From Worst-Case to Average-Case Efficiency – Approximating Combinatorial Optimization Problems

vom der Fakultät für Informatik
der Technischen Universität Chemnitz
genehmigte Dissertation
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften (Dr. rer. nat.)
vorgelegt von
Dipl.-Inf. Kai Plociennik
geboren am 13. Juni 1975 in Recklinghausen

Tag der Einreichung: 3. August 2010
Tag der Verteidigung: 27. Januar 2011

Gutachter: Prof. Dr. Hanno Lefmann,
Technische Universität Chemnitz

Prof. Dr. Andreas Goerdt,
Technische Universität Chemnitz
Abstract

For many important combinatorial optimization problems, inapproximability results exist, stating that under reasonable complexity-theoretic assumptions such as \( P \neq NP \), no worst-case efficient algorithm exists that achieves a certain (good) approximation guarantee. This is an unfortunate situation, since in practical applications, one often has to find a (good) solution for such a problem, and resources like the available time are limited. It turns out, however, that many problems with such inapproximability results can be solved satisfactorily in practice, i.e., there are algorithms which typically find in relatively short time a relatively good solution. Hence, there is a discrepancy between worst-case results and empirical observations.

The reason for this discrepancy is that often, worst-case instances for an algorithm are somehow artificial and do not typically appear in practice. Then, the algorithm can typically perform much better than what its worst-case guarantees promise. The worst-case view is too pessimistic here to adequately assess the algorithm’s performance in practice. In average-case analysis of algorithms, one draws a random input from the set of all inputs of a given size, e.g., the set of all graphs with the same number of vertices, and then examines the expected algorithm behavior for the random input. If one can prove that the expected behavior is much better than the worst-case behavior, this can explain why the algorithm typically behaves much better than in the worst case, and why a problem is tractable in practice even though inapproximability results for it exist.

In this thesis, we consider three classical combinatorial optimization problems, namely INDEPENDENT SET, COLORING, and SHORTEST COMMON SUPERSTRING. For all three problems, inapproximability results as mentioned above exist. Hence, one knows lower bounds for the approximation guarantees which are achievable by worst-case efficient algorithms (under reasonable assumptions). We show that the worst-case view in the inapproximability results is too pessimistic: We present approximation algorithms whose approximation guarantees beat the lower bounds for the ones we can achieve with worst-case efficient algorithms, and we perform average-case analyses, showing that the expected running times of the algorithms for random inputs from certain models are polynomial. Our algorithms are not worst-case but at least average-case efficient. We also perform average-case analyses for simple greedy algorithms for our problems, which are guaranteed to run in polynomial time. We show that on the average and also with high probability, these greedy algorithms perform much better than what their worst-case guarantees promise with respect to the quality of the computed solutions. For example, for SHORTEST COMMON SUPERSTRING, we are able to prove that the greedy algorithm computes with probability exponentially close to 1 an almost optimal solution. Furthermore, we investigate the behavior of some properties of random inputs for our
problems. For example, we determine a tail bound on the optimal compression that is achievable for a set of random strings for Shortest Common Superstring. To sum up, we get algorithmic results on the typical behavior of algorithms for random inputs, and structural results on the typical properties of the random inputs themselves.

It can be argued that average-case analyses as above have a drawback: In some applications, totally random inputs do not appropriately model inputs occurring in practice. In many totally random input models, the random inputs with high probability have properties that may not be present in real-world ones, and vice versa. To provide a framework for reasonably modeling real-world inputs and analyzing the behavior of algorithms for such inputs, Daniel A. Spielman and Shang-Hua Teng introduced the smoothed analysis of algorithms. Here, a malicious adversary, who tries to make the algorithm to be analyzed perform poorly, chooses an arbitrary input for the considered problem. Then, the adversarial input is subject to a slight random perturbation. Given an adversarial input, one determines the expected algorithm behavior with respect to the random perturbation, and then one determines the worst expected behavior over all possible adversarial choices. If one succeeds in proving that even the worst expected behavior is still “good,” this can explain why in practice, the algorithm performs well: Often, real-world inputs contain a small amount of randomness, e.g., if the input comes from measurements performed on a physical system. Then, the semi-random perturbation model of random inputs in smoothed analysis reasonably models the inputs occurring in practice. The proof that even a small amount of random noise leads to a good expected behavior, regardless of the adversary’s choice, can then explain the good observed behavior of the algorithm in practice.

In this thesis, we perform probabilistic analyses in the spirit of smoothed analysis. We consider the problems Independent Set and Shortest Common Superstring for perturbation models of random inputs, and achieve similar results as in our average-case analyses.
Contents

1 Introduction ........................................... 1
   1.1 Outline ........................................ 1
       1.1.1 Smoothed Analysis and Semi-Random Inputs 5
       1.1.2 Expected Running Time and Average-Case Efficiency 10
       1.1.3 Approximation Algorithms ..................... 16
       1.1.4 Complexity Classes .......................... 17
   1.2 Results for Independent Set and Coloring ............. 18
       1.2.1 Approximating Independent Set and Coloring for Random Hypergraphs 20
       1.2.2 Approximating Independent Set in Semi-Random Graphs 27
       1.2.3 Previous and Related Work ..................... 31
   1.3 Results for Shortest Common Superstring ............... 32
       1.3.1 Random Input Models and Previous Results ......... 36
       1.3.2 Our Results ................................ 40
       1.3.3 Further Notes on Previous and Related Work ........ 44
   1.4 Summary of Our Results ........................... 45
   1.5 Bibliographical Notes ............................. 47

2 Independent Set and Coloring for Random Hypergraphs 49
   2.1 Greedy Coloring of Random Hypergraphs .......... 49
   2.2 Approximating the Independence Number .......... 53
   2.3 Approximating the Chromatic Number .............. 61
   2.4 The Expected Behavior of Greedy Independent Set and Coloring .... 64
   2.5 Small Edge Probabilities ........................ 68
   2.6 Completing the Proof of Lemma 2 ................ 73
   2.7 Conclusions and Open Problems ................... 75

3 Independent Set for Semi-Random Graphs .................. 79
   3.1 Greedy Coloring of Perturbed Graphs ............ 80
   3.2 Upper Bounding the Independence Number .......... 83
       3.2.1 The Expectation of the Largest Eigenvalue .... 84
       3.2.2 A Tail Bound on the Largest Eigenvalue ....... 90
   3.3 Approximating the Independence Number ........... 93
   3.4 The Expected Behavior of Greedy Independent Set .... 97
   3.5 Small Flip Probabilities ........................ 99
   3.6 Conclusions and Open Problems .................. 101
## Contents

### 4 Shortest Common Superstring 103
  4.1 Results in the Bernoulli and Perturbation Model 103
     4.1.1 Upper Bounding the Optimal Compression 104
     4.1.2 Computing a Shortest Superstring 111
     4.1.3 The Probabilistic PTAS 114
  4.2 Generalization to the Mixing Model 117
  4.3 Conclusions and Open Problems 123

### A Mathematical Definitions and Facts 125
  A.1 Basic Inequalities 125
  A.2 Eigenvalues 126
     A.2.1 The Independence Number and Largest Eigenvalue of Random Graphs 128
     A.2.2 The Trace of Even Powers of Symmetric Matrices 129

### Bibliography 131

### Index 138
Contents

1.1 Outline .......................................................... 1
  1.1.1 Smoothed Analysis and Semi-Random Inputs .......... 5
  1.1.2 Expected Running Time and Average-Case Efficiency . 10
  1.1.3 Approximation Algorithms ............................. 16
  1.1.4 Complexity Classes .................................. 17

1.2 Results for Independent Set and Coloring ............... 18
  1.2.1 Approximating Independent Set and Coloring for Random Hypergraphs .................................. 20
  1.2.2 Approximating Independent Set in Semi-Random Graphs ................................................. 27
  1.2.3 Previous and Related Work ......................... 31

1.3 Results for Shortest Common Superstring ............... 32
  1.3.1 Random Input Models and Previous Results ........ 36
  1.3.2 Our Results ........................................... 40
  1.3.3 Further Notes on Previous and Related Work ........ 44

1.4 Summary of Our Results .................................. 45

1.5 Bibliographical Notes ...................................... 47

1.1 Outline

In theoretical computer science, an algorithm is classically considered as being efficient if it has polynomial worst-case running time. From a practical point of view, this definition can be motivated as follows: In their everyday working-life, people run algorithms on computers to find solutions for problems they have to solve. For example, a dispatcher in a logistics company may have to find a space-efficient assignment of a number of goods to some containers, and shipping of the containers has to be planned in terms of schedules and routes. Having an efficient
algorithm in the above sense, the problem solver can be sure that in any case, the algorithm will perform well, i.e., it will find a solution in an acceptable amount of time.

