse what happens if we also use crossover in our algorithm. More precisely, we show that we enrich the $(\leq \mu + 1)$-EA with a crossover operator, then the expected optimisation time drops to $O(n^{3.5}\sqrt{\log n})$.

On first glance it seems natural that the additional use of powerful variation operators should speed up computation. However, this behaviour could not be proven for a non-artificial problem so far. Several reasons for this have been discussed in the literature. In our setting, the following aspect seems crucial. The hoped for strength of the crossover operator lies in the fact that it can advance a solution significantly. E.g., it can combine two shortest paths consisting of ℓ_1 and ℓ_2 edges to one consisting of $\ell_1 + \ell_2$ edges in one operation. On the negative side this will only work if the two individuals we try to combine fit together. Thus with relatively high probability, the crossover operator will produce an invalid solution. In our setting this means that a crossover will not produce a walk at all. Often, this disadvantage seems to outnumber the chance of faster progress.

Our analysis shows that this does not happen in our setting. In fact, from the point on when our population contains all shortest paths having $O(\sqrt{n \log n})$ edges, crossover becomes so powerful that we would not even need mutation anymore.

We can prove the claimed upper bound for all three crossover operators introduced in Subsection 3.4.1. However, as the crossover operators we use become more elaborate, we need to add the following restrictions for the proof.

R1: Among two shortest paths the fitness function prefers the path consisting of fewer edges. (Needed for $\otimes_2$.)

R2: The input graph has unique shortest paths. (Needed for $\otimes_3$.)

With these restrictions we can show for each crossover operator that it successfully creates a longer path by combining two shorter paths with a certain probability. Using these success probabilities we then prove the expected optimisation time of $O(n^{3.5}\sqrt{\log n})$.

Lemma 3.10. *Let $k > 1$. Assume the population $\mathcal{I}$ contains a shortest path for any pair of vertices $(u', v') \in V_k^2$. Let $\ell \in [k+1, \ldots, 2k]$ and $(u, v) \in V_\ell^2$. Then the following holds.*

a) A single execution of the $\otimes_1$-operator generates a shortest path from u to v with probability $\Omega(\frac{2k+1-\ell}{n^4})$.

b) Assume that for all $(u', v') \in V_k^2$, $\mathcal{I}$ contains a shortest path from u' to v' consisting of at most k edges. A single execution of the $\otimes_2$-operator generates a shortest path from u to v having at most ℓ edges with probability $\Omega(\frac{(2k+1-\ell)^2}{kn^4})$.

c) Assume R2. A single execution of the $\otimes_3$-operator generates the shortest path from u to v with probability $\Omega(\frac{(2k+1-\ell)^3}{k^2n^4})$.

Proof. Claim a) The $\otimes_1$-operator can generate a shortest path from u to v by picking a path P_u starting in u and a path P_v ending in v, such that P_u together with P_v forms a path from u to v. A particular pair (P_u, P_v) is chosen with probability at least

$$(n(n-1))^{-2} = \Omega(\tfrac{1}{n^4}).$$

This leaves the task of counting the number of pairs that generate a shortest path from u to v. Let $P = ((u, w_1), (w_1, w_2), \ldots, (w_{\ell-1}, v))$ be a shortest path from u to v having ℓ edges. Then, for every vertex $w_i, i \in [\ell - k, \ldots, k]$, a shortest path from u to w_i and a shortest path from w_i to v are in the population. Hence, there are at least $2k + 1 - \ell$ pairs of paths that the $\otimes_1$-operator can combine to a shortest path from u to v. In summary, the probability that a single crossover step generates a shortest path from u to v is at least $\Omega(\frac{2k+1-\ell}{n^4})$.
Claim b) To generate a shortest path from u to v, it suffices that the $\otimes_2$-operator picks a path P_u starting in u, a path P_v ending in v, and a number $i \in [0, \ldots, |P_u|]$ such that the first i edges of P_u together with P_v form a path from u to v. The probability that a particular triple (P_u, P_v, i) with $|P_u| \leq k, |P_v| \leq k, i \leq |P_u|$ is chosen is at least

$$(n(n-1))^{-2}(k+1)^{-1} = \Omega(\tfrac{1}{kn^4}).$$

It remains to count how many such triples generate a shortest path from u to v. Let $P = ((u, w_1), \ldots, (w_{\ell-1}, v))$ be such a shortest path having ℓ edges. Let $\ell - k \leq j \leq k$. Then $\mathcal{I}$ contains a shortest path $P_u = ((u, w'_1), \ldots, (w'_{j-1}, w_j))$ from u to w_j having j edges. Since $\ell - i \leq k$ we also have that for each $i \in [\ell - k, \ldots, j]$, $\mathcal{I}$ contains a shortest path P_v from w'_i to v. Obviously, the first i edges of P_u combined with P_v form a shortest path from u to v. Hence, the total number of triples yielding a shortest path from u to v having ℓ edges is at least

$$\sum_{j=\ell-k}^{k} (j - (\ell - k) + 1) = \Omega((2k + 1 - \ell)^2).$$

Thus, the probability that $\otimes_2$ generates such a path in a single step is at least $\Omega(\frac{(2k+1-\ell)^2}{kn^4})$.
Claim c) To generate P, the $\otimes_3$-operator has to pick a path P_u starting in u, a path

P_v ending in v, and numbers $i \in [0, \ldots, |P_u|]$, $j \in [0, \ldots, |P_v|]$ such that the first i edges of P_u together with the last j edges of P_v form the path P. The probability that a particular 4-tuple (P_u, P_v, i, j) with $|P_u| \leq k, |P_v| \leq k, i \leq |P_u|, j \leq |P_v|$ is chosen is at least

$$(n(n-1))^{-2}(k+1)^{-2} = \Omega(\tfrac{1}{k^2 n^4}).$$

It remains to count the number of such 4-tuples that generate P. For this, consider two sub-paths of P, one starting at u, the other ending at v. Observe that those sub-paths are also shortest paths. Since we assume all shortest paths to be unique, both sub-paths will be in the population if they consist of at most k edges. If the sum of the numbers of edges of both paths is some $i \in [\ell, \ldots, 2k]$, they have $i-\ell$ edges in common and the number of successful crossover positions is $i-\ell+1$. The number of pairs of sub-paths that have $i-\ell$ edges in common is $2k+1-i$. Hence, the total number of 4-tuples yielding P is at least

$$\begin{aligned}\sum_{i=\ell}^{2k}(i-\ell+1)\cdot(2k+1-i) &= \sum_{i=0}^{2k-\ell}(i+1)\cdot(2k+1-i-\ell)\\ &= \Omega((2k+1-\ell)^3).\end{aligned}$$

Thus, the probability that $\otimes_3$ generates the shortest path P in a single step is at least $\Omega(\frac{(2k+1-\ell)^3}{k^2 n^4})$. □

Corollary 3.1. *Let $k > 1$ and $\ell = \frac{3k}{2}$. Assume the population $\mathcal{I}$ contains for any pair of vertices $(u', v') \in V_k^2$ a shortest path. Assuming R1 for $\otimes_2$ and R2 for $\otimes_3$ the following holds.*

a) Let $(u, v) \in V_\ell^2$. A single execution of the $\otimes_i$-operator for $i \in \{1, 2, 3\}$ will create a shortest path from u to v with probability at least $\Omega(\frac{k}{n^4})$.

b) The expected number of crossover steps until $\mathcal{I}$ contains a shortest path from u to v for all $(u, v) \in V_\ell^2$ is $O(\frac{n^4 \log(n)}{k})$.

Proof. Claim a) This follows directly by plugging ℓ into Lemma 3.10.
Claim b) This proof is similar to the proof of the coupon collector's theorem (cf. [MR95]). Let $r = |V_\ell^2| - |V_k^2| = O(n^2)$ be the number of paths that have to be found. By Claim a) the first of the sought after paths will be found after an expected number of $O(\frac{n^4}{k}\frac{1}{r})$ steps. If i paths have been found, it will take an expected number of $O(\frac{n^4}{k}\frac{1}{r-i})$ steps until the $(i+1)$-st path is found. Hence, finding all r paths takes

$$\sum_{i=0}^{r-1} O\left(\frac{n^4}{k}\right)\frac{1}{r-i} = O\left(\frac{n^4}{k}\right)\sum_{i=1}^{r}\frac{1}{i} = O\left(\frac{n^4\log(n)}{k}\right)$$

steps. □

Theorem 3.8. *Let $i \in \{1,2,3\}$. Assume that the conditions for the $\otimes_i$-operator hold. Then the $(\leq \mu + 1)$-GA using mutation and the $\otimes_i$-crossover operator with any constant rate needs an expected number of $O(n^{3.5}\sqrt{\log(n)})$ steps to solve the APSP problem.*

Proof. Let $k := \sqrt{n\log(n)}$. Both the $\otimes_i$ and the mutation operator happen with constant probability and neither can decrease the fitness of the population. Thus, for an upper bound we may consider the steps of one of the operators only. First assume that only the mutation operator is used at the beginning of the algorithm. According to Theorem 3.6 the algorithm will need an expected number of at most $O(n^{3.5}\sqrt{\log(n)})$ steps to find a shortest path from u to v for every $(u,v) \in V_k^2$. Note that Theorem 3.6 also holds if a fitness function preferring fewer edges is used. As soon as this happens, we only consider crossover until the remaining shortest paths have been found. For this we simply apply Corollary 3.1 repeatedly for the $\otimes_i$-operator until $\ell = n-1$. Hence the expected number of steps is

$$\begin{aligned}
\sum_{j=\lfloor \log_c(k)\rfloor}^{\lceil \log_c(n)\rceil} O\left(\frac{n^4\log(n)}{c^j}\right) &= O\left(n^4\log(n)\sum_{j=\lfloor \log_c(k)\rfloor}^{\lceil \log_c(n)\rceil}\frac{1}{c^j}\right)\\
&= O\left(\frac{n^4\log(n)}{c^{\log_c(k)}}\sum_{j=0}^{\lceil \log_c(\frac{n}{k})\rceil}\frac{1}{c^j}\right)\\
&= O\left(n^{3.5}\sqrt{\log(n)}\right)
\end{aligned}$$

where $c := \frac{3}{2}$. □

The Restrictions R1 and R2

We now demonstrate where our proof of the optimisation time would fail without the additional constraints for $\otimes_2$ and $\otimes_3$.

First we show why we need assumption R1 in our proof for the crossover operator $\otimes_1$. For this consider for even n the complete graph $K_n = ([1,\ldots,n], \{(u,v) \mid u,v \in [1,\ldots,n], u \neq v\})$ with edge lengths

$$w'(u,v) := \begin{cases} 1 & \text{if } |v-u| = 1 \text{ and } u,v \leq \frac{n}{2}+1,\\ \frac{2}{n} & \text{if } |v-u| = 1 \text{ and } u,v \geq \frac{n}{2}+2\\ \frac{2}{n} & \text{if } (u,v) \in \{(2,\frac{n}{2}+2),(\frac{n}{2}+2,2),(n,1),(1,n)\}\\ 1+w_{uv}^2 & \text{else} \end{cases}$$

depicted in Figure 3.6. Here, w_{uv} is the cost of the shortest path using the edges of length 1 and $\frac{2}{n}$ from u to v.

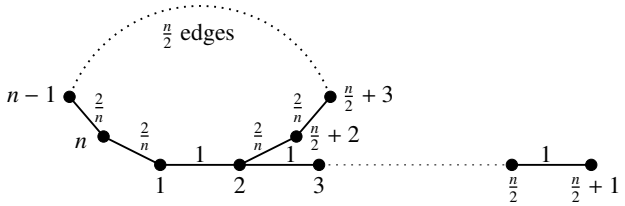

Figure 3.6: The complete graph K_n with edge lengths w'. For this graph the analysis of the $\otimes_2$-operator fails if the fitness function does not prefer individuals with fewer edges. The shown edge lengths apply to both directions of the indicated edge while the edges not shown in the figure are longer than the shortest paths shown.

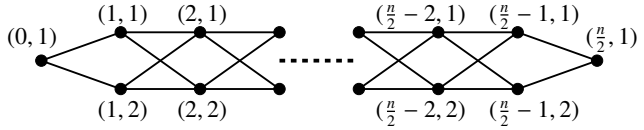

Figure 3.7: The complete graph K_n'' with edge lengths w'' for which the analysis of the $\otimes_3$-operator fails since the shortest paths are not unique. The edges shown in the figure have length 1 in both directions and the ones not depicted are longer than the shortest paths shown.

Assume, as in Lemma 3.10, that for all vertex pairs $(u, v) \in V_k^2$ a shortest path is in the population $\mathcal{I}$, and that $\ell \in [k+1, \ldots, 2k]$ and $\ell \leq \frac{n}{2}$. Now consider the computation of a shortest path from $u := 1$ to $v := \ell + 1$ using the $\otimes_2$-operator. Two such shortest paths exist, namely P_1 which uses the edge $(1, 2)$ of cost 1 and has ℓ edges and P_2 which uses the $\frac{n}{2}$ edges of cost $\frac{2}{n}$ and has $\ell - 1 + \frac{n}{2}$ edges. If $\mathcal{I}$ contains for the paths from u to i for $i \in [2, \ldots, k+1]$ the paths using the edge $(1, 2)$, the proof of Lemma 3.10b) works. However, what happens if $\mathcal{I}$ contains the paths using the $\frac{n}{2}$ edges of cost $\frac{2}{n}$? Observe that in this case there are $\Omega(n)$ possible positions to cut P_u. Hence the probability that the $\otimes_2$-operator picks a convenient triple (P_u, P_v, i) drops from $\Omega(\frac{1}{kn^4})$ to $\Omega(\frac{1}{n^5})$.

Now consider the $\otimes_3$-operator. In the proof of Lemma 3.10 we assumed that assumption R2 holds, which requires that there exists one unique shortest path for each pair of vertices. To see why this requirement is essential, we again construct a problematic graph. Consider for even n the complete graph $K_n'' := (V, \{(u, v) | u, v \in V, u \neq v\})$ with the vertex set defined as

$$V := \left[1, \ldots, \frac{n}{2} - 1\right] \times \{1, 2\} \cup \left\{(0, 1), \left(\frac{n}{2}, 1\right)\right\}.$$

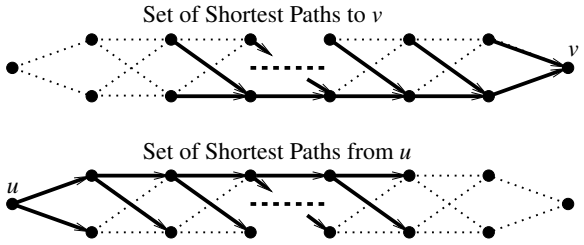

Figure 3.8: An example for sets of shortest paths in K_n'' that do not overlap enough and thus do not fulfil the requirements for the proof of Lemma 3.10c).

and edge lengths

$$w''(u = (u_1, u_2), v = (v_1, v_2)) := \begin{cases} 1 & \text{if } |v_1 - u_1| = 1, \\ 1 + w_{uv}^2 & \text{else.} \end{cases}$$

Again, w_{uv} is the length of the shortest path using the edges of length 1 from u to v. See Figure 3.7 for an example.

Observe that there are many different shortest paths connecting two vertices. Since all shortest paths connecting two vertices have an equal number of edges the graph fulfils assumption R1.