Correspondingly, using this notion of efficiency, an algorithm with superpolynomial or even exponential worst-case running time is assumed to be inefficient and impracticable. However, practitioners observe that many algorithms typically perform much better than what their worst-case guarantees promise. The classical example for this observation is the simplex method for solving linear programs, introduced by Dantzig (see Dantzig [Dan63] for a survey). Even though it has exponential worst-case running time (see e.g. [KM72, Jer73, GS79]), it is widely used in practice and typically finds a solution very quickly for real-world inputs. In other words, worst-case inputs seldomly occur in practice, and hence the worst-case view is often too pessimistic for fairly evaluating an algorithm’s performance. As another example, consider Figure 1.1. The solid line depicts the running time of a fictive algorithm for inputs from the set \{0, \ldots, 255\}. Except for the three “peaks,” the running time is close to the average running time, shown by the dotted line, which is much smaller than the time used in the worst case. Thus, on the average and also for almost all inputs, our algorithm performs much better than in the worst case.

This observation led to the development of different notions of efficiency in
1.1. Outline

theoretical computer science, one of which is average-case efficiency. Roughly, average-case efficiency of an algorithm is defined as follows\(^1\): For all input sizes \(n \geq 1\), one fixes a probability distribution \(D_n\) over all inputs of size \(n\), e.g., all graphs with \(n\) vertices. Then, given \(n\), a random input of size \(n\) is drawn according to \(D_n\), and one analyzes the expected running time of a given algorithm for the random input. The considered algorithm is average-case efficient with respect to the distributions \(D_n\) if its expected running time is polynomial in \(n\).

We can interpret average-case efficiency in two ways: Firstly, one might be interested in predicting an algorithm’s performance in practice. Proving a polynomial expected running time for an algorithm can be seen as an indication that it will perform well in practice, i.e., for an “average” or “typical” input. Secondly, having an algorithm which is observed to typically perform well in practice even though it has exponential worst-case running time, one might be interested in explaining this discrepancy between theoretical results and empirical observations. In both interpretations, since we refer to real-world inputs, it is important to have a model of random inputs that reasonably models inputs occurring in practice. Given a suitable random input model, an average-case analysis as described above can then supplement a worst-case analysis and yield more detailed insight into an algorithm’s behavior.

In this dissertation, we consider three classical combinatorial optimization problems, namely \textsc{Independent Set}, \textsc{Coloring}, and \textsc{Shortest Common Superstring}. For all three problems, inapproximability results exist, stating that under reasonable complexity-theoretic assumptions, certain approximation guarantees are not achievable by polynomial worst-case running time algorithms. Thus, we cannot hope for algorithms which are always fast and always compute a good solution. However, as mentioned above, algorithms with poor worst-case performance may typically perform much better, and hence, it might be possible to solve our three problems satisfactorily in practice. The worst-case view in the inapproximability results is then again too pessimistic for adequately determining our chances for solving the problems. This actually holds for our three problems: Our main results show that \textsc{Independent Set}, \textsc{Coloring}, and \textsc{Shortest Common Superstring} are easier to approximate if the demand for efficiency is relaxed from polynomial worst-case to polynomial expected running time with respect to random inputs. More precisely, we give approximation algorithms for the problems and perform average-case analyses. We show that our algorithms have polynomial expected running time for random inputs from certain models, and achieve approximation guarantees which cannot be achieved by worst-case efficient ones, due to the known

---

\(^1\)In fact, the actual definition is somewhat more complicated. However, an algorithm which is average-case efficient in the described sense is also average-case efficient under the actual definition. For clarity, we use this slightly simplified form here. A discussion on this topic can be found in Section 1.1.2
inapproximability results.

In addition to performing classical average-case analyses, i.e., considering random inputs, we also consider models of semi-random inputs for our problems. In these models, we achieve similar results as in the totally random ones. For the sake of clarity, we speak of a probabilistic analysis instead of an average-case analysis. Our semi-random input models are motivated as follows: Above, we argued that for the results of an average-case analysis to be meaningful with respect to practical applications, one needs a model of random inputs that reasonably models real-world inputs. In Section 1.1.1, we discuss why this may not always be the case for a model with totally random inputs. To provide a framework for accurately modeling real-world inputs and analyzing the resulting algorithm behavior, Spielman and Teng [ST04] introduced the so called smoothed analysis of algorithms. The random input models in smoothed analysis are based on an adversary choosing an arbitrary input which is then perturbed by small random modifications. Thus, the input models in smoothed analysis are semi-random. We use a similar approach in our semi-random input models and the corresponding probabilistic analyses. For a detailed discussion of smoothed analysis and semi-random input models, together with a brief discussion of our corresponding results, see Section 1.1.1.

Our Results

We sketch our main results. As mentioned, for the three problems considered in this thesis, we present approximation algorithms and perform average-case analyses, proving that the algorithms have polynomial expected running time for certain models of random inputs. For Independent Set and Coloring, the average-case analyses are performed for random uniform hypergraphs, which are generated by independently inserting all edges of a certain cardinality with some probability $p$. This extends a previously known result by Krivelevich and Vu [KV02] for random graphs from the well-known $G(n, p)$ or Erdős-Rényi model. We give the results in Section 1.2 and defer details of the proofs to Chapter 2. Furthermore, we perform a probabilistic analysis for Independent Set with semi-random graphs, achieving a similar result as for the random uniform hypergraphs. In our semi-random graph model, an adversary chooses an arbitrary graph, in which we negate the existence of every edge and every non-edge independently with a small probability. The results are summarized in Section 1.2, while the details of the proofs are given in Chapter 3.

In the average-case analysis for Shortest Common Superstring, we consider the so called Bernoulli and mixing models of random strings. In the Bernoulli model, the letters in a string are chosen independently, while the mixing model can incorporate dependencies between letters that vanish with increasing distance. Furthermore, we perform a probabilistic analysis and achieve a similar result in a
1.1. Outline

semi-random input model, where an adversary specifies a set of arbitrary strings whose letters are then mutated independently with a small probability. The results for Shortest Common Superstring are presented in Section 1.3. We defer the details of the proofs to Chapter 4.

The rest of this chapter is organized as follows. We discuss smoothed analysis and semi-random input models in more detail in Section 1.1.1. Section 1.1.2 contains a discussion on average-case efficiency of algorithms and measuring the expected running time for random inputs. Then, terms regarding approximation algorithms, which are regularly used throughout this thesis, are defined in Section 1.1.3. Section 1.1.4 contains definitions of some less common complexity classes which we mention in this thesis. Sections 1.2 and 1.3 contain our main results for Independent Set and Coloring, and Shortest Common Superstring, respectively. In Section 1.4, we briefly summarize the results stated in Sections 1.2 and 1.3, pointing out the commonalities between the different results for our three problems. Finally, we give bibliographical notes in Section 1.5. For the sake of a clear presentation, we most of the time omit floor and ceiling signs in our proofs. For the same reason, we also often do not explicitly state that we assume a certain parameter (e.g., the number of vertices in a given hypergraph or the number of strings in a set) to be large enough. This is always implicitly assumed.

1.1.1 Smoothed Analysis and Semi-Random Inputs

As mentioned, to be able to perform a meaningful average-case analysis, one needs to have a model of random inputs that reasonably models real-world inputs. However, in many models, the random inputs have certain properties with high probability that may not be present in real-world ones and vice versa. In an average-case analysis, these artificial structures may then dominate the average behavior of an algorithm and limit the significance of the result for practical applications. For example, assume that we have a $G(n, p)$ random graph model. Given a number $n \in \mathbb{N}$ of vertices and an edge probability $0 < p < 1$, in this model the $\binom{n}{2}$ possible edges between $n$ labeled vertices are inserted independently of each other with probability $p$. It is easy to show that with high probability, all vertices in the random graph have approximately the same degree. In contrast, consider e.g. the Internet, where web pages link to other ones. Some pages link to only a few other pages, while some link to a very large number of other ones. Thus, in the resulting graph, with the web pages being the vertices and the edges corresponding to the links, the vertices have very different degrees. It can hence be argued that a (totally) random graph does not accurately model real-world graphs. An algorithm performing well for graphs whose vertices have similar degrees might have good average-case performance in the $G(n, p)$ model but perform poorly on the Internet graph. The random structures present with high probability here inadequately dominate the analysis.
In their Gödel- and Fulkerson-Prize winning paper [ST04], Daniel A. Spielman and Shang-Hua Teng introduced the smoothed analysis of algorithms. It is a framework for modeling real-world inputs and analyzing the behavior of algorithms for such inputs. The goal is to explain the good observed behavior of algorithms (which might have poor worst-case performance) on practical inputs. In their survey paper [ST09] on smoothed analysis, Spielman and Teng nicely describe the difference between random and real-world instances as follows:

“For example, one can see what a random image looks like by disconnecting most TV sets from their antennas, at which point they display ‘static.’ These random images do not resemble actual television shows.”

The idea of the random input models in smoothed analysis can be described as follows: Given an algorithm for a certain problem, a malicious adversary, who tries to make the algorithm perform badly, chooses an arbitrary input \( x \) for the problem. Then, \( x \) is perturbed by small random modifications. In other words, a small amount of “random noise” is added to \( x \). For a fixed adversarial choice \( x \), one analyzes the expected behavior of the algorithm with respect to the random perturbation of \( x \). Then, one determines the worst expected behavior over all possible adversarial inputs. The goal is to show that regardless of the adversary’s choice, the algorithm performs well in expectation. Here, “how well” the algorithm behaves exactly depends on the magnitude of the random perturbation. Assume that the adversary has chosen a worst-case instance. For very small amounts of random noise, the perturbed instance will very likely be “close to worst-case,” and the algorithm will perform not so well in expectation; but with increasing magnitude of the random perturbation, the perturbation will “smooth out” the special properties of the worst-case instance, and we can expect that the algorithm performs better and better. We have a trade-off between algorithm performance and the amount of random noise.

Notice that smoothed analysis is a mixture between worst-case and average-case analysis. For different amounts of random noise, it interpolates between the two. Since its random inputs are neither completely random nor completely arbitrary, neither random nor arbitrary structures can dominate the analysis as was discussed above for totally random inputs.

Formally, the smoothed complexity, which for consistence with our notation we call the smoothed running time here, is defined as follows. Let \( I_n \) be the set of all inputs with size \( n \) of a given problem. Let us measure the magnitude of perturbation by \( \varepsilon > 0 \), and denote by \( x \sim R(x, \varepsilon) \) the process that, given an adversarial input \( x \in I_n \) and \( \varepsilon > 0 \), a random input \( x \) is obtained by randomly perturbing \( x \) with magnitude \( \varepsilon \). For example, \( I_n \) could be \([0, 1]^n\), and \( x \sim R(x, \varepsilon) \) could be the process
of randomly flipping every bit in \( x \) independently with probability \( \varepsilon \). If we write \( E_{x \sim R(x,\varepsilon)}[\cdot] \), we mean an expectation with respect to a random input \( x \sim R(x,\varepsilon) \). Now, let \( t_A(x) \) denote the running time of algorithm \( A \) on input \( x \). The smoothed running time of \( A \) for input size \( n \) and perturbation magnitude \( \varepsilon > 0 \) is defined as

\[
t_A^{sm}(n, \varepsilon) = \max_{x \in I_n} E_{x \sim R(x,\varepsilon)}[t_A(x)] .
\]  

Using this, the goal in smoothed analysis is to show that an algorithm has polynomial smoothed complexity or polynomial smoothed running time as we call it here. An algorithm \( A \) has polynomial smoothed running time if there are constants \( n_0, c, k_1, k_2 \in \mathbb{N} \) and a real constant \( \varepsilon_0 > 0 \) such that for all \( n \geq n_0 \) and \( 0 < \varepsilon \leq \varepsilon_0 \),

\[
t_A^{sm}(n, \varepsilon) \leq c \cdot n^{k_1} \cdot \varepsilon^{-k_2} .
\]

Hence, the worst expected running time given \( n, \varepsilon \) must be polynomial in \( n \) and \( 1/\varepsilon \). Here, we can see the above-mentioned (polynomial) trade-off between the amount \( \varepsilon \) of random noise and expected algorithm performance. For a polynomially small magnitude \( \varepsilon = 1/n^d, d \in \mathbb{N} \) fixed, of perturbation, an algorithm with polynomial smoothed running time has polynomial expected running time regardless of the adversary’s choice. This can explain the good observed behavior in practice.

As an example, consider again Figure 1.1. The dashed line shows the “perturbed running time” of the algorithm underlying the figure, i.e., given an adversarial input \( x \), the expected running time with respect to the random perturbation of \( x \). The perturbation model used is as follows: Given an adversarial input \( x \in \{0, \ldots, 255\} \), we draw a random input uniformly from the set \( \{x \in \{0, \ldots, 255\} : |x - x| \leq 5\} \), i.e., from the neighborhood of \( x \) with radius 5. The smoothed running time, the maximum of the perturbed running times, is shown by the dash-dotted line. Since it is considerably smaller than the worst-case running time, the algorithm performs considerably better in the smoothed model than in the worst case.

Using smoothed analysis, Spielman and Teng were the first to convincingly explain the above-mentioned discrepancy between the fact that the simplex algorithm for linear programs is successfully applied to practical instances while it has exponential worst-case running time: In [ST04], they showed that a certain version of the simplex algorithm has polynomial smoothed complexity. Roughly, the perturbation model is that the adversary can choose real coefficients for the linear program’s objective function and constraints, and then a Gaussian random variable with mean 0 and standard deviation \( \sigma \) is independently added to each coefficient in the constraints. Since \( \sigma \) measures the magnitude of perturbation, it is the equivalent of our \( \varepsilon \) (cf. (1.2) for \( \varepsilon = \sigma \)). Earlier, average-case analyses for the simplex method had been performed (see e.g. Borgwardt [Bor82] and Smale [Sma83]).
showing that for certain models of random inputs, the simplex method has polynomial expected running time. However, these analyses did not explain as convincingly as the analysis of Spielman and Teng in [ST04] why the simplex method is fast in practice, because the totally random input models did not have as much resemblance with real-world inputs as the model in smoothed analysis has.

The motivation for using a semi-random (perturbation) input model in smoothed analysis is that real-world inputs often are actually neither completely random nor completely arbitrary. For example, if the input originates from measurements performed on a physical system, the measurements introduce small random errors, perturbing “the real” input slightly. As another example, consider DNA sequencing as an application. Since the DNA string one wants to determine is evolved by random mutations from some ancestor’s DNA, the input can be seen as an arbitrary string (the ancestor’s DNA) with random modifications. To conclude, a perturbation model is often arguably a reasonable model for real-world inputs and might be more appropriate than a model of totally random inputs. Furthermore, while a classical average-case analysis may yield that an algorithm “performs well for almost all inputs,” a smoothed analysis may yield that it “performs well for almost all inputs in the neighborhood of every input,” which might be seen as a stronger indication that the algorithm will perform well in practice.

**Our Results – Probabilistic Analyses** In this thesis, we perform probabilistic analyses “in the spirit of smoothed analysis” for **Independent Set** and **Shortest Common Superstring**. As mentioned above, we achieve similar results using semi-random (perturbation) input models as for random inputs.

For **Independent Set**, our semi-random input model is the following: An adversary first chooses a graph $G$. Then, for a small flip probability $\varepsilon > 0$, each edge of $G$ is removed and each non-edge of $G$ is inserted with probability $\varepsilon$. In other words, we negate the existence of each potential edge with probability $\varepsilon$. The decisions for the potential edges are made independently of each other. Our model was proposed by Spielman and Teng [ST09] in their survey paper on smoothed analysis, in which it is described as a “smoothed extension” of the $G(n, p)$ model. For details on the input model and the results see Section 1.2.2. Proof details are given in Chapter 3.