Now assume that $\mathcal{I}$ contains the shortest paths from $u := (0, 1)$ to i for $i \in [1, \ldots, k] \times \{0, 1\}$ and from j to $v := (\frac{n}{2}, 1)$ for $j \in \left[\frac{n}{2} - k, \ldots, \frac{n}{2}\right] \times \{0, 1\}$ as given in Figure 3.8. Then for any shortest path from u to v the population will not contain all sub-paths of length up to k, as needed by the proof of Lemma 3.10. Even more, any pair of paths, one starting in u, the other ending in v, will only overlap on at most two vertices.

3.4.5 Experimental Results

In the previous sections we saw that the asymptotic worst case optimisation time of the $(\leq \mu + 1)$-EA is $\Theta(n^4)$, while that of the $(\leq \mu + 1)$-GA is $O(n^{3.5}\sqrt{\log n})$. In this section we will show that this difference is in fact noticeable in practice. For this we implemented Algorithm 3.1 given in Section 3.2.1 with the three different crossover operators and ran it on the following three graph classes.

The first class consists of the weighted complete graphs K_n with edge lengths w from Section 3.4.3. This graphs have edge weights 1 for all edges (u, v) with $|v - u| = 1$, and weight n for all other edges.

The second and third graph classes are the ones used in Section 3.4.4 to argue why we need additional restrictions in the proofs if the operators $\otimes_2$ or $\otimes_3$ are used. Note however, that we only used this restrictions in the proof. Our algorithm does not prefer paths with fewer edges. Nor does it need unique shortest paths.

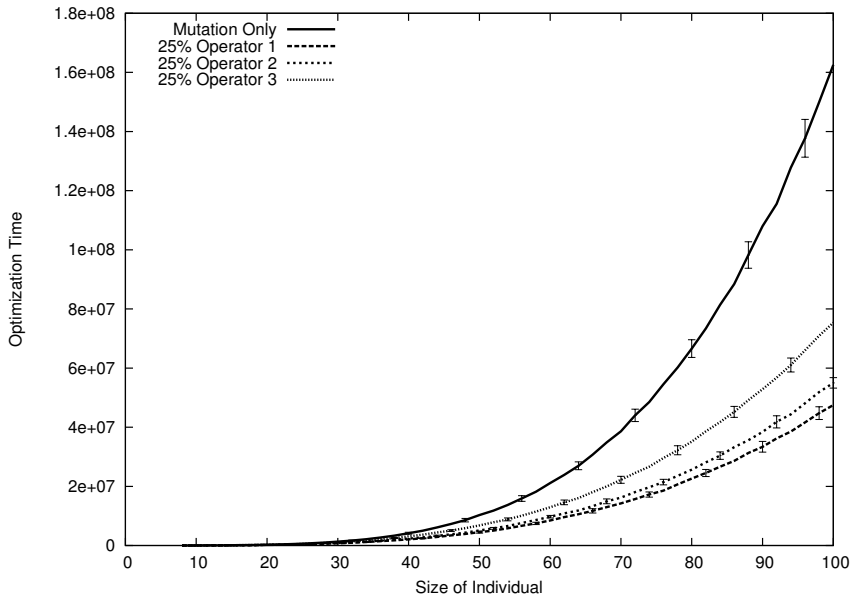

Figure 3.9: Optimisation time for the various crossover operators on K_n with edge lengths w (see Section 3.4.3).

We ran four variations of Algorithm 3.1 on all three graph classes mentioned above. The first variation only uses mutation while the remaining three variations used the three different crossover operators $\otimes_1$, $\otimes_2$ and $\otimes_3$. The crossover probability was set to $p_\otimes := \frac{1}{4}$.

For all three graph classes we considered all graphs having an even number of vertices between 8 and 100. On each instance the algorithm was run 50 times. The average optimisation times for the experiments are shown in Figure 3.9, Figure 3.10, and Figure 3.11. To keep the plots legible we only plotted the standard deviation for every fourth data point. For all instances of 40 or more edges, the standard deviation in the data is below 10%.

It can clearly be seen that adding any of the crossover operators indeed speeds up the computation considerably. The results also show that the "bad graphs" K_n with edge lengths w' and K_n'' with edge lengths w'' from Section 3.4.4 are not hard to solve for the corresponding crossover operators. In comparison to the other graph classes, the mutation operator is more effective on K_n'' with w''. The reason is probably that, due to the structure of w'', the mutation operator has a lot of possibilities to create shortest paths. Thus, the difference between runs with and without crossover are not quite as noticeable.

To estimate the different exponents of the optimisation times with and without crossover, we additionally ran the algorithms another 20 times each on larger

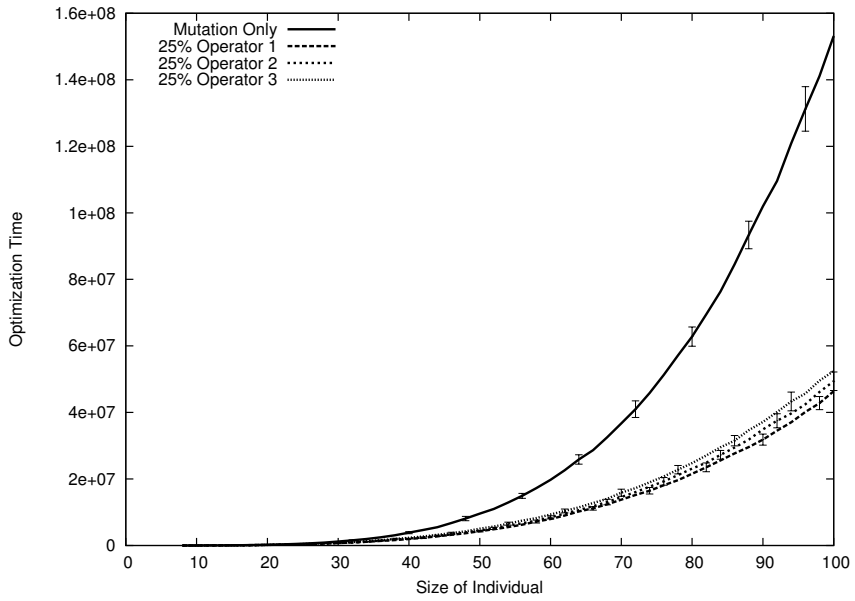

Figure 3.10: Optimisation time for the various crossover operators on K'_n with edge lengths w' (see Figure 3.6).

instances of size $50, 60, 70, \ldots, 250$. We chose these bigger input sizes to weaken the effect of the lower order terms on the optimisation time. To visualise the different exponents, we use log-log plots. This means that both the x-axis and the y-axis are scaled logarithmically. The reason for this is that for any polynomial $f(x) = ax^n + o(x^n)$, a log-log plot will plot the function

$$\log\left(f\left(\log^{-1}(x)\right)\right) = \log\left(a\left(e^x\right)^n + o\left(\left(e^x\right)^n\right)\right) = nx + o(x),$$

thus exposing the highest exponent of the polynomial.

Figure 3.12, Figure 3.13, and Figure 3.14 show the log-log plots. The difference in the exponent of the optimisation time between the mutation-only algorithm and any of the algorithms using crossover can easily be discerned in the plots. We also calculated the slope of the plots to approximate the order of growth of the optimisation time. The results are shown in table 3.1. Only using mutation may indeed cause a quartic optimisation time as the numbers for the graph K_n with weights w and w' show. Also, on all three examples crossover seems to be slightly faster than the $O(n^{3.5}\sqrt{\log n})$ upper bound shown by us.

The experiments also show that $\otimes_1$ seems to have a slight edge over $\otimes_2$ which in turn is slightly faster than $\otimes_3$. We conjecture that this is caused by the fact that the simpler crossover operators on average combine longer paths than the more complicated ones.

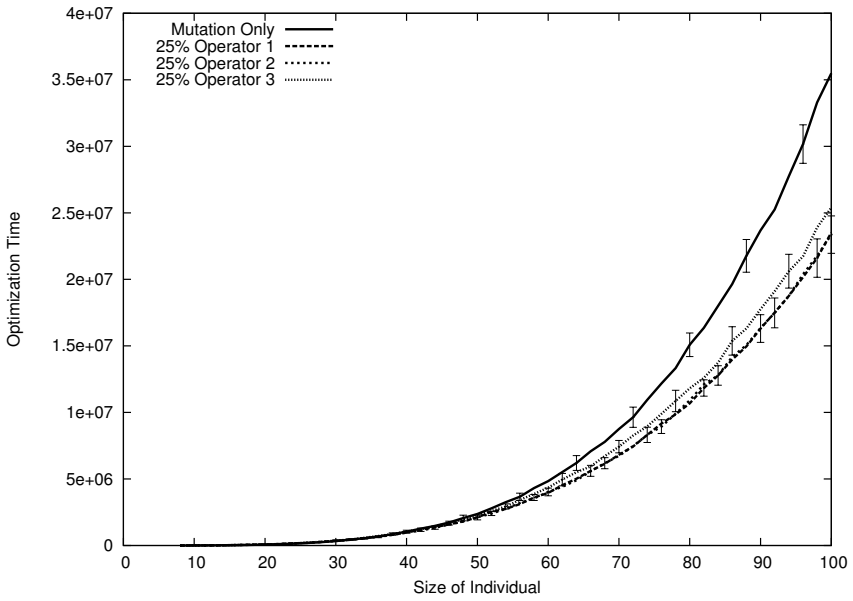

Figure 3.11: Optimisation time for the various crossover operators on K_n'' with edge lengths w'' (see Figure 3.7).

3.5 Conclusion and Discussion

In this chapter we studied the behaviour of evolutionary algorithms for the single source shortest path problem and the all-pairs shortest path problem.

We gave a tight analysis of the optimisation time of an evolutionary algorithm for the SSSP problem that was given in [STW04]. This includes improving the upper bound of $O(n^2\ell\log(\frac{n}{\ell}))$ that is implicit in a proof in that paper to $O(n^2\max\{\ell,\log(n)\})$. Furthermore we gave a carefully selected lower bound graph for all values of n and ℓ.

At least as important as the precise bounds for this particular problem are the methods we developed for this analysis. Past arguments suggested a coupon-collector like behaviour in finding the shortest paths. Those, however, cannot be employed to obtain such sharp bounds. Indeed, our analysis shows that the true behaviour is different. Namely, the different shortest paths grow at comparable speeds that are strongly concentrated around their expected values.

Armed with this methodology, we also gave an analysis of an evolutionary algorithm for the all-pairs shortest path problem. Here we were able to show a tight bound of $\Omega(n^4)$ on the optimisation time if only mutation is used. We defined a class of worst-case graphs for which the algorithm indeed has a quartic optimisation time.

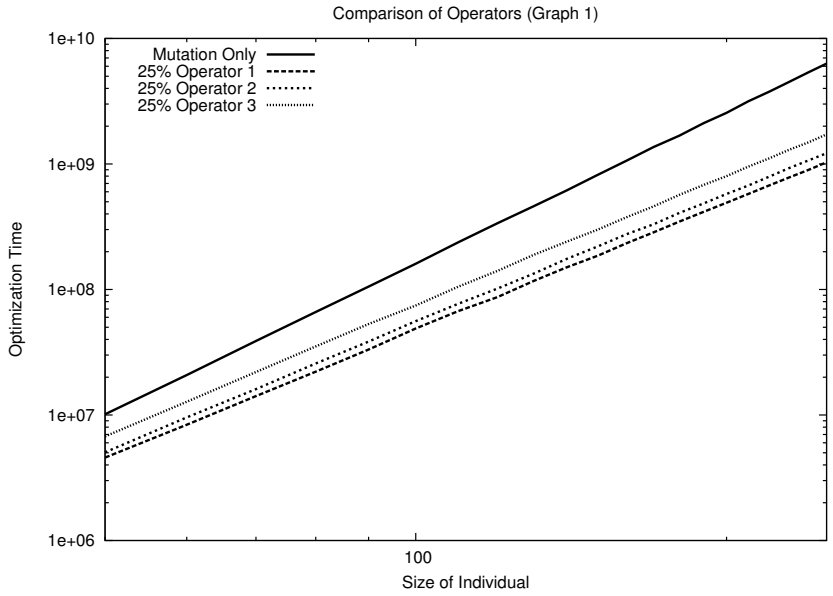

Figure 3.12: Log-log plots for K_n with edge lengths w .

	w	w'	w''
Mutation Only	4.00	4.01	3.90
Crossover ($\otimes_1$)	3.37	3.38	3.43
Crossover ($\otimes_2$)	3.41	3.42	3.41
Crossover ($\otimes_3$)	3.44	3.36	3.41

Table 3.1: The slope of the log-log plots in Figure 3.12, Figure 3.13 and Figure 3.14.

The most interesting result however is that a natural evolutionary algorithm using only mutation is provably outperformed by one using mutation and crossover. Indeed, we showed that adding crossover to the evolutionary algorithm for the APSP problem improves its optimisation time to $O(n^{3.5}\sqrt{\log n})$. Clearly, this cannot compete with classical algorithms that are custom tailored for the all-pairs shortest path problem. Still, although the difference is only by a factor of nearly $\sqrt{n}$, this is the first time that crossover was proven to be useful in the context of a non-artificial problem. Hence this result gives a better theoretical foundation for the use of crossover in practical applications than previous results on artificially defined pseudo-boolean functions.

While our work is very satisfying from the methodological point of view, some

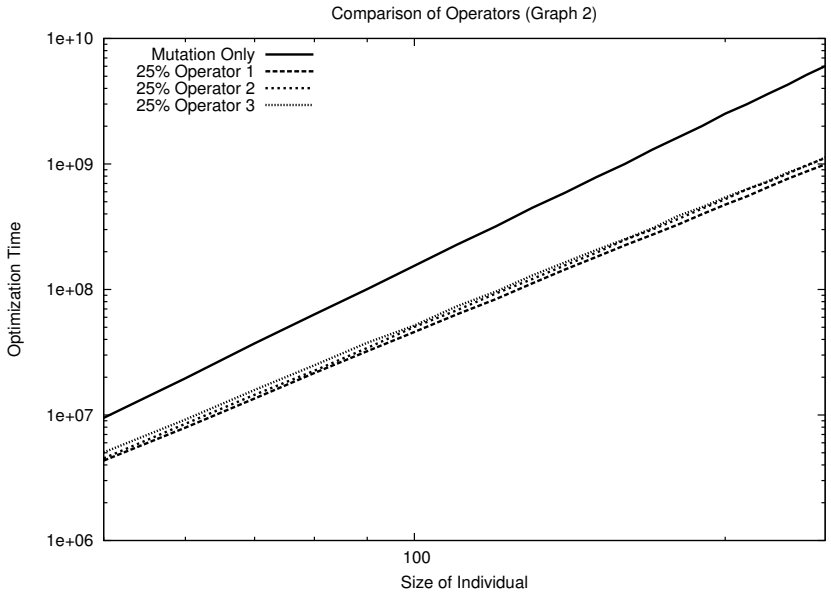

Figure 3.13: Log-log plots for K'_n with edge lengths w'.

interesting aspects remain open.

For the SSSP problem the most challenging question from a broader perspective is whether the multi-criteria fitness function is necessary. Recall that we accept a newly created individual only if for no vertex the distance to the source is increased. A natural (single-criteria) alternative would be to consider the average distance. In [STW04] it is argued that the multi-criteria fitness function is necessary for the algorithm to run properly. However, the counterexample given there only works if vertices not connected to the source are assumed to have an infinite distance to the source. In this case, changing the number of ∞–distance vertices does not change the average distance, and hence the EA finds itself on a large plateau of constant fitness. A simple way to overcome this (and the one you would choose naturally in an implementation) would be to replace the infinite distance of such vertices by a large, but finite number.

This problem resembles the one of maximising linear functions $f : \{0,1\}^n \to \mathbb{R}, x \mapsto \sum_{i=1}^n a_i x_i$ with positive coefficients a_i. Viewing f as the multi-criteria fitness function $x \mapsto (a_1 x_1, \ldots, a_n x_n)$, a simple coupon collector argument shows an optimisation time of $\Theta(n \log(n))$. That the same bound also holds for the fitness function f itself is the result of a highly complex analysis by [DJW02]. Attempts to simplify this result later led to the invention of the drift analysis method in evolutionary computation (cf. [HY04]). With this development in mind, it seems likely that it is very difficult to prove that a single-criteria EA can solve the SSSP

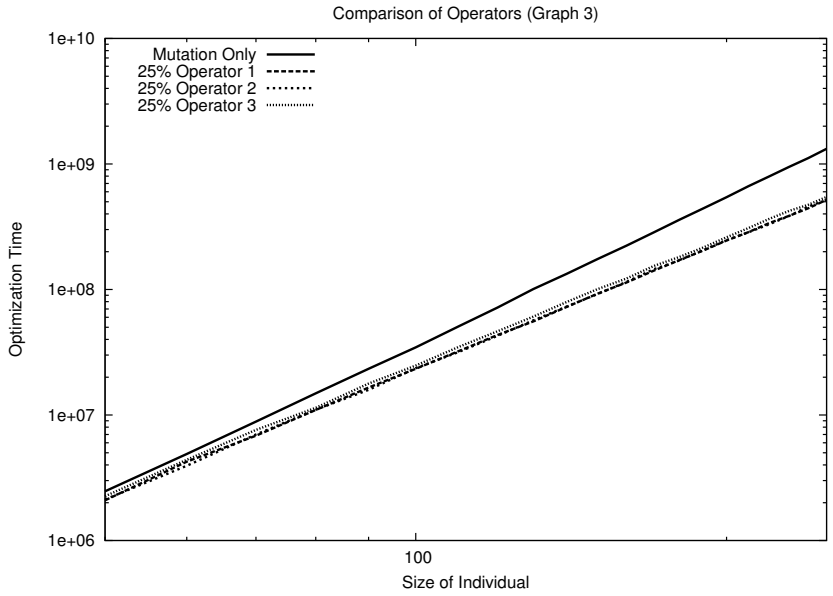

Figure 3.14: Log-log plots for K_n'' with edge lengths w''.

efficiently.

Still, some progress on this problem has been made in [BBD+09]. There it is shown that the single-criteria formulation of the SSSP problem can be solved in polynomial time. However, the run-time bounds obtained are not tight. Hence a full understanding of this problem is still missing.

For the APSP problem it is interesting to note that we did not provide any lower bound for the optimisation time of the algorithm with crossover. Indeed, in [DT09], an improved upper bound together with a matching lower bound are given which show that the optimisation time is $\Theta(n^{3.25}\sqrt[4]{\log n})$.

One more interesting aspect still remains open. We still miss a broader classification of which (non-artificial) problems can profit from using crossover.

Chapter 4

Hole Detection

In this chapter we study how much geometric information hides in the communication graph of a wireless sensor network. More precisely, we give an algorithm that can find holes in this graph by identifying the nodes on the boundary of a hole. In Section 4.1 we introduce the problem and give an overview of our contribution.

Our algorithm is based on the intuition from a continuous scenario. We discuss this idea in Section 4.2 and derive the concrete algorithm for (discrete) wireless sensor networks.

After stating the algorithm we will then analyse its runtime in Section 4.3. Another interesting aspect discussed there is the distributed and localised implementation.

The main part of this chapter consists of the correctness proof of our algorithm in Section 4.4.

We implemented the algorithm and will discuss some experimental results obtained by our implementation in Section 4.5.

This chapter is based on joint work with Stefan Funke [FK06].

4.1 Introduction

Imagine the following scenario: during a long summer drought, forest fires have started in a large region of a remote nature preserve that is hardly accessible by ground transportation. To be able to continuously assess the situation and plan appropriate countermeasures, planes are sent out to deploy thousands of wireless sensor nodes. Due to cost restrictions and to achieve the maximum life-time by energy savings, these sensor nodes are rather low-capability devices. They are only equipped with temperature and humidity sensors, a simple processing unit and a small radio device that allows for communication between nearby sensor nodes. One of the first goals is now to have this network organise itself such

that messages are routed within the network, regions of interest (e.g. the current fire-front) can be identified, and gathered data can be efficiently queried.

Achieving this goal becomes quite challenging since the only information a node has about the global network topology are its immediate neighbours with whom it can communicate. Lacking an energy-hungry GPS unit and being deployed from a plane in a rather uncontrolled fashion, none of the sensor nodes is aware of its geographic location.

Assume the area of interest is some region $\mathcal{R}$. The planes have deployed sufficiently many sensors such that the area of interest is completely monitored by the sensors. Unfortunately, not all sensors will be operational upon reaching the ground. Some of them might fall right into the flames and be destroyed, others might plunge into a lake or pond and be unable to perform their monitoring task. Paradoxically, we are particularly interested in those areas where there's an ongoing fire (and maybe also where there is a lake or pond). However, sensor nodes that fell into these areas are unable to report this fact.

We want to detect the (boundaries of) such holes in the monitored space created by fire or other phenomena via examination of the *communication graph* of the wireless nodes.

In essence, the problem that we consider is that of identifying holes just by examining a communication graph. If in a sufficiently large region sensors break down, this hole will also manifest itself in the communication graph. Hence holes identified using the communication graph are indicative of some large-scale special event in the region to be monitored. By 'identifying holes' we mean (a) that for every point on the boundary of a hole we want the algorithm to mark a sensor node nearby and (b) every sensor node marked by the algorithm lies near a boundary of a hole.

Related Work

In Fang et al. [FGG06] the authors present an algorithm for detecting holes for the case where the individual nodes know about their geographic location. Fekete et al. in [FKP+04] describe a method to identify boundaries in a wireless network which does *not* require the nodes to be aware of their geographic position. However, their method assumes a *uniform* distribution of the network nodes in all non-hole areas. In a more recent paper [KFPF06] the same authors present a deterministic approach for boundary recognition which does not rely on a uniform node distribution. Their paper also proposes interesting methods to aggregate the information gathered by the boundary recognition step in a higher-level topology sketch of the network. But their boundary detection algorithm does not come with a theoretical correctness guarantee. It also appears to require a rather high node density in practice, too. In a recent contribution by Funke [Fun05] a heuristic

boundary detection algorithm based on a similar intuition as the algorithm presented here is developed. Unfortunately, correctness cannot be proven for this heuristic algorithm (see Section 4.2.1 for a brief explanation of the problems with this approach). Furthermore its computation is not localised as it requires the computation of distance fields over the whole network.

Newer work on this topic was recently published by Funke and Milosavljevic [FM07] and Wang et al. [WGM06]. Those papers also work in a general setting. They also allow to compute virtual coordinates for the sensor nodes.

Our Contribution

We present an algorithm for detecting hole boundaries in a wireless network that is represented purely by its communication graph. If the structure of the communication graph is a unit-disk graph determined by the geographic locations of the nodes, we can prove the correctness of our algorithm. More precisely, under some additional conditions our algorithm correctly identifies nodes near boundaries and never misclassifies any nodes. The employed proof technique and sampling condition has some similarity to the one used in the area of shape reconstruction [ABE98]. The algorithm is very simple and has running time linear in its input. While the theoretical analysis is not very satisfying in that it makes rather strong assumptions about the input setting, our experimental results show that the algorithm performs much better in practice. Note that this is the case for many shape reconstruction algorithms. The idea of using the geometry information hidden in the connectivity structure of the communication graph to identify topological features has only recently been addressed [KFPF06, FKP+04, Fun05]. The algorithm presented here is to our knowledge the first that comes with some formal guarantee.

As a general topic of interest we propose the problem of recovering geometric information from purely combinatorial connectivity information as it arises for example as communication graphs in the context of wireless networks. While such communication graphs do not explicitly contain geographic location information, they were created based on certain geometric properties and hence implicitly bear some geometry information. Recovering or 'reverse engineering' this information appears to be an interesting challenge. We will show that the implicit geometry information suffices to identify certain topological features of the network. The question is how much more can be accomplished using geometry information hidden in the connectivity of the network.

4.2 The Hole Detection Algorithm

4.2.1 The Continuous Case

Picture the following continuous variant of our problem. Given a (possibly non-simply) connected region $\mathcal{R} \subset \mathbb{R}^2$ and some point $p \in \mathcal{R}$ – we call that point *seed* –, we consider the isolevels of the *geodesic distance* function d_p from p in the domain $\mathcal{R}$ as in Figure 4.1. That is, for any point $x \in \mathcal{R}$, $d_p(x)$ denotes the minimum Euclidean length of an open curve $\Gamma \subset \mathcal{R}$ with one endpoint being p, the other x. Or, in other words, $d_p(x)$ is the length of the shortest path from p to x which stays within $\mathcal{R}$ and avoids all holes.

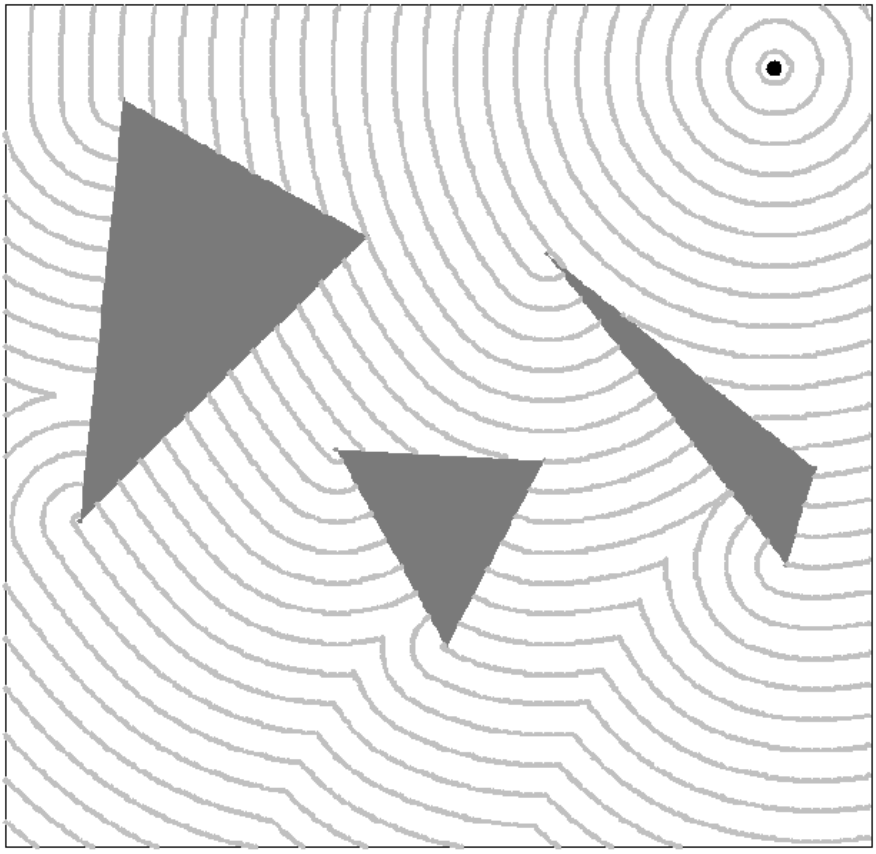

Figure 4.1: The induced isolines for one seed and three triangular holes in the continuous case.

The *isolevel*, *isoline*, or *contour* of level k of the distance function d_p is the set of points $I(k) = \{x \in \mathcal{R} : d_p(x) = k\}$. In Figure 4.1, we have depicted the

```
HoleDetection(G = (V, E))
1  Compute a maximal independent set I of G.
2  M := ∅.
3  for each s ∈ I do
4        M := M∪ExamineSeedNeighbourhood(G, s).
5  return M
```

Algorithm 4.2: Pseudo-Code for the main hole-detection routine for UDGs.

contours of level $10, 30, 50, \ldots$. If the region $\mathcal{R}$ is free of holes and all points on the boundary of $\mathcal{R}$ can be seen from p (without obstruction by a hole), the contour of level k is a subset of the circle centred at p with radius k. In the more general case with polygonal obstacles, though, the contour of level k is a collection of (possibly disconnected) circular arcs.

What is interesting for our purposes is the observation that for almost every point x of a hole boundary or the outer boundary, some contour is *broken* at that point. We say a contour is *broken* at a point p or p is called an *endpoint* of that contour, if the contour does not intersect an arbitrarily small ball around p in a topological 1-disk. The only way that a boundary point x with $d_p(x) = k$ might not be the endpoint of a component of the contour of level k is in case that the tangents of the contour of level k and of the boundary agree at x. In Figure 4.1, for example this is the case in the lower right and upper left corner of the region $\mathcal{R}$, where the 'wave-fronts' hit the outer boundary tangentially. The same happens at the upper left tip of the rightmost triangular hole in the picture. The 'reverse observation' that every breakpoint of a contour is also on a hole or outer boundary is unfortunately not true, which was incorrectly claimed in [Fun05]. When several holes are arranged in a particular order, broken isolevels that are not near any boundary might appear. For example, consider three wave-fronts that meet each other in a single point and induce a contour which consists of a single point. However, if one assumes that all holes are circular and only considers isolevels near the seed that do not interact with more than one hole, then every breakpoint of a contour coincides with a hole or outer boundary point and vice versa.

4.2.2 The Discrete Case

The key idea of our hole detection routine is to make use of these 'broken isolines' to determine nodes that are close to the outer boundary or to a boundary of

some hole. In the following we describe a rather straightforward translation of the intuition from the continuous setting to the discrete setting.

In the discrete case we consider the *communication graph* of a wireless network. It has a node for each wireless station and an unweighted edge between two nodes if the respective stations can communicate with each other. For simplicity, let us assume that two nodes can communicate with each other if they are within distance of at most one (communication radius). The communication graph arising from this assumption is a so-called *unit disk graph* (UDG).

A priori, the region of interest is the whole two-dimensional euclidean plane. We subtract from it the set O of disjoint holes. This leaves us with the region without holes, i.e. in which sensors have survived. Denote this region by

$$\mathcal{R} := \mathbb{R}^2 \setminus \bigcup_{o \in O} o.$$

In the absence of geographic location information, the only distance measure available for the algorithm is the hop distance in the unit-disk graph. For a graph $G = (V, E)$ we denote the *hop distance* or *graph distance* between two vertices $u, v \in V$ by $\mathrm{d}(u, v)$. If we consider the distances in a subgraph $U \subseteq V$ we will write $\mathrm{d}^U(u, v)$. In contrast, the *Euclidean distance* between two points $p, q \in \mathbb{R}^2$ will be denoted by $|pq|$.

With this we can then define the notion of isolevels for the discrete case.