For **Shortest Common Superstring**, we present a generic model of semi-random strings which e.g. can model the following process: An adversary chooses an arbitrary set of strings. Then, for a small mutation probability $p_m > 0$, each letter is mutated independently with probability $p_m$. In case of a mutation, a different letter is chosen according to some probability distribution. For details of the perturbation model and the corresponding results see Section 1.3.1. Proofs are given in Chapter 4.

Above, we said that our analyses are “in the spirit” of smoothed analysis. The reason is the following: Consider **Independent Set**. Recall that we present an al-
1.1. Outline

Algorithm $A$ which beats the lower bounds for the approximation guarantees that are achievable by worst-case efficient algorithms, due to known inapproximability results. Thus, even the fastest algorithm $A^*$ which achieves an approximation guarantee that is as good as ours has superpolynomial worst-case running time (assuming $P \neq NP$, see Section 1.2). We show that therefore, we cannot expect to find an algorithm that has our approximation guarantee and has polynomial smoothed running time.

Let $\hat{G}_n$ be the set of all graphs with $n$ vertices. By our convention, for an adversarial graph $G \in \hat{G}_n$ and a flip probability $\varepsilon > 0$, by $G \sim R(G, \varepsilon)$ we denote a random graph $\mathcal{G}$ produced by our semi-random model. According to (1.1) and (1.2), to prove a polynomial smoothed running time for an algorithm $A^*$, we have to show that for sufficiently large $n$ and sufficiently small flip probability $\varepsilon > 0$:

$$\max_{G \in \hat{G}_n} \mathbb{E}_{G \sim R(G, \varepsilon)}[t_{A^*}(G)] \leq p(n, \varepsilon^{-1})$$  \hfill (1.3)

such that $p(n, \varepsilon^{-1})$ is a fixed bivariate polynomial in $n$ and $1/\varepsilon$.

Now, assume that $\varepsilon = 1/n^3$. Then, the probability that in a graph on $n$ vertices, at least one edge is flipped is at most $\left(\frac{n}{2}\right) \cdot \frac{1}{n^3} \leq n^2/n^3 = 1/n = o(1)$. Hence, for $G \in \hat{G}_n$, with probability $1-o(1)$, a random graph $\mathcal{G} \sim R(G, \varepsilon)$ is exactly $\mathcal{G} = G$. Let $G^\text{wc}_n$ be a graph from $\hat{G}_n$ that maximizes the running time of $A^*$ over $\hat{G}_n$, i.e., $G^\text{wc}_n$ is a worst-case instance (which the adversary can choose). For our choice $\varepsilon = 1/n^3$,

$$\mathbb{E}_{G \sim R(G^\text{wc}_n, \varepsilon)}[t_{A^*}(\mathcal{G})] \geq \mathbb{E}_{G \sim R(G^\text{wc}_n, \varepsilon)}[t_{A^*}(\mathcal{G}) | \mathcal{G} = G^\text{wc}_n] \cdot \Pr_{G \sim R(G^\text{wc}_n, \varepsilon)}[\mathcal{G} = G^\text{wc}_n] = t_{A^*}(G^\text{wc}_n) \cdot (1-o(1)) = \Omega(t_{A^*}(G^\text{wc}_n)),$$

which is superpolynomial for every algorithm $A^*$ that achieves an approximation guarantee as good as ours (assuming $P \neq NP$). But (1.3) for $\varepsilon = 1/n^3$ demands that

$$\mathbb{E}_{G \sim R(G^\text{wc}_n, \varepsilon)}[t_{A^*}(\mathcal{G})] \leq p(n, \varepsilon^{-1}) = p(n, n^3) = \text{poly}(n),$$

a contradiction to $A^*$’s superpolynomial worst-case running time. We conclude that we cannot expect to find an algorithm for INDEPENDENT SET achieving our approximation guarantee that has polynomial smoothed running time. For SHORTEST COMMON SUPERSTRING, analog arguments hold.

Instead, in our results, we do the following: We use the same semi-random perturbation model as in smoothed analysis, and show that if the magnitude of perturbation is sufficienly high, our algorithms have polynomial expected running time.
1.1.2 Expected Running Time and Average-Case Efficiency

As usual, the term *efficient* algorithm throughout this thesis always refers to one with polynomial worst-case running time. To be able to analyze the expected running times of algorithms, we define some technical terms. All algorithms in this thesis for which a polynomial expected running time is claimed consist of a sequence of steps, of which some are executed for a given input. If a step in such an algorithm outputs a solution, the algorithm terminates (this also holds for all other algorithms). Some steps in our algorithms try to certify that a computed solution’s approximation ratio is sufficiently good to achieve the desired approximation guarantee. For example, some steps test whether the value of the solution is bounded by a certain threshold. If the step does not succeed in its attempt, it *fails*. Finally, let the *effort* of a step be the time spent if it is executed. Then, the expected running time of a step is the product of its effort and its execution probability. Furthermore, by linearity of expectation, the expected running time of an algorithm is the sum of the expected running times of its steps.

**Average-Case Efficiency** In the introduction, for the sake of clarity we gave a slightly simplified definition of average-case efficiency, i.e., we said that an algorithm is average-case efficient if its expected running time for a random input is polynomial in the input size. This is the definition we use throughout this thesis. The reason is that it is simple and, more importantly, it implies average-case efficiency as defined in the theory of *average-case complexity*, which is an average-case version of classical worst-case complexity theory.

In complexity theory, one tries to separate the problems that are tractable from the ones that are intractable. The separation between tractable and intractable problems is based on the existence of a “practicable” algorithm. One considers a problem to be tractable if such an algorithm exists, and intractable otherwise. The exact definition of what a practicable algorithm is depends on the type of result one wants to achieve. Consider e.g. NP-completeness theory, which is a classical worst-case complexity theory. Here, an algorithm is (implicitly) seen as being practicable if it has polynomial worst-case running time. Consequently, problems in P (the ones for which such an algorithm exists) are considered tractable, while NP-hard problems (for which it is unlikely that such an algorithm exists) are considered intractable.

In contrast to this worst-case view, average-case complexity theory tries to separate the “typically” or “average-case” tractable problems from the ones that are intractable even in this weaker sense of tractability. Analogously to above, the distinction between typically tractable and intractable problems is based on the existence of a “typically practicable” algorithm. In consequence, one needs a definition of a typically practicable or average-case efficient algorithm that is reasonable in this context. The definition one uses in average-case complexity theory is that an
algorithm is average-case efficient if it has so called \textit{average polynomial running time}. In this section, we discuss the exact definition and its motivation.

It would be tempting to define average-case efficiency of an algorithm in our simple way of requiring polynomial expected running time for random inputs, which seems to be the canonical average-case version of worst-case efficiency. However, surprisingly subtle difficulties arise when trying to formulate the “right” notion of average-case efficiency in the context of average-case tractability. As mentioned, we need a definition of an average-case efficient algorithm such that considering a problem to be typically tractable if and only if such an algorithm exists is reasonable. It turns out that our simple definition of polynomial expected running time is \textit{not} a reasonable definition in this context. We explain this next.

To make the different definitions of average-case efficiency clear and precise, we restate and define some terms. In both our definition and the one used in average-case complexity theory, given a problem \( \Pi \), for every input size \( n \in \mathbb{N} \), one fixes a probability distribution \( D_n \) over all inputs of size \( n \). For example, \( D_n \) may be a distribution over all graphs with \( n \) vertices. Let \( \mathcal{D} = \{ D_n \mid n \in \mathbb{N} \} \) be the set of all distributions \( D_n \). We call \( \mathcal{D} \) a \textit{distribution over all inputs} of \( \Pi \). Let \( x \sim D_n \) denote the process that the input \( x \) is randomly drawn according to \( D_n \). If we write \( \mathbb{E}_{x \sim D_n}[\cdot] \) or \( \Pr_{x \sim D_n}[\cdot] \), we then mean an expectation or probability with respect to a random \( x \) from \( D_n \). Finally, for a (deterministic) algorithm \( A \) and an input \( x \), let \( t_A(x) \) denote the running time of \( A \) on input \( x \). Our definition of average-case efficiency is as follows.

\textbf{Definition 1} (a.c. efficiency, thesis version). Fix a distribution \( \mathcal{D} = \{ D_n \mid n \in \mathbb{N} \} \) over all inputs of a problem. An algorithm \( A \) is \textit{average-case efficient} with respect to \( \mathcal{D} \) if

\[ \mathbb{E}_{x \sim D_n}[t_A(x)] = \text{poly}(n) \, . \]

In their survey paper on average-case complexity, Bogdanov and Trevisan [BT06] explain why Definition 1 is not reasonable to determine whether a problem is average-case tractable or not, using roughly the following example: Assume that \( D_n \) is the uniform probability distribution over all \( \{0,1\} \)-strings of length \( n \). Furthermore, assume that we have an algorithm \( A \) which on a certain model of computation (say, a random access machine with logarithmic cost) runs in time \( \Theta(n) \) on all strings except the string \( 0^n \) and in time \( \Theta(2^n) \) on the string \( 0^n \). Then, the expected running time of \( A \) on this machine model is

\[
\mathbb{E}_{x \sim D_n}[t_A(x)] = \sum_{y \in \{0,1\}^n} t_A(y) \cdot \Pr_{x \sim D_n}[x = y] \\
= \Theta(n) \cdot \frac{2^n - 1}{2^n} + \Theta(2^n) \cdot \frac{1}{2^n} \\
= \Theta(n) + \Theta(1) = \Theta(n) \, . \]
Hence, $A$ has polynomial expected running time and is therefore average-case efficient under Definition 1, which matches our intuition that an algorithm with linear expected running time is typically efficient.

Now, assume that we execute $A$ on another model of computation which is quadratically slower, i.e., the running time $t'_A(x)$ on input $x$ in the new machine model is $t'_A(x) = t_A(x)^2$. Since $A$ runs only quadratically slower in the new model, we would definitely say that it is still typically efficient. In other words, we clearly want a notion of average-case efficiency that is robust against changes of the machine model which introduce polynomial speed-ups or slow-downs. But observe that

$$E_{x \sim D_n}[t'_A(x)] = \sum_{y \in \{0,1\}^n} t'_A(y) \cdot \Pr_{x \sim D_n}[x = y]$$

$$= \sum_{y \in \{0,1\}^n} t_A(y)^2 \cdot \Pr_{x \sim D_n}[x = y]$$

$$= \Theta(n^2) \cdot \frac{2^n - 1}{2^n} + \Theta(2^{2n}) \cdot \frac{1}{2^n}$$

$$= \Theta(n^2) + \Theta(2^n) = \Theta(2^n),$$

which is exponential. Thus, on the second machine model, our algorithm is not average-case efficient under Definition 1, even though it has this property in the first model and even though we intuitively consider it to be typically efficient on the slower model. (After all, the probability that it runs in quadratic time is exponentially close to one!) To conclude, Definition 1 is not robust against polynomial speed-ups or slow-downs in the machine model, and hence it is not suitable to define whether a problem is average-case tractable or not.

Instead of requiring polynomial expected running time, one defines average-case efficiency in the following way: One allows an algorithm to spend large amounts of time on some inputs (even exponential ones), but the probability that it spends a certain (large) amount of time has to be “reasonably small.” In other words, one demands a (polynomial) trade-off between the amount of time spent on an input and its probability, i.e., increasingly large amounts of spent time have to have smaller and smaller probability.

Next, we mention two existing, equivalent definitions of average-case efficiency as used in average-case complexity theory (Definitions 2 and 3), and discuss their relations to each other and their meaning. The two definitions, together with further discussions and motivations, are presented in the survey paper [BT06] on average-case complexity by Bogdanov and Trevisan. The following explanations and arguments can partly be found in similar form in [BT06].

In the used definitions of average-case efficiency, for technical reasons, an algorithm $A$ gets besides the random input $x$ from $D_n$ also the value $n$ as input. The running time of $A$ on inputs $x, n$ is denoted $t_A(x; n)$. 
1.1. Outline

Definition 2 (a.c. efficiency, trade-off version). Fix a distribution $D = \{D_n | n \in \mathbb{N}\}$ over all inputs of a problem. An algorithm $A$ has average polynomial running time with respect to $D$ if there is an $\varepsilon > 0$ and a polynomial $p(n)$ such that for all $n, t \in \mathbb{N}$,

$$\Pr_{x \sim D_n}[t_A(x; n) \geq t] \leq \frac{p(n)}{t^\varepsilon}.$$  \hspace{1cm} (1.4)

To see that Definition 2 reasonably captures our intuitive notion of a typically efficient algorithm, observe e.g. that an algorithm having average polynomial running time under this definition runs in polynomial time with high probability: Assume that an algorithm $A$ has average polynomial running time, and fix a polynomial $p'(n)$ and a constant $\varepsilon' > 0$ such that (1.4) is fulfilled using these values for $p(n)$ and $\varepsilon$, respectively. Let $p''(n) = (n \cdot p'(n)^{1/\varepsilon'})$, which is obviously a polynomial. Then, the probability that $A$ spends time at least $p''(n)$ for a random input $x \sim D_n$ is due to (1.4)

$$\Pr_{x \sim D_n}[t_A(x; n) \geq p''(n)] \leq \frac{p'(n)}{p''(n)^{\varepsilon'}} = \frac{p'(n)}{((n \cdot p'(n)^{1/\varepsilon'})^{\varepsilon'}} = \frac{1}{n}.$$

Therefore, $A$ runs in polynomial time with probability at least $1 - 1/n$, and it is justified to say that it is typically efficient.

Above, we mentioned that our Definition 1 is not robust against polynomial changes in the speed of the machine model. It is easy to see that in contrast, Definition 2 indeed is robust against such changes: Assume that $A$ has average polynomial running time in some model $M$. Clearly, this also holds in any other model $M'$ which is faster than $M$. Now, assume that we have a model $M'$ in which the running time on input $x \sim D_n$ is $t'_A(x; n) = c \cdot t_A(x; n)^k$ for fixed $c > 0$ and fixed $k \geq 2$, i.e., $M'$ is polynomially slower than $M$. Fix a polynomial $p'(n)$ and a constant $\varepsilon' > 0$ such that (1.4) is fulfilled for $A$ in model $M$ for $p(n) = p'(n)$ and $\varepsilon = \varepsilon'$. Then, in the slower model $M'$, due to (1.4) we get

$$\Pr_{x \sim D_n}[t'_A(x; n) \geq t] = \Pr_{x \sim D_n}[c \cdot t_A(x; n)^k \geq t] = \Pr_{x \sim D_n}[t_A(x; n) \geq (t/c)^{1/k}] \leq \frac{p'(n)}{((t/c)^{1/k})^{\varepsilon'}} = \frac{c^{\varepsilon'/k} \cdot p'(n)}{t^{\varepsilon'/k}} = \frac{p''(n)}{t^{\varepsilon'}}.$$
for the polynomial \( p''(n) = c\varepsilon' / k \cdot p'(n) \) and the constant \( \varepsilon'' = \varepsilon' / k > 0 \). Thus, for \( p(n) = p''(n) \) and \( \varepsilon = \varepsilon'' \), (1.4) is fulfilled for \( A \) in the slower model \( M' \). Hence, \( A \) has average polynomial running time also in \( M' \). We conclude that Definition 2 is robust against polynomial changes in the speed of the machine model.

An alternative definition of average-case efficiency, which is equivalent to Definition 2, is the following.