Definition 4.1. *Let S be the set of wireless nodes in the euclidean plane and let $s \in S$. We then recursively define the isolevels of s as*

$$\begin{aligned} \mathcal{L}^0(s) &:= \{p \in \mathcal{R} : |ps| \leq 1\} \\ \mathcal{L}^{i+1}(s) &:= \left\{ p \in \left(\mathcal{R} \setminus \bigcup_{j \leq i} \mathcal{L}^j(s) \right) : \exists s' \in S \cap \mathcal{L}^i(s) : |s'p| \leq 1 \right\}. \end{aligned}$$

In other words, the $(i + 1)$th isolevel consists of all points in the plane that are in the communications range of some node in the ith isolevel but do not belong to a smaller isolevel.

Formally, we have that for every point $p \in \mathcal{R}$ there would be at least one sensor s within Euclidean distance $|ps| \leq r_{\text{sense}}$. Here r_{sense} is the *sensing radius* of the sensor nodes. In the example from the introduction that would be the radius within which they can monitor or estimate temperature or humidity.

Typically, the communication radius is considerably larger than the sensing radius r_{sense}. Let $\kappa = \frac{1}{r_{\text{sense}}}$ be the ratio between these two quantities. Clearly, the larger the value κ becomes, the denser the communication graph gets. We formalise this as follows.

ExamineSeedNeighbourhood($G = (V, E)$, $s \in V$)
1 $M := \emptyset$.
2 compute shortest distances from s to each $v \in V$ up to distance $h_{\max}$.
3 $\mathcal{L}^i(s) := \{v \in V \mid \mathrm{d}(s, v) = i\}$ $\forall i \in [h_{\min}, \ldots, h_{\max}]$.
4 **for** each $i \in [h_{\min}, \ldots, h_{\max}]$ **do**
5 $\quad M := M \cup$ ExamineISOLevel($G, \mathcal{L}^i(s), s$).
6 **return** M

Algorithm 4.3: Subroutine to inspect the neighbourhood of a node.

Definition 4.2. *A set S of sensors is called an ϵ-good sensor distribution if*

$$\forall p \in \mathcal{R} : \exists s \in S \; : |sp| \leq \epsilon.$$

In other words, for a set of sensors to be ϵ-good, there must be a sensor within distance ϵ from each point outside the holes. This is a reasonable assumption if one also wants to ensure that in 'regular' areas of $\mathbb{R}^2$, where sensors have survived, sensing coverage is guaranteed.

Mimicking the continuous case, we pick a set of nodes I in the network to serve as seeds and determine hop-distances in a bounded neighbourhood of $h_{\max}$ hops around each $s \in I$. We then examine the isolevels around s and try to detect whether these isolevels form closed annuli or are broken up. This can be done by repeated shortest path computations *within* the subgraph induced by the nodes in the respective isolevel $\mathcal{L}$. We first compute graph distances $\mathrm{d}^{\mathcal{L}}(v_1, \cdot)$ within $\mathcal{L}$ from some arbitrary node $v_1 \in \mathcal{L}$. We then set v_2 to be a node furthest from v_1. Let v_2' be the node on a shortest path from v_1 to v_2 at distance $\lfloor \frac{\mathrm{d}(v_1, v_2)}{2} \rfloor$. We remove all nodes within a two-hop neighbourhood of v_2' to get the the subgraph $\mathcal{L} \setminus \{v' : d(v', v_2') \leq 2\}$. If there still exists a path from v_1 to v_2 in this induces subgraph, we take this as an indication that the respective isolevel has a circular shape. Hence we return without marking any nodes as being close to a boundary. Otherwise we compute distances $\mathrm{d}^{\mathcal{L}}(v_2, \cdot)$ from v_2. Let v_3 be a node furthest from v_2. After the same check for connectivity without the neighbourhood of a node v_3' halfway between v_2 and v_3, we take it for granted that the isolevel is broken. Furthermore we mark the nodes v_2 and v_3 as being close to the extreme points of that isolevel.

The high-level description of the algorithm can be found in Algorithm 4.2. A more detailed description of the node neighbourhood and isolevel examination can be found in Algorithm 4.3 and Algorithm 4.4. See Figure 4.5 for a snapshot of the algorithm when examining the isolevels three to five from one seed node.

ExamineISOLevel$\left(G = (V, E), \mathcal{L}^i(s) \subseteq V, s \in V\right)$

1 **if** $\mathcal{L}^i(s)$ induces non-connected sub-graph
2 **return** $\emptyset$
3 Choose $v_1 \in \mathcal{L}^i(s)$ arbitrary
4 Compute hop-distance function $\mathrm{d}^{\mathcal{L}^i(s)}(v_1, \cdot)$ from v_1 in $\mathcal{L}^i(s)$
5 $v_2 = \operatorname{argmax}\left(\mathrm{d}^{\mathcal{L}^i(s)}(v_1, \cdot)\right)$
6 $v_2' = v : \mathrm{d}^{\mathcal{L}^i(s)}(v_1, v) = \lfloor \mathrm{d}^{\mathcal{L}^i(s)}(v_1, v_2)/2 \rfloor$ on shortest path from v_1 to v_2
7 **if** $\exists$ path $v_1, \ldots, v_2$ in $\mathcal{L}^i(s) \setminus \{v \in \mathcal{L}^i(s) \mid \mathrm{d}(v, v_2') \leq 2\}$
8 **return** $\emptyset$
9 Compute hop-distance function $\mathrm{d}^{\mathcal{L}^i(s)}(v_2, \cdot)$ from v_2 in $\mathcal{L}^i(s)$
10 $v_3 = \operatorname{argmax}\left(\mathrm{d}^{\mathcal{L}^i(s)}(v_2, \cdot)\right)$
11 $v_3' = v : \mathrm{d}^{\mathcal{L}^i(s)}(v_2, v) = \lfloor \mathrm{d}^{\mathcal{L}^i(s)}(v_2, v_3)/2 \rfloor$ on shortest path from v_2 to v_3
12 **if** $\exists$ path $v_2, \ldots, v_3$ in $\mathcal{L}^i(s) \setminus \{v \in \mathcal{L}^i(s) \mid \mathrm{d}(v, v_3') \leq 2\}$
13 **return** $\emptyset$
14 **return** $\{v_2, v_3\}$

Algorithm 4.4: Subroutine to examine an Isolevel.

Restricting the seed set to a maximal independent set allows us to bound the overall running time of the algorithm. The isolevel examination routine only considers isolevels that are connected, since arguing about distances within small connected components seems rather difficult. Nevertheless we will show later in the theoretical analysis that our algorithm still doesn't miss any boundary. This is true because for any point p on the boundary of a hole, there is at least one seed node $s \in I$ such that some connected isolevel around s is cut near p. Hence some node near p is actually marked as being on the boundary.

The outcome of the algorithm is a set of marked nodes that hopefully are all close to some boundary and identifying all holes. As the marking happens independently, it might be the case that a node is marked several times or neighbouring nodes are marked. For a compact representation it might be desirable to only keep a maximal independent set of the marked nodes. In fact, this is what we have done in our implementation to allow for better visual inspection of the results. On the other hand, if a closed boundary representation is desired, one can locally compute connecting paths between the nodes of the maximal independent set of the marked nodes.

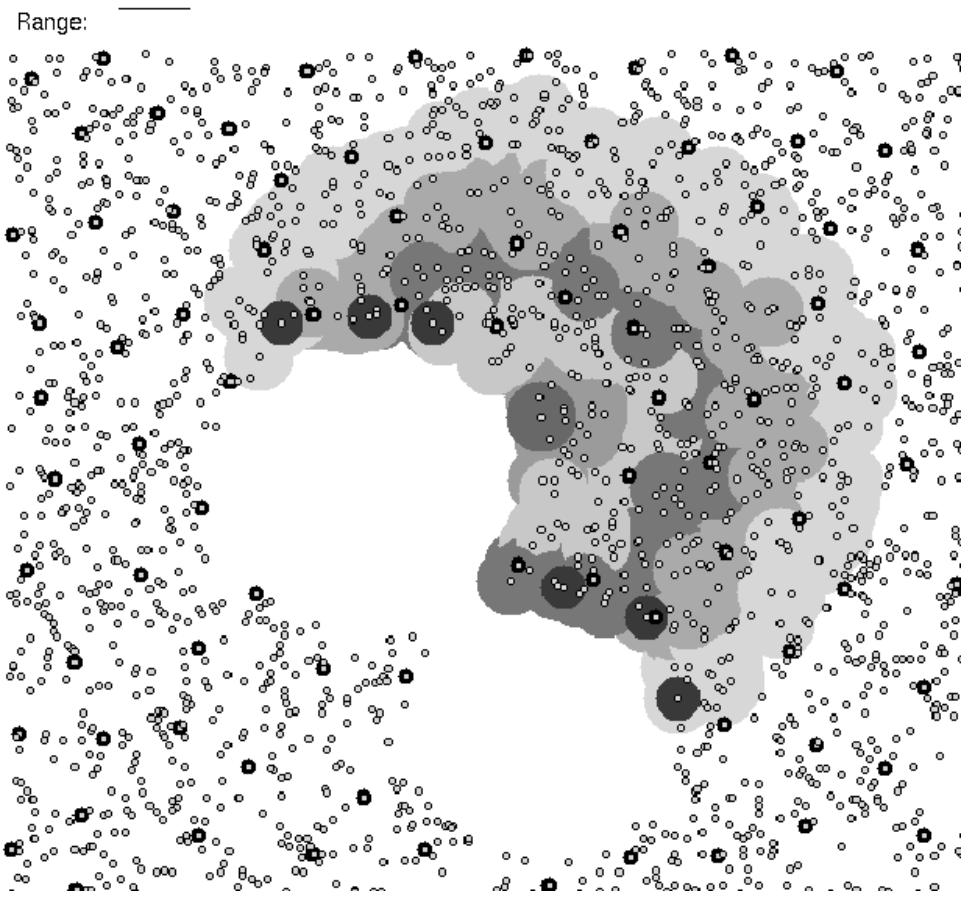

Figure 4.5: Snapshot while executing the algorithm. Isolevels examined from one seed and marked nodes where levels 3,4,5 are broken (large black circles). Small black circles denote the seed set I, little dots the remaining nodes.

4.3 Runtime

We now analyse the runtime of our algorithm. Obviously, the algorithm itself could run on any undirected graph, be it an unit disk graph or not. However, for arbitrary graphs the result of the algorithm does not hold any special meaning. The UDG property is also needed to prove a linear runtime.

For our proof of the runtime we first discuss the runtime of the isolevel examination subroutine given by Algorithm 4.4. This result can then be easily extended to compute the runtime needed to inspect the neighbourhood of each node in the maximal independent set. These rather simple results do not depend on the fact that the underlying graph is an unit disk graph. However, in the final proof we

need this property to show that the algorithm has linear runtime.

Since the algorithm operates on various subsets of the whole graph, the following definition will come in handy.

Definition 4.3. *For an undirected graph $G(V, E)$ and a set of vertices $U \subseteq V$ defined the induced edge set*

$$E(U) := \{e = (u, v) \in E : u \in U\}$$

Lemma 4.1. *Let $G(V, E)$ be a connected UDG and $\mathcal{L}^i(s)$ the ith isolevel for some node $s \in V$ and constant $i \in \mathbb{N}$. Then the examination of isolevel $L := \mathcal{L}^i(s)$ by the isolevel examination subroutine (see Algorithm 4.4) needs time at most $c|E(L)|$ for some positive constant c.*

Proof. Computing a hop-distance function on L can be done by simply applying BFS to L. This computation also provides nodes at maximum distance and half the maximum distance induced by the hop-distance function. Marking all nodes in a 2-neighbourhood of a vertex as removed can also be done by using BFS on L. Finally, checking if a path between two nodes exists is just another application of BFS on L without the 2-neighbourhood of some vertex. Obviously, BFS touches each edge in $E(L)$ at most a constant number of times. Hence the examination of the isolevel L can be done in time at most $c|E(L)|$ for some constant $c > 0$. □

Corollary 4.1. *Let $s \in V$ be a node of a connected UDG $G(V, E)$. Denote by $N := N_{h_{\max}}(s) := \{v \in V : d_G(v, s) \leq h_{\max}\}$ the set of nodes within hop-distance $h_{\max}$ of s. Then Algorithm 4.3 spends time at most $c|E(N)|$ on input $G(V, E)$ and s for some positive constant c.*

Proof. Computation of the hop distance and of the isolevels can be done by a simple BFS run constrained to maximum depth $h_{\max}$ on G. Hence this step is linear in $|E(N)|$.

Now consider the examination of the at most $h_{\max}$ isolevels. By Lemma 4.1 isolevel $\mathcal{L}^i(s)$ can be examined in time at most $c_i|E(\mathcal{L}^i(s))|$ for some constant $c_i > 0$. Since each isolevel $\mathcal{L}^i(s)$ is contained in $E(N)$ the time needed for the examination is bound by

$$\sum_{1 \leq i \leq h_{\max}} c_i \left|E(\mathcal{L}^i(s))\right| \leq h_{\max} \max_{1 \leq i \leq h_{\max}} c_i \left|E(\mathcal{L}^i(s))\right| \leq \left(h_{\max} \max_{1 \leq i \leq h_{\max}} c_i\right) |E(N)| .$$

Since $h_{\max}$ and all c_i are positive constants this concludes the proof. □

Theorem 4.1. *For a connected UDG $G(V, E)$ the hole detection algorithm (see Algorithm 4.2) runs in time $O(|V| + |E|)$.*

Proof. First the algorithm has to compute a maximal independent set $I \subseteq V$. This can easily be done in a greedy fashion in time $O(|V| + |E|)$. For this we repeatedly select a node from the graph and delete its neighbourhood.

The second part consists of the computation of the isolevels for each node in the independent set and their examination. By Corollary 4.1, the algorithm can do this in time $c_s|E(N_{h_{\max}}(s))|$ for each node s in the independent set. Hence with constant $c := \max_{s \in I} c_s$ the total runtime is

$$\sum_{s \in I} c_s \left|E(N_{h_{\max}}(s))\right| \leq c \sum_{s \in I} \sum_{e \in E} 1_{E(N_{h_{\max}}(s))}(e) = c \sum_{e \in E} \sum_{s \in I} 1_{E(N_{h_{\max}}(s))}(e),$$

where $1_F(e) := 0$ if $e \notin F$ and $1_F(e) := 1$ if $e \in F$ is the characteristic function of some subset $F \subseteq E$.

The last sum can also be read as the number of nodes in the independent set that are at distance at most $h_{\max}$ from one of the endpoints of the edge $e \in E$. To show that this sum is bound by a constant, we now need the fact that the graph under consideration is a unit disk graph. Consider one of the endpoints of $e = (u, v)$, say u. Draw a ball of radius $\frac{1}{2}$ around each node in the independent set with hop-distance at most $h_{\max}+1$ from u. Since each two nodes in the independent set have euclidean distance more than one (or there would be an edge between them) those balls are disjoint. On the other hand, those balls are all contained in a ball of radius $h_{\max} + 2$ around the node u. But then there can be at most

$$\frac{\pi(h_{\max} + 2)^2}{\pi\left(\frac{1}{2}\right)^2} = O\left(h_{\max}^2\right)$$

such balls. Thus we can conclude that the runtime of the algorithm is

$$\sum_{s \in I} c_s \left|E(N_{h_{\max}}(s))\right| \leq c \sum_{e \in E} \sum_{s \in I} 1_{E(N_{h_{\max}}(s))}(e) = c \sum_{e \in E} O\left(h_{\max}^2\right) = O\left(|E|\right).$$

□

4.3.1 Distributed and Localised Implementation

For application in a wireless sensor network scenario, it is important that the employed algorithms can be implemented without a centralised control. Fortunately, the formulation of our algorithm allows for a straightforward localised implementation. Many algorithms, as for example the one given in [MW05], are known for computing a maximal independent set in a distributed manner. The isolevel examinations themselves are inherently local (restricted to a constant size neighbourhood of the respective seed nodes). This is an improvement on the heuristic approach presented in [Fun05]. In that algorithm, four distance functions over the

whole network need to be computed. Furthermore the (particularly outer) isolevels computed by that approach extend over a long distance within the network and cannot be examined by local computations.