**Definition 3** (a.c. efficiency, expected time version). Fix a distribution \( D = \{D_n \mid n \in \mathbb{N}\} \) over all inputs of a problem. An algorithm \( A \) has average polynomial running time with respect to \( D \) if there is an \( \varepsilon > 0 \) such that

\[
E_{x \sim D_n}[t_A(x; n)^{\varepsilon''}] = O(n) .
\] (1.5)

We show that Definitions 2 and 3 are equivalent: First, suppose that an algorithm \( A \) fulfills Definition 2, and fix a polynomial \( p'(n) \) and a constant \( \varepsilon' > 0 \) such that (1.4) is fulfilled using these values for \( p(n) \) and \( \varepsilon \), respectively. Now, fix \( c > 0 \) and \( k \geq 1 \) such that \( p'(n) \leq c \cdot n^k \) for all \( n \in \mathbb{N} \). Using this, let \( \varepsilon'' = \varepsilon' / (k + 2) \). We show that (1.5) is fulfilled using \( \varepsilon = \varepsilon'' \). Therefore, \( A \) fulfills Definition 3. To see that (1.5) is fulfilled, observe that

\[
E_{x \sim D_n}[t_A(x; n)^{\varepsilon''}] = \sum_{t=1}^{\infty} P_{x \sim D_n}[t_A(x; n)^{\varepsilon''} \geq t]
\]

\[
= \sum_{t=1}^{\infty} P_{x \sim D_n}[t_A(x; n) \geq t^{1/\varepsilon''}]
\]

\[
\leq n + \sum_{t=n}^{\infty} \frac{c \cdot n^k}{(t^{1/\varepsilon''})^{\varepsilon'}}
\]

\[
= n + \sum_{t=n}^{\infty} \frac{c \cdot n^k}{t^{k+2}} .
\] (1.6)

To derive (1.6), we used (1.4) for \( p(n) = p'(n) \leq c \cdot n^k \) and \( \varepsilon = \varepsilon' \), while (1.7) follows by choice of \( \varepsilon'' \). Since for \( t \geq n \), we have \( t^{k+2} \geq n^k \cdot t^2 \), we can conclude with (1.7) that

\[
E_{x \sim D_n}[t_A(x; n)^{\varepsilon''}] \leq n + \sum_{t=n}^{\infty} \frac{c}{t^2}
\]

\[
= n + O(1) = O(n) .
\]

To conclude, \( A \) fulfills (1.5) with \( \varepsilon = \varepsilon'' \). This completes our proof that Definition 3 is fulfilled if this holds for Definition 2.
Now, assume that Definition 3 is fulfilled, and fix $\varepsilon' > 0$ such that for $\varepsilon = \varepsilon'$, (1.5) holds. To see that Definition 2 is fulfilled, observe that
\[
\Pr_{x \sim D_n}[t_A(x; n) \geq t] = \Pr_{x \sim D_n}[t_A(x; n)^{\varepsilon} \geq t^{\varepsilon'}] \\
\leq \frac{\mathbb{E}_{x \sim D_n}[t_A(x; n)^{\varepsilon}]}{t^{\varepsilon'}} \\n= \frac{O(n)}{t^{\varepsilon'}}.
\]
(1.8)

To get (1.8), we used Markov's inequality, which states that for every random variable $X$ that takes on only nonnegative values and every $\alpha > 0$, \[
\Pr[X \geq \alpha] \leq \frac{\mathbb{E}[X]}{\alpha}.
\]
For (1.9), we used that (1.5) holds for $\varepsilon = \varepsilon'$. We conclude that for suitable $p(n)$ and $\varepsilon > 0$, we can fulfill (1.4) and therefore, Definition 2 is fulfilled. This completes our proof that Definitions 2 and 3 are equivalent.

Using the above arguments, we can now easily show that in (1.5), we can replace the term $O(n)$ by $\text{poly}(n)$ without changing the definition: Clearly, if for some $\varepsilon' > 0$, we have $\mathbb{E}_{x \sim D_n}[t_A(x; n)^{\varepsilon'}] = O(n)$, then also $\mathbb{E}_{x \sim D_n}[t_A(x; n)^{\varepsilon'}] = \text{poly}(n)$. For the other direction, assume that $\mathbb{E}_{x \sim D_n}[t_A(x; n)^{\varepsilon'}] \leq p'(n)$ for a certain polynomial $p'(n)$ and fixed $\varepsilon' > 0$. Then, (1.9) yields $\Pr_{x \sim D_n}[t_A(x; n) \geq t] \leq \frac{p'(n)}{t^{\varepsilon'}}$. Therefore, Definition 2 is fulfilled, which we have already shown to imply Definition 3 (in the original form with $O(n)$ in (1.5) instead of $\text{poly}(n)$). Therefore, $\mathbb{E}_{x \sim D_n}[t_A(x; n)^{\varepsilon'}] = O(n)$ for suitable $\varepsilon > 0$.

At the beginning of this section, we claimed that our definition of average-case efficiency as having polynomial expected running time (Definition 1) is more restrictive than Definitions 2 and 3, i.e., our definition implies the latter two. This can be easily seen: Assume that we have an algorithm $A$ fulfilling Definition 1, i.e., $\mathbb{E}_{x \sim D_n}[t_A(x)] \leq p(n)$ for a certain polynomial $p(n)$. For $\varepsilon = 1$ and $O(n)$ replaced by $p(n)$, Definition 3 is fulfilled. (We just have proved that we can replace $O(n)$ by an arbitrary polynomial in Definition 3.) We conclude that fulfilling Definition 1 implies Definitions 2 and 3. In other words, all of our results in which we present an algorithm with polynomial expected running time for some problem, show that the problem is solvable in average polynomial running time, i.e., is average-case tractable in the sense used in the literature on average-case complexity.

For the machine model we assume in our results, assume e.g. that we have a random access machine with logarithmic cost. However, since our results only show that the expected running time is upper bounded by some polynomial, and do not give a concrete polynomial that upper bounds the expected running time, the precise model is not important as long as it is only polynomially slower than a random access machine with logarithmic cost. It is easy to see that our proofs work
for any machine model whose speed is polynomially related to a random access machine with logarithmic cost: We only argue that we can efficiently execute some elementary (arithmetical) operations.

1.1.3 Approximation Algorithms

In this section, we define some notions related to approximation algorithms which we use regularly throughout this thesis. For an excellent introduction to the field of approximation algorithms, we refer the reader to Vazirani [Vaz01].

For an optimization problem \( \Pi \), let \( I_\Pi \) be the set of all inputs. For an input \( x \in I_\Pi \), let \( S_\Pi(x) \) contain the possible solutions for \( x \). For the problems considered in this thesis, \( S_\Pi(x) \) is always finite (neglecting unreasonable solutions where necessary). Therefore, the following minimum and maximum are well-defined: If we have a minimization problem, we denote the optimum (the minimal value of a solution) for a given input \( x \) by \( \text{opt}_\Pi(x) = \min_{S \in S_\Pi(x)} f_\Pi(S) \). For a maximization problem, we denote the optimum by \( \text{opt}_\Pi(x) = \max_{S \in S_\Pi(x)} f_\Pi(S) \).

In the context of approximation algorithms, we measure the quality of a solution \( S \in S_\Pi(x) \) as the factor by which its value deviates from the optimum. Let the approximation ratio of \( S \) be

\[
\text{ar}(S) := \max \left\{ \frac{f_\Pi(S)}{\text{opt}_\Pi(x)}, \frac{\text{opt}_\Pi(x)}{f_\Pi(S)} \right\}.
\]

Notice that by this definition, for both minimization and maximization problems, optimal solutions have approximation ratio 1, while worse solutions correspond to larger ratios. Hence, we are interested in algorithms that find solutions with as small as possible (close to 1) approximation ratios. For a function \( g: \mathbb{N} \to [1, \infty) \), we say that an approximation algorithm for an optimization problem has approximation guarantee (or factor) \( g(n) \) if it guarantees the following: For all input sizes \( n \in \mathbb{N} \) and every input of size \( n \), it computes a solution with approximation ratio at most \( g(n) \).

In Chapter 4, we present an approximation scheme for Shortest Common Superstring with polynomial expected running time for random inputs from certain models. An approximation scheme is an algorithm that gets an input \( x \in I_\Pi \) of a problem \( \Pi \) together with a rational quality parameter \( \varepsilon > 0 \) as its input. For every input \( x \) and every \( \varepsilon > 0 \), it guarantees to compute a solution \( S \in S_\Pi(x) \) with approximation ratio at most \( 1 + \varepsilon \). As usual, we call an approximation scheme which for every fixed \( \varepsilon > 0 \) has polynomial worst-case running time a polynomial time approximation scheme (PTAS). Due to the average-case efficiency of the approximation scheme we present for Shortest Common Superstring, we might say that we have a probabilistic PTAS for the problem.
1.1.4 Complexity Classes

Below, we mention some known inapproximability results for our three problems. These results hold under certain (reasonable) complexity-theoretic assumptions such as $P \neq NP$. Since some of the considered complexity classes are less common, we give their definitions in this section for clarity.

- The class $P$ (*deterministic polynomial-time*) as usual denotes the class of decision problems $L$ for which a deterministic polynomial-time Turing machine exists that decides $L$.

- The class $NP$ (*nondeterministic polynomial-time*) as usual denotes the class of decision problems $L$ for which a nondeterministic polynomial-time Turing machine exists that accepts $L$. (This means, for inputs $x \in L$, at least one computation path accepts, while for inputs $x \notin L$, no computation path accepts.)

- The class $ZPP$ (*zero-error probabilistic polynomial-time*) is the class of decision problems $L$ for which a probabilistic polynomial-time Turing machine $M$ exists which has the following properties: $M$ always outputs yes, no, or ?, where the latter means “don’t know.” For inputs $x \in L$, it outputs yes with probability greater than $1/2$ and never outputs no. If $x \notin L$, it outputs no with probability greater than $1/2$ and never outputs yes. In other words, $M$ never makes a mistake but is allowed to answer “don’t know” with probability less than $1/2$.

- The class $\text{BPTIME}(f(n))$ (*bounded-error probabilistic $f(n)$-time*) is the class of decision problems $L$ for which a probabilistic Turing machine $M$ with the following properties exists: For input size $n$, the running time of $M$ is upper bounded by $f(n)$. It always outputs yes or no. For inputs $x \in L$, it outputs yes with probability greater than $2/3$, for inputs $x \notin L$, it outputs no with probability greater than $2/3$. In other words, $M$ is allowed to make mistakes, but this must happen with probability less than $1/3$. For a class $F$ of functions $f(n)$, we let $\text{BPTIME}(F) = \bigcup_{f \in F} \text{BPTIME}(f(n))$.

- The class $\text{maxSNP}$ (*maximization strict NP*) is a class of optimization problems. The exact definition of $\text{maxSNP}$ is far too complex to reasonably present it here in a brief summary. Roughly, it is defined as follows: $\text{maxSNP}$ is the class of all optimization problems that are L-reducible to a problem in $\text{maxSNP}_0$. The latter class is a maximization version of the class $\text{SNP}$ (*strict NP*) of decision problems which are expressible in so called *existential second-order logic* using only universal quantifiers. Finally, a problem is $\text{maxSNP}$-hard if all problems in $\text{maxSNP}$ are L-reducible to it.
An important known result is that maxSNP-hard problems admit no polynomial time approximation scheme, unless $P = NP$. Hence, assuming $P \neq NP$, for a maxSNP-hard problem, there is a constant $f > 1$ such that no polynomial worst-case running time approximation algorithm with approximation guarantee $f$ exists. (We mention such a result for Shortest Common Superstring in Section 1.3).

For detailed definitions and further discussions, we refer the reader to Papadimitriou [Pap94].

### 1.2 Results for Independent Set and Coloring

We start by defining the problems **Independent Set** and **Coloring**. A *hypergraph* $H = (V, E)$ consists of a finite set $V$ of *vertices* and a set $E$ of *edges*, which are subsets of $V$. If all edges have the same cardinality $d$, we say that $H$ is $d$-uniform. Finally, a *graph* is a 2-uniform hypergraph. This defines graphs as usual, without self-loops.

Given a hypergraph $H = (V, E)$, a subset $V' \subseteq V$ of its vertices is called *independent* if it spans no edges, i.e., for no edge $e \in E$, all of $e$’s vertices are contained in $V'$. For a graph $G = (V, E)$, this simplifies to requiring that for no edge $e = \{v, w\} \in E$, both $v$ and $w$ are contained in $V'$.

For a hypergraph $H$, the size of a largest independent set in $H$ is its *independence number* $\alpha(H)$. We can now define the problem **Independent Set**.

**Problem Independent Set**

**Input:** A hypergraph $H = (V, E)$

**Task:** Find an independent set $V' \subseteq V$ of $H$ with maximum cardinality.

Next, we define the problem **Coloring**. Given a hypergraph $H = (V, E)$, a *coloring* with $k$ colors of $H$’s vertices is a partition $C = \{C_1, \ldots, C_k\}$ of the vertex set $V$ into $k$ pairwise disjoint *color classes*. A coloring is *feasible* if all color classes are independent sets. Given $H$, the smallest $k \in \mathbb{N}$ for which a feasible coloring with $k$ colors exists is $H$’s *chromatic number* $\chi(H)$. We define the problem **Coloring** as follows.

**Problem Coloring**

**Input:** A hypergraph $H = (V, E)$

**Task:** Find a feasible coloring of $H$’s vertices with as few as possible colors.

The computational complexities of **Independent Set** and **Coloring** are well studied. It is well known (see Karp [Kar72]) that both problems are $NP$-hard.
for graphs as inputs. Since it is therefore unlikely that efficient algorithms which always compute an optimal solution exist for the problems, approximation algorithms have been studied extensively. By our definition of approximation ratios in Section 1.1.3 as the factor by which we deviate from the optimum, an independent set $I$ in a hypergraph $H$ has approximation ratio $\alpha(H)/|I|$, and a coloring with $k$ colors has approximation ratio $k/\chi(H)$. Throughout the rest of this section and Chapters 2 and 3, $n$ always denotes the number of vertices of a hypergraph under investigation.

We mention some known results about efficient approximation of Independent Set and Coloring, and start with positive results, i.e., efficient approximation algorithms. To the author’s knowledge, for both problems with graphs as inputs, the best known efficient approximation algorithm has approximation guarantee $O(n \cdot (\log \log n)^2/(\log n)^3)$. The algorithm for Independent Set is due to Feige [Fei04], the one for Coloring is due to Halldórsson [Hal93]. With respect to hypergraphs, Hofmeister and Lefmann [HL98] presented efficient algorithms for both problems with $d$-uniform hypergraphs as inputs, $d \geq 2$ fixed, and showed that the algorithms achieve approximation guarantee $O(n/(\log^{(d-1)} n)^2)$. Here, $\log^{(d-1)}$ denotes the $(d-1)$-fold iterated logarithm. Later, Halldórsson [Hal00] showed that the approximation guarantee is in fact $O(n/\log n)$ for all $d \geq 3$, improving the known performance guarantees of the algorithms. For fixed $d \geq 3$, Krivelevich and Sudakov [KS03] gave an efficient algorithm for Coloring with $d$-uniform hypergraphs as inputs, with approximation guarantee $O(n \cdot (\log \log n)^2/(\log n)^2)$.

Observe that for both problems, we can trivially achieve approximation guarantee $n$: Since all possible solutions have a value between 1 and $n$, no solution can have an approximation ratio larger than $n$. Thus, simply outputting a single vertex for Independent Set (or, of course, a coloring with $n$ colors for Coloring) achieves approximation ratio $n$. With this in mind, the approximation ratios of the form $O(n/(\log \log(n)))$ guaranteed by the above-mentioned algorithms seem not much better than what is trivially achievable. Unfortunately, however, no considerably better approximation guarantee seems achievable for our problems by efficient algorithms: For graphs as inputs, Håstad [Hås99] showed for Independent Set, and Feige and Kilian [FK96] for Coloring, that assuming $\text{NP} \not\subseteq \text{ZPP}$, for any $\varepsilon > 0$, no efficient algorithm with approximation guarantee $n^{1-\varepsilon}$ exists. For fixed $d \geq 3$, Hofmeister and Lefmann [HL98] extended these results to Independent Set and Coloring with $d$-uniform hypergraphs as inputs. Krivelevich and Sudakov [KS03] did the same for Coloring. A more recent result by Zuckerman [Zuc07] shows that the above inapproximability results for graphs also hold under the weaker assumption $\text{P} \neq \text{NP}$. It directly follows that the above inapproximability results for uniform hypergraphs in [HL98] and [KS03] also hold under the weaker assumption $\text{P} \neq \text{NP}$, since they are achieved via reductions from the graph case. For graphs, Khot and Ponnuswami [KP06] improved on the above bound $n^{1-\varepsilon} = 2^{(1-\varepsilon)\log n}$ as-
assuming $NP \not\subseteq \text{BPTIME}(2^{(\log n)^{O(1)}})$ holds: For any $\gamma > 0$, no efficient algorithm with approximation guarantee $n/2^{(\log n)^{3/4+\gamma}}$ exists, which is $2^{(1-o(1))}\log n$ if $\gamma < 1/4$. These inapproximability results strongly suggest that for every $d \geq 2$, all efficient approximation algorithms for \textsc{Independent Set} and \textsc{Coloring} on $d$-uniform hypergraphs in the worst case achieve almost trivial approximation ratios.