4.4 Correctness

After discussing the runtime of our algorithm in the last section, we will now show that it works as expected. In other words, we will show that under certain conditions on the geometry of the holes there exists a lower bound on the node density (in the area without holes) such that

a) for each point on a hole boundary, the algorithm marks a sensor node close to it,

b) the algorithm only mark such nodes that are close to a hole boundary.

While the previous section only required that the input is a (connected) unit disk graph, we need additional conditions to prove the correctness.

For one, we will only consider circular holes. Assume that all holes have radius at least r_{hole} and are at least some distance Δ_{hole} apart from each other.

We will now prove several Lemmata, each of which requires certain constraints on the input parameters Δ_{hole}, r_{hole} and ϵ discussed above. Those constraints will also depend on the choice of the algorithmic parameters $h_{\min}$ and $h_{\max}$ that determine the minimum and maximum isolevel inspected by the algorithm.

At the end, we will collect all the implied constraints on these parameters and show that they can be chosen such that all Lemmata hold for the following choice of the parameters. Assume that the algorithm inspects isolevels $h_{\min} = 4$ to $h_{\max} = 8$. If all holes have radius at least $r_{\text{hole}} = 115$, the minimum distance between two holes is at least $\Delta_{\text{hole}} = 18$, and the region is sampled with $\epsilon \leq 1/64$, then we can guarantee correctness of the output.

4.4.1 Isolevels and Containing Annuli

We now argue about the shape of the isolevels. If no holes are present, $\mathcal{L}^0(s)$ is obviously just a unit disk. All other isolevels $\mathcal{L}^i(s), i \geq 1$, should be similar to annuli in this case, give or take some multiples of ϵ.

In the presence of holes however, the shape becomes more interesting. We will now show that in this case an isolevel is still similar to an annuli (without the hole of course). More precisely, we show that it both contains a somewhat fat annulus and is contained in a not too large annulus.

Define the disk of radius $r \in \mathbb{R}$ around a point $p \in \mathbb{R}^2$ by $\mathcal{B}(p, r)$. Then we have the following Lemma.

Lemma 4.2. *Assume each point in $\mathcal{R}$ is at most $\epsilon \leq \frac{1}{4}$ away from a sensor node. Then for all $0 \leq i \leq \frac{\Delta_{hole}}{2} - 1$ the following holds. If the radius of each hole is at least $r_{hole} \geq \frac{(i+2)^2}{8(\sqrt{\epsilon}-\epsilon)}$ then*

$$a) \mathcal{B}(s, i - 4i\epsilon + 1) \cap \mathcal{R} \subseteq \bigcup_{0 \leq j \leq i} \mathcal{L}^j(s)$$

$$b) \bigcup_{0 \leq j \leq i} \mathcal{L}^j(s) \subseteq \mathcal{B}(s, (i + 1)) \cap \mathcal{R}.$$

Proof. Claim b) immediately follows from the fact that $\mathcal{L}^i(s)$ contains points with hop-distance i from s and each hop crosses a distance of at most 1.

To prove claim a) we use induction on i. For $i = 0$, we have

$$\mathcal{B}(s, i - 4i\epsilon + 1) \cap \mathcal{R} = \mathcal{B}(s, 1) \cap \mathcal{R} = \mathcal{L}^i(s).$$

Now consider the case $i + 1$. By induction, the Lemma holds for i and we have $\mathcal{B}(s, i - 4i\epsilon + 1) \cap \mathcal{R} \subseteq \bigcup_{0 \leq j \leq i} \mathcal{L}^j(s)$. Let $p \in \mathcal{R}$ be a point with

$$i - 4i\epsilon + 1 < |sp| \leq (i + 1) - 4(i + 1)\epsilon + 1.$$

Consider the intersection $I := \mathcal{B}(p, 1) \cap \mathcal{B}(s, i - 4i\epsilon + 1)$ of the unit disk around p with the disk contained in the union of the first i isolevels. Our goal is to find a disk of radius ϵ in I that is also contained in $\mathcal{R}$. Such a disk would (since it is contained in $\mathcal{R}$) contain at least one sensor node. This sensor node would then be at distance at most 1 from p and inside the union of the first i isolevels. Thus, p must be in the union of the first $i + 1$ isolevels, proving the Lemma.

The difficulty lies in the fact that not necessarily all of I has to be contained in $\mathcal{R}$. However, we can use the fact that each hole is circular and rather large to show that there indeed exists a suitable ϵ-disk in I.

For this we will now construct two points z, z' and show that one of them is the centre of a suitable ϵ-disk. Consider the point m on the line $\overline{ps}$ with distance $|pm| = 1 - 2\epsilon$. Let M be the line through m perpendicular to $\overline{ps}$. Now define z and z' as the two opposite points on M that are distance $|zm| = |z'm| = \sqrt{\epsilon}$ away from m. See Figure 4.6 for a picture of this construction.

We will show that the ϵ-disks around both points lie in I by proving that z and z' lie by at least ϵ inside the corresponding disks around p and s. Using the Pythagorean theorem and the fact that $3\epsilon^2 \leq \epsilon$ we get

$$\begin{aligned} |pz| = |pz'| &= \sqrt{(1 - 2\epsilon)^2 + \sqrt{\epsilon}^2} \\ &= \sqrt{1 - 3\epsilon + 4\epsilon^2} \\ &\leq \sqrt{1 - 3\epsilon + \epsilon + \epsilon^2} \end{aligned}$$

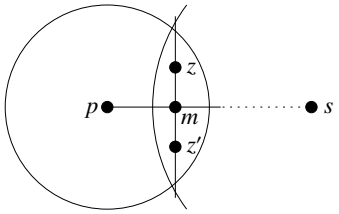

Figure 4.6: Construction of the points z and z'.

$$= 1 - \epsilon$$

hence the ϵ-disks around both points are inside $\mathcal{B}(p, 1)$.

Substituting $x = i - 4i\epsilon + 1$ and again using the Pythagorean theorem yields

$$\begin{aligned}
|sz| = |sz'| &= \sqrt{(|sp| - |pm|)^2 + \sqrt{\epsilon}^2} \\
&\leq \sqrt{((x + 1 - 4\epsilon) - (1 - 2\epsilon))^2 + \epsilon} \\
&= \sqrt{(x - 2\epsilon)^2 + \epsilon} \\
&= \sqrt{x^2 - 4x\epsilon + 4\epsilon^2 + \epsilon} \\
&= \sqrt{(x - \epsilon)^2 + (-2x\epsilon + 3\epsilon^2 + \epsilon)} \\
&= \sqrt{(x - \epsilon)^2 + (-2i\epsilon + 8i\epsilon^2 - 2\epsilon + 3\epsilon^2 + \epsilon)} \\
&= \sqrt{(x - \epsilon)^2 + (2i\epsilon(4\epsilon - 1) + \epsilon(3\epsilon - 1))} \\
&\leq \sqrt{(x - \epsilon)^2} \\
&= i - 4i\epsilon + 1 - \epsilon
\end{aligned}$$

where the last inequality follows from $\epsilon \leq \frac{1}{4}$. This establishes that $\mathcal{B}(z, \epsilon) \subseteq I$ and $\mathcal{B}(z', \epsilon) \subseteq I$.

To complete the proof we need to show that (at least) one of the ϵ-disks lies in $\mathcal{R}$. Since $\Delta_{\text{hole}} \geq 2(i + 1)$ only one hole can interfere with $\mathcal{L}^i(s)$. Assume w.l.o.g. that the centre of the respective circular hole o lies on the same side of $\overline{ps}$ as z'. In the worst case, i.e. obstructing as much from I as possible, o has both s and p on its boundary and 'eats away' the whole portion of I that lies below $\overline{ps}$. S We now will calculate an upper bound on the minimum radius that o needs to have to not interfere with $\mathcal{B}(z, \epsilon)$. As z lies $\sqrt{\epsilon}$ above $\overline{ps}$ the circular segment formed by $\overline{ps}$ and o must have height at most $\sqrt{\epsilon} - \epsilon$. This ensures that z is at least ϵ away from o, and thus that $\mathcal{B}(z, \epsilon)$ is contained in $\mathcal{R}$. With the height of the circular segment fixed to $h = \sqrt{\epsilon} - \epsilon$ the worst case radius will occur for the maximum possible chord length $|ps|$. By construction this is at most $c = |ps| \leq (i + 1) - 4(i + 1)\epsilon + 1$.

Using $j := i + 1$ the minimum radius needed can be bound by

$$
\begin{aligned}
r_{\text{hole}} &\leq \frac{h}{2} + \frac{c^2}{8h} \\
&= \frac{1}{2}(\sqrt{\epsilon} - \epsilon) + \frac{(j - 4j\epsilon + 1)^2}{8(\sqrt{\epsilon} - \epsilon)} \\
&= \frac{1}{2(\sqrt{\epsilon} - \epsilon)} \left(\left(\sqrt{\epsilon} - \epsilon\right)^2 + \frac{(j - 4j\epsilon + 1)^2}{4} \right) \\
&= \frac{1}{2(\sqrt{\epsilon} - \epsilon)} \left(\epsilon \left(1 - \sqrt{\epsilon}\right)^2 + \frac{j^2 + 16j^2\epsilon^2 + 1 - 8j^2\epsilon + 2j - 8j\epsilon}{4} \right) \\
&= \frac{1}{2(\sqrt{\epsilon} - \epsilon)} \left(\left(\frac{1}{4} + 4\epsilon^2 - 2\epsilon\right) j^2 + \left(\frac{1}{2} - 2\epsilon\right) j + \epsilon \left(1 - \sqrt{\epsilon}\right)^2 + \frac{1}{4} \right) \\
&= \frac{1}{2(\sqrt{\epsilon} - \epsilon)} \left(\left(\frac{1}{4} - 2\epsilon\,(1 - 2\epsilon)\right) j^2 + \frac{1}{2} j + \frac{1}{4} + \epsilon \left(\left(1 - \sqrt{\epsilon}\right)^2 - 2j \right) \right) \\
&\leq \frac{1}{2(\sqrt{\epsilon} - \epsilon)} \left(\frac{1}{4} j^2 + \frac{1}{2} j + \frac{1}{4} \right) \\
&\leq \frac{(j + 1)^2}{8(\sqrt{\epsilon} - \epsilon)}.
\end{aligned}
$$

□

From this proof we can immediately deduce that the ith isolevel is "sandwiched" by two similar-sized annuli.

Corollary 4.2. *Under the assumptions from Lemma 4.2, the ith isolevel $\mathcal{L}^i(s)$ is contained in the intersection of the annulus $\mathcal{A}^i_{\text{outer}}(s) := \mathcal{A}(s, i - 4(i-1)\epsilon, i + 1)$ with $\mathcal{R}$.*

Corollary 4.3. *Under the assumptions from Lemma 4.2, the ith isolevel $\mathcal{L}^i(s)$ contains the intersection of the annulus $\mathcal{A}^i_{\text{inner}}(s) := \mathcal{A}(s, i, i - 4i\epsilon + 1)$ with $\mathcal{R}$.*

4.4.2 Non-Contractible Graph Cycles

When our algorithm examines the ith isolevel (or more precisely the set of nodes contained in level $\mathcal{L}^i(s)$) it basically tries to decide whether there exists a cycle containing s in its interior. We will now show that such a cycle exists if and only if the isolevel is not broken.

Definition 4.4. *We call a cycle in the graph induced by the nodes in the ith isolevel a* non-contractible cycle *if its corresponding polygon cannot be contracted to a single point without sweeping over the seed point s.*

Lemma 4.3. *Let* $4 \leq i < \frac{1-2\epsilon}{4\epsilon}$, $\epsilon \leq \frac{1}{12}$ *and* $\mathcal{A}_{\text{inner}}(s, i, i - 4i\epsilon + 1) \subseteq \mathcal{L}^i(s)$. *Then the isolevel examination subroutine given in Algorithm 4.4 finds a non-contractible cycle and returns without marking any nodes as being close to a boundary.*

Proof. Since $i < \frac{1-2\epsilon}{4\epsilon}$ the annulus has width at least

$$\begin{aligned}\text{width}(\mathcal{A}_{\text{inner}}(s, i, i - 4i\epsilon + 1)) &= 1 - 4i\epsilon \\ &> 1 - 4\epsilon(\frac{1 - 2\epsilon}{4\epsilon}) \\ &= 2\epsilon.\end{aligned}$$

Thus, for each point halfway between the inner and outer border there must be a sensor node inside the annulus. While it is clear that the isolevel contains a non-contractible cycle, we need to show that the algorithm will be able to find one.

Let v_1, v_2, v_2' be the sensors picked by the algorithm. First consider the point p where the line through v_1 and the middle point s of the isolevel cuts the inner boundary of the isolevel opposite to v_1. There must be a node s_p in $\mathcal{L}^i(s)$ that is at most 2ϵ away from p. By Corollary 4.2 we know that the inner boundary of $\mathcal{L}^i(s)$ is a circle of radius at least $i - 4(i - 1)\epsilon$. Since $i \geq 4$ and $\epsilon \leq \frac{1}{12}$ its circumference is

$$\begin{aligned}c &= 2\pi(i - 4(i - 1)\epsilon) \\ &\geq 2\pi(4 - 12\epsilon) \\ &\geq 6\pi.\end{aligned}$$

But this means that $\mathrm{d}(v_1, v_2) \geq \mathrm{d}(v_1, s_p) \geq 9$. Hence the node v_2' is at least four hops away from v_1 and v_2. Thus when the algorithm removes the 2-neighbourhood $\{v \in \mathcal{L}^i(s) : \mathrm{d}(v, v_2') \leq 2\}$ of v_2' the remaining graph is still connected and v_1 and v_2 are not removed. Because of this the algorithm can find an alternative path from v_1 to v_2 in $\mathcal{L}^i(s) \setminus \{v \in \mathcal{L}^i(s) : \mathrm{d}(v, v_2') \leq 2\}$. But this path cannot pass near v_2' since the removal of its 2-hop neighbourhood has cut $\mathcal{L}^i(s)$ there. Hence the original path from v_1 to v_2 computed by the algorithm together with the path from v_1 to v_2 that the algorithm finds in the augmented isolevel form a non-contractible cycle. □

Corollary 4.4. *The hole detection algorithm will not mark any nodes in* $\mathcal{L}^i(s)$ *if the isolevel is not cut by a hole.*

Let us now show that whenever our algorithm marks a node, there is some boundary point nearby

Lemma 4.4. *Let* $4 \leq i < \frac{1-4\epsilon}{4\epsilon}$ *and let isolevel* $\mathcal{L}^i(s)$ *be cut by hole* $o \in \mathcal{O}$. *Assume the isolevel examination subroutine given in Algorithm 4.4 on input* $\mathcal{L}^i(s)$ *marks nodes* v_2, v_3 *as boundary nodes. Then there exist boundary points* $p_1, p_2 \in \delta o$ *such that* $|v_i p_i| \leq 2.8$. *Furthermore it holds that* $d^{\mathcal{L}^i(s)\setminus o}(v_2, v_3) \geq \pi(i - 2(i - 1)\epsilon)$.

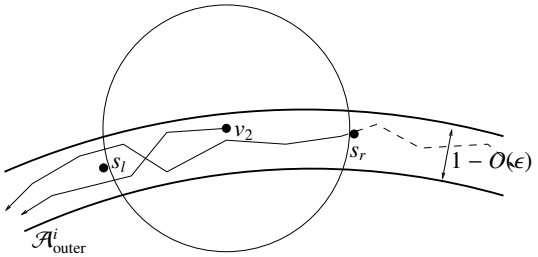

Figure 4.7: Situation for v_2 if nodes are marked.

Proof. Consider the neighbourhood within distance 2.8 of v_2. If there is some hole boundary nearby, we're done, so assume otherwise. Consider the intersection points of the boundary of $\mathcal{A}^i_{\text{inner}}(s)$ and $\mathcal{B}(v_2, 2.8)$. Let t_{l1} and t_{l2} be the intersection points on one side and t_{r1}, t_{r2} be the intersection points on the other side. Further let h_l be the point on $\mathcal{B}(v_2, 2.8)$ halfway between t_{l1} and t_{l2} and h_r analogously the point halfway between t_{r1} and t_{r2}. Denote by s_l, s_r the closest nodes to h_l, h_r respectively. See Figure 4.7 for an illustration. Clearly s_l and s_r are contained within $\mathcal{A}^i_{\text{inner}}(s)$.

Obviously we have $\mathrm{d}^{\mathcal{L}^i(s)}(v_1, s_l) \leq \mathrm{d}^{\mathcal{L}^i(s)}(v_1, v_2)$ and $\mathrm{d}^{\mathcal{L}^i(s)}(v_1, s_r) \leq \mathrm{d}^{\mathcal{L}^i(s)}(v_1, v_2)$ since v_2 has maximum distance. Since s_l and s_r are at most 3 hops from v_2 we also have $\mathrm{d}^{\mathcal{L}^i(s)}(v_1, v_2) \leq \mathrm{d}^{\mathcal{L}^i(s)}(v_1, s_l) + 3$ and $\mathrm{d}^{\mathcal{L}^i(s)}(v_1, v_2) \leq \mathrm{d}^{\mathcal{L}^i(s)}(v_1, s_r) + 3$.

Without loss of generality let the shortest path from v_1 to v_2 pass by s_l. Since the algorithm marked some nodes as being close to the boundary it follows that no shortest path exists after removing the 2-neighbourhood $\{v \in \mathcal{L}^i(s) : \mathrm{d}(v, v_2') \leq 2\}$ of v_2'. Hence the shortest path from v_1 to s_r also has to pass by s_l or this would yield a different path to v_2. But because s_l and s_r lie on opposite sides of $\mathcal{B}(v_2, 2.8)$ this implies that $\mathrm{d}^{\mathcal{L}^i(s)}(v_1, s_l) + 4 \leq \mathrm{d}^{\mathcal{L}^i(s)}(v_1, s_r)$. This however contradicts the inequalities we derived above since combining them yields $\mathrm{d}^{\mathcal{L}^i(s)}(v_1, s_l) \leq \mathrm{d}^{\mathcal{L}^i(s)}(v_1, s_r) + 3$.

Now consider v_3. Obviously the second part of the isolevel examination subroutine given in Algorithm 4.4 is identical to the first part except that it starts from v_2. Hence the same argument as above shows that v_3 must be close to a boundary.

Finally the fact that $\mathrm{d}^{\mathcal{L}^i(s)\setminus o}(v_2, v_3) \geq \pi(i - 2(i-1)\epsilon)$ follows since o can cut away at most half of the isolevel. □

Together Corollary 4.4 and Lemma 4.4 show that the algorithm will only mark nodes close to a hole. To complete our proof of the correctness of the algorithm we still need to show that for each point on a hole boundary a node close to it will be marked.

Lemma 4.5. *Let $p \in \delta o$, $o \in O$ be a point on a hole boundary. Consider the isolevel examination subroutine given in Algorithm 4.4. There exists a seed node*

$s \in I$ such that for some $\mathcal{L}^i(s)$ with $i \in [h_{\min}, \ldots, h_{\max}]$ the subroutine reaches the last step. Furthermore, one of the nodes marked by the subroutine will be at distance at most 4.8 *from p.*

Proof. Consider the tangent $T_o(p)$ to o at p and some point q on $T_o(p)$ with $|pq| = h_{\max} - 3$. Because of the convexity of o and the fact that two holes are at least $\Delta_{\text{hole}} > h_{\max}$ apart, no hole is close to p. Due to the sampling condition we have $|qs_q| \leq \epsilon$ for some node s_q. Let $s \in I$ be the closest seed to s_q. Since a maximal independent set in a graph is also a dominating set we have $|s_q s| \leq 1$. Hence, by the triangle inequality $|sq| \leq 1 + \epsilon$ holds. Let s_p be the sensor closest to p. Then $|s_p p| \leq \epsilon$ holds. We have

$$\begin{aligned} |s_p s| &\leq |s_p p| + |pq| + |qs| \\ &\leq \epsilon + (h_{\max} - 3) + (1 + \epsilon) \\ &\leq h_{\max} - 1. \end{aligned}$$

Hence there is an isolevel $\mathcal{L}^i(s)$ of s which is considered by the subroutine and contains s_p.

As $h_{\max} < r_{\text{hole}}$, both the contained annulus $\mathcal{A}^i_{\text{inner}}(s)$ as well as the containing annulus $\mathcal{A}^i_{\text{outer}}(s)$ (and also $\mathcal{L}^i(s)$) are completely cut by o. But since $2h_{\max} < \Delta_{\text{hole}}$, i.e. there is no other hole 'nearby', they still remain connected. As the removal of the 2-hop neighbourhood of v_2' again cuts $\mathcal{L}^i(s)$ around v_2' there does not exist an alternative path from v_1 to v_2 (same argumentation for v_3'). Thus the algorithm does not return before the last step.

We still need to show that the algorithm marks a node near p. Since i is small compared to the radius of the obstacle o, $\mathcal{L}^i(s)$ is almost orthogonally cut by o. So the nodes within $\mathcal{L}^i(s)$ that are at most ϵ away from o are grouped into two small clusters of diameter at most two. Those clusters are far away from each other in the graph distance. According to Lemma 4.4 our algorithm marks one node in each of the clusters.

The distance of the marked node to p follows from the cluster diameter and Lemma 4.4. □

4.4.3 Plugging Everything Together

With the above Lemmata we can now prove the correctness of the algorithm. We require that we only have circular holes that are large and well-separated (relative to the communication range 1 of the sensor nodes). Also the sensor nodes must be distributed sufficiently dense. Note however, that there is no further requirement on the distribution itself. In particular, we do not require a uniform distribution. Under those assumptions, our algorithm faithfully marks nodes close to all points on the boundaries of all holes but nowhere else.

The previous Lemmata gave several conditions on the parameters Δ_{hole}, r_{hole}, $h_{\min}$, $h_{\max}$ and ϵ. For the inner annuli not getting too thin and the outer annuli not getting too thick, we want $4h_{\max}\epsilon \leq \frac{1}{2}$. By Lemma 4.2 we need $\epsilon \leq \frac{1}{4}$, $h_{\max} \leq \frac{\Delta_{\text{hole}}}{2} - 1$ as well as $r_{\text{hole}} \geq \frac{(h_{\max}+2)^2}{8(\sqrt{\epsilon}-\epsilon)}$. For Lemma 4.3 we require $h_{\min} \geq 4$, $h_{\max} < \frac{1-2\epsilon}{4\epsilon}$ and $\epsilon \leq \frac{1}{12}$. Additionally, for Lemma 4.4 we need $h_{\max} < \frac{1-4\epsilon}{4\epsilon}$ Lastly, Lemma 4.5 requires $(h_{\max} - h_{\min}) \geq 3$. We haven't tried hard to determine the best parameter values, but the following values allow us to state the main theorem.

Theorem 4.2. *Let O be a set of circular holes of radius at least $r_{hole} = 115$ such that all holes are at least $\Delta_{hole} = 18$ apart. Let S be a set of sensors that form an ϵ good sensor distribution in the non-hole area for $\epsilon = \frac{1}{64}$. Assume Algorithm 4.2 inspects isolevels between $h_{\min} = 4$ and $h_{\max} = 8$. Then it will mark a sensor node within distance 4.8 of each boundary point and will only mark nodes that are within distance 2.8 of a boundary point.*

4.5 Experimental Results

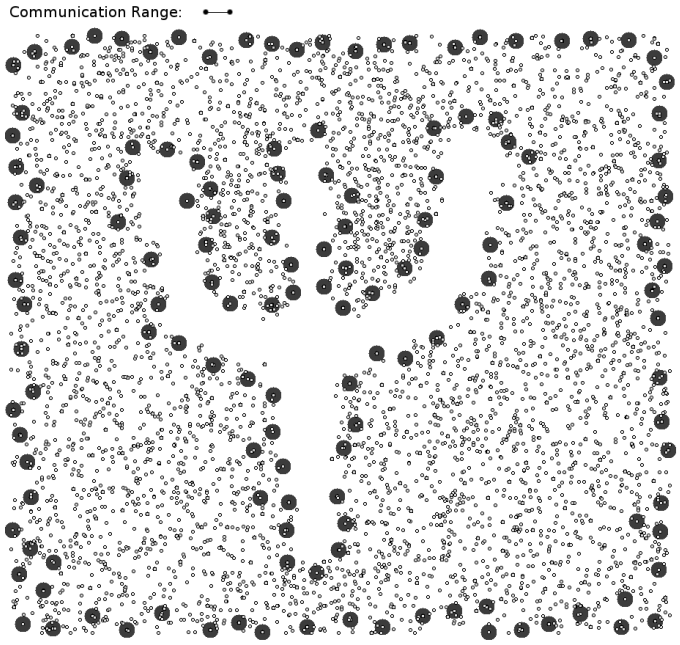

Figure 4.8: Example output of Algorithm 4.2. Around 5000 randomly distributed nodes, communication range of 30 and respective average degrees of the communication graph of 25.

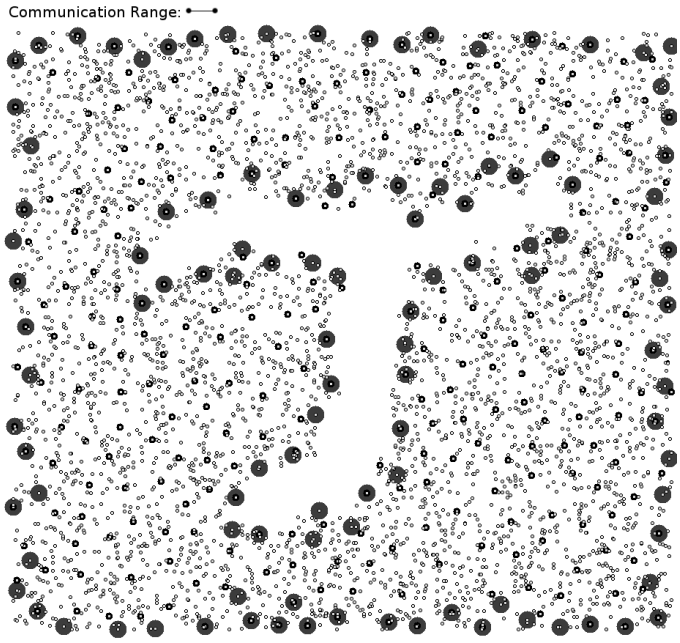

Figure 4.9: Example output of Algorithm 4.2. Around 5000 randomly distributed nodes, communication range of 33 and respective average degrees of the communication graph of 27.

We have implemented our algorithm and simulated different deployments of sensor nodes in the real plane. As to be expected, in practice the requirements on the node density as well as on the shape and distance between holes are by far not as strong as the theoretical analysis suggests. Our experiments were conducted with algorithm parameters $h_{\min} = 2$, $h_{\max} = 6$. Also, instead of removing all nodes within a 2-hop neighbourhood of v'_2 and v'_3 we only removed nodes within a 1-hop neighbourhood. We first generated sensor nodes uniformly at random and then built the unit-disk graph based on varying communication ranges.

Our observation was that for node distributions where the average degree in the resulting communication graph was above around 25, our algorithm seems to perform reasonably well. We note that for more regular node distributions like for example a grid or a perturbed grid, even average node degrees of down to 10 seem to give good results. Compared to the heuristic approach presented in [Fun05] the algorithm does slightly worse. This is mainly because we have avoided to add any heuristics that improve the practical performance but would make the algorithm somewhat different from the one we have proven correctness for.

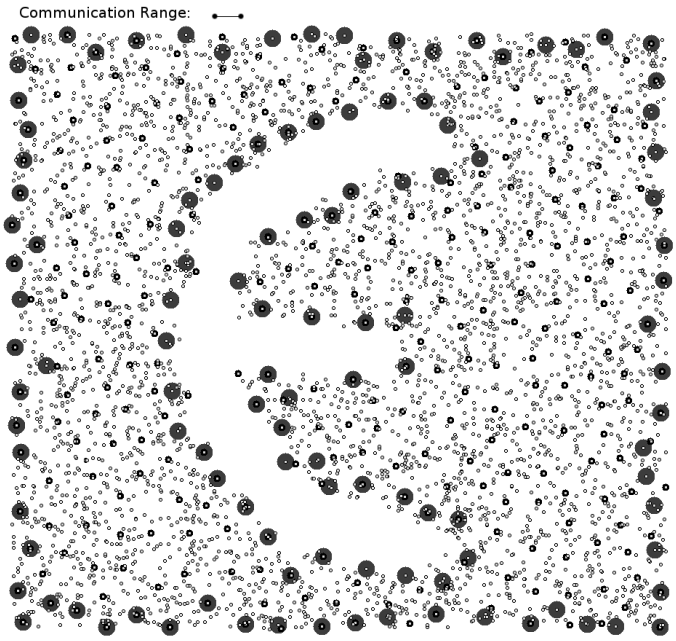

Figure 4.10: Example output of Algorithm 4.2. Around 5000 randomly distributed nodes, communication range of 33 and respective average degrees of the communication graph of 28.

Non-Uniform Node-Distributions

The whole reasoning why our algorithm actually works does not rely on a uniform density of the sensor nodes in non-hole regions. This is a large improvement over previous work like for example the algorithm presented in [FKP+04].

If one assumes a uniform distribution a much easier algorithm can be devised. Roughly speaking, one can determine if a node is close to some boundary or not by simply examining its degree in the communication graph, i.e. to how many neighbours it can talk. For nodes not close to some boundary, this should always roughly be the same number. On the other hand, if a node is close to any boundary, it should have less immediate neighbours that it can talk to.

The output of our algorithm for an input with non-uniform sensor distribution can be found in Figure 4.11.

Problematic Inputs

Of course there are settings where the algorithm does not perform as well as in the previously discussed examples. One shortcoming is that it has problems to distinguish holes that are close to each other. Recall that this also manifested

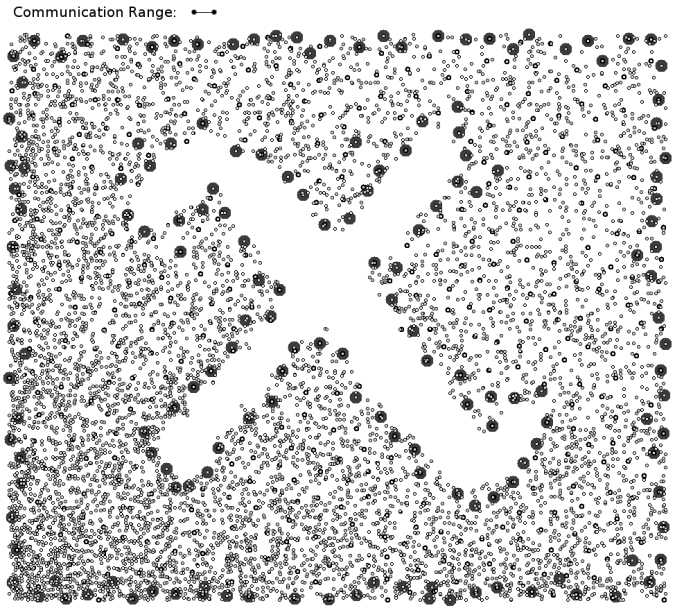

Figure 4.11: Example output of Algorithm 4.2 for a non-uniform node distribution.

itself in the analysis as we required a certain minimum distance Δ_{hole} between holes. A particular bad example is given in Figure 4.12. The problem is that for all seeds that could be used to detect the boundary nodes between the two holes the respective isolevels consist of more than one component. Hence the isolevel examination subroutine returns immediately without marking any nodes. Of course one might let the algorithm examine all the connected components of an isolevel if more than one exists. However, this would not allow for a proof of correctness anymore. Note however, that the actual hole distance needed for the algorithm to work is in practice much smaller than the bound on Δ_{hole} given in Theorem 4.2.

Another shortcoming of the algorithm is that it relies on the non-hole regions being sufficiently densely sampled. Otherwise we cannot ensure that the isolevels around a seed node form closed cycles. When decreasing the node density (or equivalently decreasing the communication range) the algorithm starts to break down more and more. See Figure 4.13, Figure 4.14 and Figure 4.15 for a sequence of outputs with decreasing communication ranges. The roughly equivalent value for ϵ is determined as $\epsilon \approx \frac{1}{\sqrt{\text{avg.deg.}}}$. In this particular sequence of examples this yields 0.18, 0.21, and 0.25.

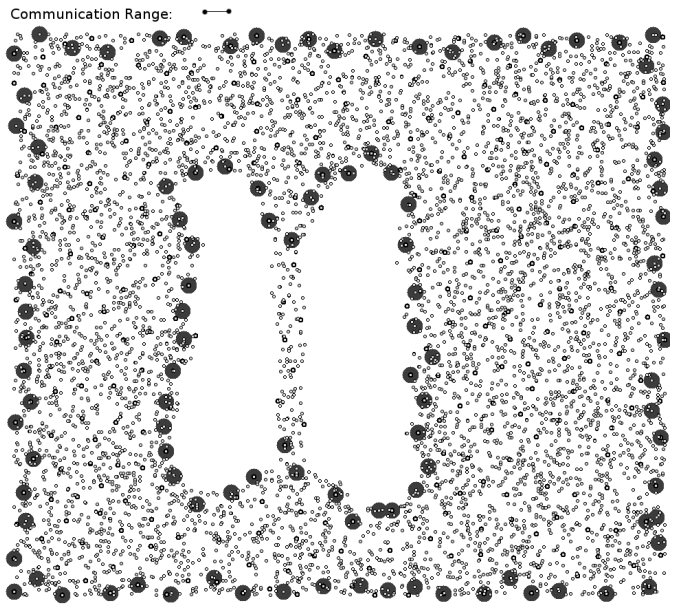

Figure 4.12: Example output of Algorithm 4.2 for two holes that are too close to each other to detect the boundary.

4.6 Conclusion and Discussion

The tight connection between geographic location and connectivity of wireless networks gives rise to many interesting problems. While the topology of a wireless network does not explicitly hold any geometric information, the fact that its connectivity is determined by geographic proximity relations allows for techniques to extract (part) of the underlying geometry. In this Chapter we explored that direction. However, there are still many challenging problems to be solved. Ultimately, of course, one would like to recover the exact relative positions of the network nodes. This is an NP-hard problem for general unit-disk graphs as was shown by Kuhn et al. in [KMW04]. However, a constant approximation has not yet been ruled out.

Apart from the application in the described scenario, the identification of hole boundaries or holes as such has in fact many other applications. A topological description of a network like 'The network has one large hole' can be much more compact than remembering the connectivity between all nodes. Furthermore, since you need many changes to close a large hole or create another one, such a topological descriptions tend to be more stable under small changes of nodes or network links.

There are routing schemata like [BGJ06] which are based on such topolog-

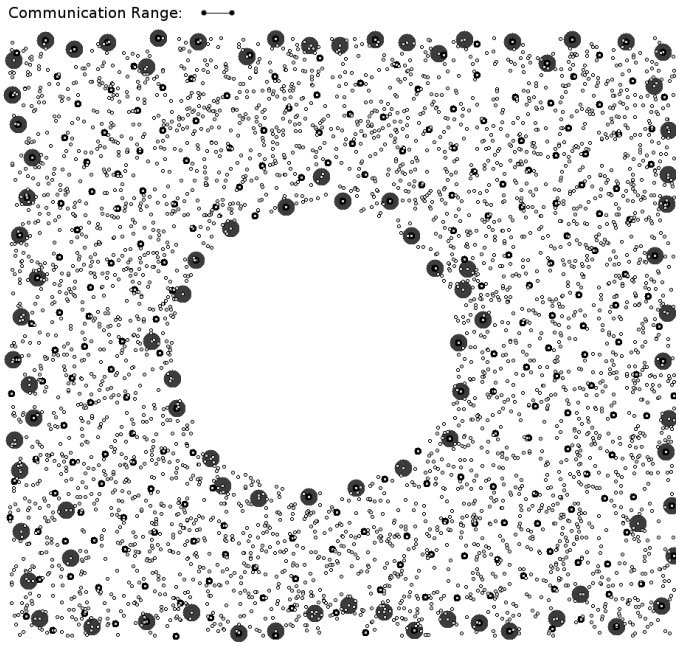

Figure 4.13: Example output of Algorithm 4.2 for for decreasing communication ranges with range 35 and average degree 31.

ical sketches of the network. Simulation results show that they enjoy an inherent stability against small network changes. The recent work by Fekete et al. [KFPF06] also describes methods how to use boundary recognition to obtain a compact topology sketch of the network.

There is also another class of problems where boundary detection algorithms are useful. In [RRP+03] Rao et al. discuss a method that – in spite of its NP-hardness – tries to find a faithful embedding of the whole unit-disk graph in the plane. Their algorithm bases its decisions on the assumption that hop-distances in an unit-disk graph in some sense approximate Euclidean distances in the plane. This doesn't have to be true in the presence of holes, though. In this case holes might cause the geodesic and the graph distance to differ vastly from the Euclidean distance. However, if holes are detected, then one could check whether a graph distance – measured by a shortest hop path in the graph – is 'truthful'. For this one must assure that the respective shortest path does not come close to any hole boundary. Otherwise it might not reflect Euclidean distances.

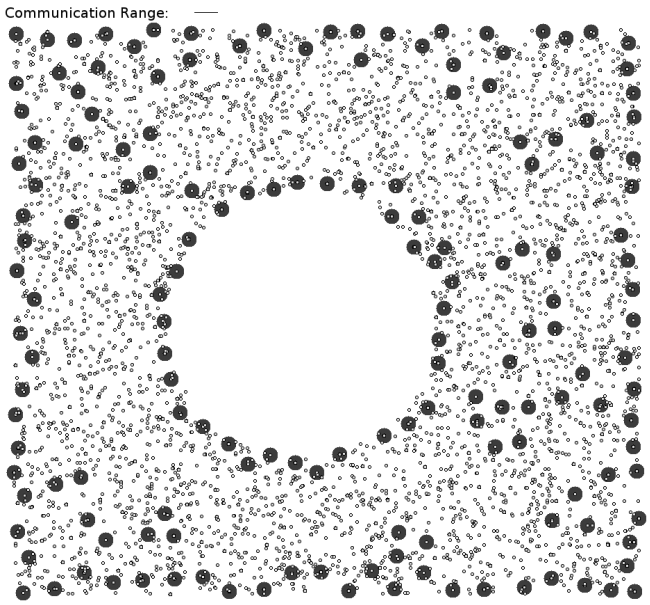

Figure 4.14: Example output of Algorithm 4.2 for for decreasing communication ranges with range 30 and average degree 23.

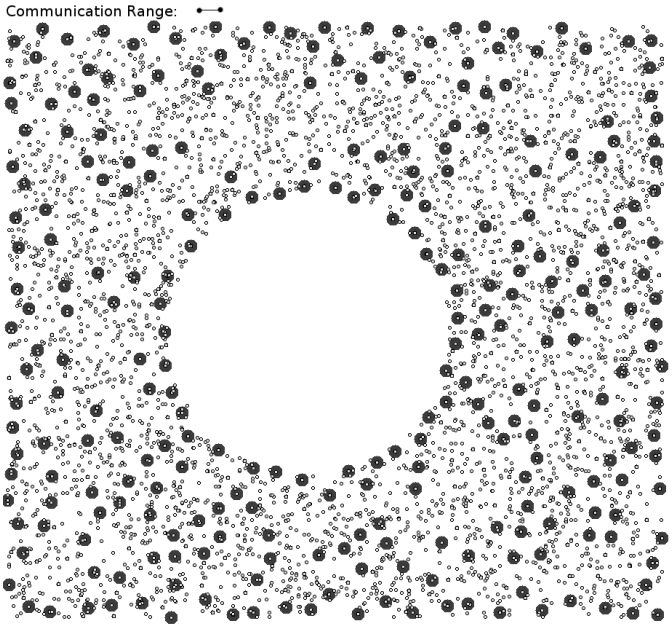

Figure 4.15: Example output of Algorithm 4.2 for for decreasing communication ranges with range 25 and average degree 16.

Appendix A

List of Publications

Conference Articles

- with Stefan Funke, Kurt Mehlhorn and Susanne Schmitt:
  "Controlled Perturbation for Delaunay Triangulation",
  SODA 2005 ([FKMS05])
- with Benjamin Doerr, Tobias Friedrich and Ralf Osbild:
  "Rounding of Sequences and Matrices, with Application",
  WAOA 2005 ([DFKO06a])
- with Stefan Funke:
  "Hole Detection or: How Much Geometry Hides in Connectivity?",
  SoCG 2006 ([FK06])
- with Benjamin Doerr, Tobias Friedrich and Ralf Osbild:
  "Unbiased Matrix Rounding",
  SWAT 2006 ([DFKO06b])
- with Benjamin Doerr:
  "Unbiased Rounding of Rational Matrices",
  FSTTCS 2006([DK06])
- with Benjamin Doerr and Tobias Storch:
  "Faster Evolutionary Algorithms by Superior Graph Representation",
  FOCI 2007 ([DKS07])
- with Dimo Brockhoff, Tobias Friedrich, Nils Hebbinghaus, Frank Neumann and Eckart Zitzler:
  "Do Additional Objectives Make a Problem Harder?",
  GECCO 2007 ([BFH+07])

- with Benjamin Doerr and Edda Happ:
  "A Tight Bound for the (1+1)-EA on the Single Source Shortest Path Problem",
  CEC 2007 ([DHK07])

- with Edda Happ, Daniel Johannsen and Frank Neumann:
  "Rigorous Analyses of Fitness-Proportional Selection for Optimizing Linear Function",
  GECCO 2008 ([HJKN08])

- with Benjamin Doerr and Thomas Jansen:
  "Comparing Global and Local Mutations on Bit String",
  GECCO 2008 ([DJK08])

- with Benjamin Doerr and Edda Happ:
  "Crossover Can Provably be Useful in Evolutionary Computation",
  GECCO 2008 ([DHK08])

Journal Articles

- with Dimo Brockhoff, Tobias Friedrich, Nils Hebbinghaus, Frank Neumann and Eckart Zitzler:
  "On the Effects of Adding Objectives to Plateau Function",
  IEEE Trans. on Evolutionary Computation, Vol. 13(3), 2009 ([BFH+09])

- with Benjamin Doerr and Edda Happ:
  "Tight Analysis of the (1+1)-EA for the Single Source Shortest Path Problem",
  Evolutionary Computation, Vol. 19(4), 2012 ([DHK11])

- with Benjamin Doerr and Edda Happ:
  "Crossover Can Provably Be Useful In Evolutionary Computation",
  Theoretical Computer Science, Vol. 425, 2012 ([DHK12])

Technical Reports

- "A Fast Root Checking Algorithm", ECG-TR-363109-0 ([Kle04b])

- with Stefan Funke, Kurt Mehlhorn and Susanne Schmitt: "Controlled Perturbation for Delaunay Triangulation", ACS-TR-121103-0 ([FKMS06])

Thesis

- “Controlled Perturbation for Voronoi Diagrams” Universit des Saarlandes, 2004, Diplomarbeit ([Kle04a])

Bibliography

[ABE98] Nina Amenta, Marshall Bern, and David Eppstein. The crust and the β-skeleton: Combinatorial curve reconstruction. *Graphical Models and Image Processing*, 60(2):125–135, 1998.

[AD11] Anne Auger and Benjamin Doerr, editors. *Theory of Randomized Search Heuristics: Foundations and Recent Developments*, volume 1 of *Series on Theoretical Computer Science*. World Scientific Publishing Co., 2011.

[AKOT03] Tetsuo Asano, Naoki Katoh, Koji Obokata, and Tokuyama Takeshi. Matrix rounding under the L_p-discrepancy measure and its application to digital halftoning. *SIAM Journal on Computing*, 32(6):1423–1435, 2003.

[AS08] Noga Alon and Joel H. Spencer. *The Probabilistic Method*. Series in Discrete Mathematics and Optimization. Wiley, third edition, 2008.

[Bac66] Michael Bacharach. Matrix rounding problems. *Management Science*, 12(9):732–742, 1966.

[Bar75] Zsolt Baranyai. On the factorization of the complete uniform hypergraph. In *Infinite and finite sets (Colloquium, Keszthely, 1973; dedicated to P. Erdős on his 60th birthday), Vol. I*, volume 10 of *Colloquia Mathematica Societatis János Bolyai*, pages 91–108. North-Holland, 1975.

[BBD+09] Surender Baswana, Somenath Biswas, Benjamin Doerr, Tobias Friedrich, Piyush P. Kurur, and Frank Neumann. Computing single source shortest paths using single-objective fitness functions. In *Proceedings of the Tenth Foundations of Genetic Algorithms Conference (FOGA)*, pages 59–66. ACM Press, 2009.

[BC04] Nadia Brauner and Yves Crama. The maximum deviation just-in-time scheduling problem. *Discrete Applied Mathematics*, 134(1-3):25–50, 2004.

[BFH+07] Dimo Brockhoff, Tobias Friedrich, Nils Hebbinghaus, Christian Klein, Frank Neumann, and Eckart Zitzler. Do additional objectives make a problem harder? In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 765–772. ACM Press, 2007.

[BFH+09] Dimo Brockhoff, Tobias Friedrich, Nils Hebbinghaus, Christian Klein, Frank Neumann, and Eckart Zitzler. On the effects of adding objectives to plateau functions. *IEEE Transactions on Evolutionary Computation*, 13(3):591–603, 2009.

[BGJ06] Jehoshua Bruck, Jie Gao, and Anxiao (Andrew) Jiang. Map: Medial axis based geometric routing in sensor networks. *Wireless Networks*, 13(6):835–853, 2006.

[BH04] Tom Bohman and Ron Holzman. Linear versus hereditary discrepancy. *Combinatorica*, 25(1):39–47, 2004.

[BS84] József Beck and Joel Spencer. Well-distributed 2-colorings of integers relative to long arithmetic progressions. *Acta Arithmetica*, 43:287–294, 1984.

[CCE85] Beverley D. Causey, Lawrence H. Cox, and Lawrence R. Ernst. Application of transportation theory to statistical problems. *Journal of the American Statistical Association*, 80(392):903–909, 1985.

[CE82] Lawrence H. Cox and Lawrence R. Ernst. Controlled rounding. *INFOR: Information Systems and Operational Research*, 20:423–432, 1982.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, third edition, 2009.

[Cox87] Lawrence H. Cox. A constructive procedure for unbiased controlled rounding. *Journal of the American Statistical Association*, 82(398):520–524, 1987.

[DFKO06a] Benjamin Doerr, Tobias Friedrich, Christian Klein, and Ralf Osbild. Rounding of sequences and matrices, with applications. In *Proceedings of the Third Workshop on Approximation and Online Algorithms (WAOA)*, volume 3879 of *Lecture Notes in Computer Science*, pages 96–109. Springer, 2006.

[DFKO06b] Benjamin Doerr, Tobias Friedrich, Christian Klein, and Ralf Osbild. Unbiased matrix rounding. In *Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 4059 of *Lecture Notes in Computer Science*, pages 102–112. Springer, 2006.

[DG10] Benjamin Doerr and Leslie A. Goldberg. Drift analysis with tail bounds. In *Proceedings of the Eleventh International Conference on Parallel Problem Solving from Nature (PPSN)*, volume 6238 of *Lecture Notes in Computer Science*, pages 174–183. Springer, 2010.

[DHK07] Benjamin Doerr, Edda Happ, and Christian Klein. A tight bound for the (1+1)-EA on the single source shortest path problem. In *Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC)*, pages 1890–1895. IEEE Press, 2007.

[DHK08] Benjamin Doerr, Edda Happ, and Christian Klein. Crossover can provably be useful in evolutionary computation. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO)*. ACM Press, 2008. Best Paper Awards.

[DHK11] Benjamin Doerr, Edda Happ, and Christian Klein. Tight analysis of the (1+1)-EA for the single source shortest path problem. *Evolutionary Computation*, 19(4):673–691, 2011.

[DHK12] Benjamin Doerr, Edda Happ, and Christian Klein. Crossover can provably be useful in evolutionary computation. *Theoretical Computer Science*, 425:17–33, 2012.

[DHN06] Benjamin Doerr, Nils Hebbinghaus, and Frank Neumann. Speeding up evolutionary algorithms through restricted mutation operators. In *Proceedings of the Ninth International Conference on Parallel Problem Solving from Nature (PPSN)*, volume 4193 of *Lecture Notes in Computer Science*, pages 978–987. Springer, 2006.

[Dij59] Edsger Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[DJ07] Benjamin Doerr and Daniel Johannsen. Adjacency list matchings — an ideal genotype for cycle covers. In *Proceedings of the Ninth Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 1203–1210. ACM Press, 2007.

[DJK08] Benjamin Doerr, Thomas Jansen, and Christian Klein. Comparing global and local mutations on bit strings. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO)*. ACM Press, 2008.

[DJW02] Stefan Droste, Thomas Jansen, and Ingo Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276(1-2):51–81, 2002.

[DJW12] Benjamin Doerr, Daniel Johannsen, and Carola Winzen. Multiplicative drift analysis. *Algorithmica*, 64(4):673–697, 2012.

[DK06] Benjamin Doerr and Christian Klein. Unbiased rounding of rational matrices. In *Proceedings of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 4337 of *Lecture Notes in Computer Science*, pages 200–211. Springer, 2006.

[DKS07] Benjamin Doerr, Christian Klein, and Tobias Storch. Faster evolutionary algorithms by superior graph representation. In *Proceedings of the First IEEE Symposium on Foundations of Computational Intelligence (FOCI)*, pages 245–250. IEEE Press, 2007.

[Doe00] Benjamin Doerr. Linear and hereditary discrepancy. *Combinatorics, Probability and Computing*, 9(4):349–354, 2000.

[Doe04] Benjamin Doerr. Linear discrepancy of totally unimodular matrices. *Combinatorica*, 24(1):117–125, 2004.

[Doe06] Benjamin Doerr. Generating randomized roundings with cardinality constraints and derandomizations. In *Proceedings of the 23rd Annual Symposium on Theoretical Aspects of Computer Science*, volume 3884 of *Lecture Notes in Computer Science*, pages 571–583. Springer, 2006.

[DT09] Benjamin Doerr and Madeleine Theile. Improved analysis methods for crossover-based algorithms. In *Proceedings of the Eleventh Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 247–254. ACM Press, 2009.

[Fel75] Ivan P. Fellegi. Controlled random rounding. *Survey Methodology*, 1(2):123–133, 1975.

[FF62] Lester R. Ford, Jr., and Delbert R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[FGG06] Qing Fang, Jie Gao, and Leonidas J. Guibas. Locating and bypassing holes in sensor networks. *Mobile Networks and Applications*, 11(2):187–200, 2006.

[FK06] Stefan Funke and Christian Klein. Hole detection or: "How much geometry hides in connectivity?". In *Proceedings of the 22nd Annual Symposium on Computational Geometry (SoCG)*, pages 377–385. ACM Press, 2006.

[FKMS05] Stefan Funke, Christian Klein, Kurt Mehlhorn, and Susanne Schmitt. Controlled perturbation for Delaunay triangulations. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1047–1056. Society for Industrial and Applied Mathematics, 2005.

[FKMS06] Stefan Funke, Christian Klein, Kurt Mehlhorn, and Susanne Schmitt. Controlled perturbation for Delaunay triangulations. Technical Report ACS-TR-121103-03, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 2006.

[FKP+04] Sándor P. Fekete, Alexander Kröller, Dennis Pfisterer, Stefan Fischer, and Carsten Buschmann. Neighborhood-based topology recognition in sensor networks. In *First International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS)*, volume 3121 of *Lecture Notes in Computer Science*, pages 123–136. Springer, 2004.

[Flo62] Robert W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.

[FM07] Stefan Funke and Nikola Milosavljevic. Network sketching or: "How much geometry hides in connectivity? – Part II". In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 958–967. Society for Industrial and Applied Mathematics, 2007.

[For93] Stephanie Forrest. Genetic algorithms: Principles of natural selection applied to computation. *Science*, 261(5123):872–878, 1993.

[Fun05] Stefan Funke. Topological hole detection in wireless sensor networks and its applications. In *Proceedings of the DIALM-POMC Joint Workshop on Foundations of Mobile Computing*, pages 44–53. ACM Press, 2005.

[FW04] Simon Fischer and Ingo Wegener. The Ising model on the ring: Mutation versus recombination. In *Proceedings of the Sixth Annual Conference on Genetic and Evolutionary Computation (GECCO)*, volume 3102 of *Lecture Notes in Computer Science*, pages 1113–1124. Springer, 2004.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co, 1979.

[GW03] Oliver Giel and Ingo Wegener. Evolutionary algorithms and the maximum matching problem. In *Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2607 of *Lecture Notes in Computer Science*, pages 415–426. Springer, 2003.

[HJKN08] Edda Happ, Daniel Johannsen, Christian Klein, and Frank Neumann. Rigorous analyses of fitness-proportional selection for optimizing linear functions. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO)*. ACM Press, 2008. Nominated for Best Paper Awards.

[HKL96] Pavol Hell, David Kirkpatrick, and Brenda Li. Rounding in symmetric matrices and undirected graphs. *Discrete Applied Mathematics*, 70(1):1–21, 1996.

[Hol75] John H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.

[HY04] Jun He and Xin Yao. A study of drift analysis for estimating computation time of evolutionary algorithms. *Natural Computing*, 3(1):21–35, 2004.

[Jäg08] Jens Jägersküpper. A blend of Markov-chain and drift analysis. In *Proceedings of the Tenth International Conference on Parallel Problem Solving from Nature (PPSN)*, volume 5199 of *Lecture Notes in Computer Science*, pages 41–51. Springer, 2008.

[JDJW05] Thomas Jansen, Kenneth A. De Jong, and Ingo Wegener. On the choice of the offspring population size in evolutionary algorithms. *Evolutionary Computation*, 13(4):413–440, 2005.

[Joh77] Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM*, 24(1):1–13, 1977.

[JW99] Thomas Jansen and Ingo Wegener. On the analysis of evolutionary algorithms – A proof that crossover really can help. In *Proceedings of the Seventh Annual European Symposium on Algorithms (ESA)*, volume 1643 of *Lecture Notes in Computer Science*, pages 184–193. Springer, 1999.

[JW02] Thomas Jansen and Ingo Wegener. The analysis of evolutionary algorithms – A proof that crossover really can help. *Algorithmica*, 34(1):47–66, 2002.

[JW05] Thomas Jansen and Ingo Wegener. Real royal road functions – Where crossover provably is essential. *Discrete Applied Mathematics*, 149(1-3):111–125, 2005.

[KAG90] James P. Kelly, Arjang A. Assad, and Bruce L. Golden. The controlled rounding problem: Relaxations and complexity issues. *Operations-Research-Spektrum*, 12(3):129–138, 1990.

[KFPF06] Alexander Kröller, Sándor P. Fekete, Dennis Pfisterer, and Stefan Fischer. Deterministic boundary recognition and topology extraction for large sensor networks. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm (SODA)*, pages 1000–1009. ACM Press, 2006.

[KGAB90] James P. Kelly, Bruce L. Golden, Arjang A. Assad, and Edward K. Baker. Controlled rounding of tabular data. *Operations Research*, 38(5):760–772, 1990.

[KGJV83] Scott Kirkpatrick, D. Gelatt Jr., and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[Kle04a] Christian Klein. Controlled perturbation for Voronoi diagrams. Diplomarbeit, Universität des Saarlandes, Saarbrücken, Germany, April 2004.

[Kle04b] Christian Klein. A fast root checking algorithm. Technical Report ECG-TR-363109-02, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 2004.

[KMW04] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Unit disk graph approximation. In *Proceedings of the DIALM-POMC Joint Workshop on Foundations of Mobile Computing*, pages 17–23. ACM Press, 2004.

[Knu95] Donald E. Knuth. Two-way rounding. *SIAM Journal on Discrete Mathematics*, 8(2):281–290, 1995.

[MHF93] Melanie Mitchell, John H. Holland, and Stephanie Forrest. When will a genetic algorithm outperform hill climbing? In *Proceedings of the Seventh Neural Information Processing Systems Conference (NIPS)*, volume 6 of *Advances in Neural Information Processing Systems*, pages 51–58. Morgan Kaufmann, 1993.

[MN99] Kurt Mehlhorn and Stefan Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[MRR+53] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[MW05] Thomas Moscibroda and Roger Wattenhofer. Maximal independent sets in radio networks. In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Principles of Distributed Computing*, pages 148–157. ACM Press, 2005.

[Neu04] Frank Neumann. Expected runtimes of evolutionary algorithms for the Eulerian cycle problem. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation (CEC)*, pages 904–910. IEEE Press, 2004.

[NW04] Frank Neumann and Ingo Wegener. Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. In *Proceedings of the Sixth Annual Conference on Genetic and Evolutionary Computation (GECCO)*, volume 3102 of *Lecture Notes in Computer Science*, pages 713–724. Springer, 2004.

[NW05] Frank Neumann and Ingo Wegener. Minimum spanning trees made easier via multi-objective optimization. In *Proceedings of the Seventh Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 763–769. ACM Press, 2005.

[NW10] Frank Neumann and Carsten Witt. *Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity*. Natural Computing Series. Springer, 2010.

[OHY07] Pietro S. Oliveto, Jun He, and Xin Yao. Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results. *International Journal of Automation and Computing*, 4(3):281–293, 2007.

[Rag88] Prabhakar Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *Journal of Computer and System Sciences*, 37(2):130–143, 1988.

[RGK97] Brendan J. Ring, John A. George, and Chong Juin Kuan. A fast algorithm for large-scale controlled rounding of 3-dimensional census tables. *Socio-Economic Planning Sciences*, 31(1):41–55, 1997.

[RRP+03] Ananth Rao, Sylvia Ratnasamy, Christos Papadimitriou, Scott Shenker, and Ion Stoica. Geographic routing without location information. In *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 96–108. ACM Press, 2003.

[RRS95] Yuval Rabani, Yuri Rabinovich, and Alistair Sinclair. A computational view of population genetics. In *Proceedings of the Twenty-seventh Annual ACM Symposium on Theory of Computing (STOC)*, pages 83–92. ACM Press, 1995.

[SGLY+04] Juan-José Salazar-González, Philip Lowthian, Caroline Young, Giovanni Merola, Stephen Bond, and David Brown. Getting the best results in controlled rounding with the least effort. In *Proceedings of the CASC Project Final Conference, Privacy in Statistical Databases*, volume 3050 of *Lecture Notes in Computer Science*, pages 58–72. Springer, 2004.

[Spe94] Joel Spencer. Randomization, derandomization and antirandomization: Three games. *Theoretical Computer Science*, 131(2):415–429, 1994.

[STW04] Jens Scharnow, Karsten Tinnefeld, and Ingo Wegener. The analysis of evolutionary algorithms on sorting and shortest paths problems. *Journal of Mathematical Modelling and Algorithms*, 3(4):349–366, 2004.

[Sud05] Dirk Sudholt. Crossover is provably essential for the Ising model on trees. In *Proceedings of the Seventh Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 1161–1167. ACM Press, 2005.

[SW04] Tobias Storch and Ingo Wegener. Real royal road functions for constant population size. *Theoretical Computer Science*, 320(1):123–134, 2004.

[SY93] George Steiner and Scott Yeomans. Level schedules for mixed-model, just-in-time processes. *Management Science*, 39(6):728–735, 1993.

[War62] Stephen Warshall. A theorem on Boolean matrices. *Journal of the ACM*, 9(1):11–12, 1962.

[Weg01] Ingo Wegener. Theoretical aspects of evolutionary algorithms. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 2076 of *Lecture Notes in Computer Science*, pages 64–78. Springer, 2001. invited paper.

[WGM06] Yue Wang, Jie Gao, and Joseph S. B. Mitchell. Boundary recognition in sensor networks by topological methods. In *Proceedings of the Twelfth Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 122–133. ACM Press, 2006.

[Wit05] Carsten Witt. Worst-case and average-case approximations by simple randomized search heuristics. In *Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 3404 of *Lecture Notes in Computer Science*, pages 44–56. Springer, 2005.

[Wit06] Carsten Witt. Runtime analysis of the (μ + 1) EA on simple pseudo-Boolean functions. *Evolutionary Computation*, 14(1):65–86, 2006.

[WW01] Leon Willenborg and Ton de Waal. *Elements of Statistical Disclosure Control*, volume 155. Springer, 2001.

[WW05] Ingo Wegener and Carsten Witt. On the optimization of monotone polynomials by simple randomized search heuristics. *Combinatorics, Probability and Computing*, 14(1-2):225–247, 2005.

Index