As mentioned above, we perform average-case analyses for \textsc{Independent Set} and \textsc{Coloring} on random uniform hypergraphs and a probabilistic analysis for \textsc{Independent Set} on semi-random graphs. These analyses show that the problems are easier to approximate on the average, i.e., our algorithms guarantee approximation ratios that beat the above inapproximability bounds and have polynomial running time in expectation. Section 1.2.1 contains details of our results for random hypergraphs. The results for semi-random graphs are presented in Section 1.2.2. A discussion of previous and related work for all our \textsc{Independent Set} and \textsc{Coloring} results can be found in Section 1.2.3.

### 1.2.1 Approximating Independent Set and Coloring for Random Hypergraphs

We define the model of random hypergraphs we use in the average-case analyses for \textsc{Independent Set} and \textsc{Coloring}. For a natural uniformity constant $d \geq 2$, our random $d$-uniform hypergraphs are produced as follows: For a number $n \in \mathbb{N}$ of vertices and an edge probability $0 \leq p \leq 1$, we first create $n$ labeled vertices. Then, each of the $\binom{n}{d}$ possible edges of cardinality $d$ is inserted independently of the others with probability $p$. We denote the resulting probability distribution by $H_d(n, p)$. Notice that $H_d(n, p)$ canonically generalizes the well-known $G(n, p)$ model, and $H_2(n, p) = G(n, p)$.

Krivelevich and Vu [KV02] performed an average-case analysis for \textsc{Independent Set} and \textsc{Coloring} with random graphs from $G(n, p)$ as inputs. For both problems, they presented deterministic algorithms with polynomial expected running time over $G(n, p)$, where the edge probability $p = p(n)$ is required to fulfill $n^{-1/2+\varepsilon} \leq p \leq 3/4$ for fixed $\varepsilon > 0$. An approximation guarantee of $O((np)^{1/2}/\log n)$ is achieved by both algorithms. Notice that this approximation guarantee considerably beats the above-mentioned inapproximability bound of $n^{1-\varepsilon}$, $\varepsilon > 0$ arbitrarily small, for worst-case efficient algorithms.

**Our Results**

We extend Krivelevich and Vu’s result [KV02] from $G(n, p)$ to the $H_d(n, p)$ model. For \textsc{Independent Set}, we present two deterministic algorithms, called \texttt{ApproxIndSet} and \texttt{ImprovedIndSet}. The following theorem is proved for \texttt{ApproxIndSet}.
ApproxIndSet\(^{(\text{KV02})}\). For increasing uniformity constant \(d\), the ApproxIndSet has approximation guarantee \(O(n \cdot (p/\log n)^{(d-1)})\) and polynomial expected running time for random inputs \(H\) from \(H_d(n, p)\).

Notice that besides the random hypergraph \(H\), ApproxIndSet is given the edge probability \(p\) and the value \(\varepsilon\), which determines the lower bound for the possible values of \(p\).

For a smaller range of edge probabilities, we can improve the approximation guarantee of ApproxIndSet. We present an algorithm called ImprovedIndSet, and prove the following theorem.

**Theorem 2.** Fix an integer \(d \geq 2\). There is a constant \(c(d) > 0\) such that for probability \(p = p(n)\) with \(c(d) \cdot (\ln n)/n^{1-1/d} \leq p \leq 3/4\), ImprovedIndSet\((H, p)\) has approximation guarantee \(O(n^{1/2} \cdot p^{-(d-3)/(2d-2)}/(\log n)^{(d-1)})\) and polynomial expected running time for random hypergraphs \(H\) from \(H_d(n, p)\).

It is not hard to see that by choice of \(p = \Omega((\log n)/n^{1-1/d})\) in Theorem 2, ImprovedIndSet’s approximation guarantee \(O(n^{1/2} \cdot p^{-(d-3)/(2d-2)}/(\log n)^{(d-1)})\) is asymptotically smaller than the approximation guarantee \(O(n \cdot (p/\log n)^{(d-1)})\) of ApproxIndSet, so we actually achieve an improvement for this range of edge probabilities.

We briefly discuss the approximation ratio guaranteed by ImprovedIndSet for increasing uniformity constant \(d\). For \(d = 2\) (i.e., in the \(G(n, p)\) model), it guarantees approximation ratio \(O((np)^{1/2}/\log n)\), matching the one achieved by Krivelevich and Vu’s algorithm [KV02]. For \(d = 3\), approximation guarantee \(O((n/\log n)^{1/2})\) is achieved. For \(d = 4\), we get \(O(n^{1/2} \cdot p^{-1/6}/(\log n)^{1/3})\). For increasing \(d \geq 4\), notice that the approximation guarantee is

\[
O(n^{1/2} \cdot p^{-(d-3)/(2d-2)}/(\log n)^{(d-1)}) = O((n/p)^{1/2} \cdot (p/\log n)^{(d-1)})
\]

Since \((p/\log n)^{(d-1)} = o(1)\), the approximation guarantee is for every value of \(d \geq 4\) better than \(O((n/p)^{1/2})\), but since \(\lim_{d \to \infty} 1/(d-1) = 0\), the improvement gets smaller and smaller as \(d\) approaches infinity. Thus, for large \(d\), the approximation guarantee is not much better than \(O((n/p)^{1/2})\).

We discuss the idea of algorithm ImprovedIndSet. Like the corresponding one presented by Krivelevich and Vu in [KV02], it uses the following well-known greedy coloring algorithm as a subroutine. For the sake of simplicity and w.l.o.g., throughout Section 1.2 and Chapters 2 and 3, we always assume that the vertex set \(V\) in a hypergraph \(H = (V, E)\) is \(V = \{1, \ldots, n\}\).
Algorithm 1 GreedyColor(H)

1: Set \( C_1 := \{1\} \) and \( C := \{C_1\} \). Set \( \chi := 1 \). \( \triangleright \) \( \chi \) colors already processed vertices with \( \chi \) colors.

2: \( \textbf{for} \) each vertex \( v = 2, \ldots, n \) \( \textbf{do} \)
   
   \( \textbf{if} \) for some class \( C_i \in C \), the set \( C_i \cup \{v\} \) is independent \( \textbf{then} \)
   
   Set \( C_i := C_i \cup \{v\} \) for the smallest such \( i \).
   
   \( \textbf{else} \)
   
   Create new class \( C_{\chi+1} \). \( \triangleright \)
   
   Set \( \chi := \chi + 1 \), set \( C_{\chi} := \{v\} \), and \( C := C \cup \{C_{\chi}\} \).

3: \( \textbf{end for} \)

3: Output \( C \).

In other words, given a hypergraph, we scan the vertices one by one. For the current vertex, we assign the smallest possible color such that the coloring of the already scanned vertices remains feasible.

Considering the fact that GreedyColor is worst-case efficient together with the above-mentioned inapproximability results for Independent Set and Coloring, we cannot expect that it achieves an approximation ratio of the form \( n^{1-\varepsilon} \), \( \varepsilon > 0 \) fixed, for every graph. We can also easily prove that it performs worse for some inputs: Kučera [Kuč91] has shown that for every \( \delta > 0 \) and every vertex number \( n \), there is a graph on \( n \) vertices with chromatic number at most \( n^{\delta} \) such that the greedy algorithm uses \( \Omega(n/\log n) \) colors, even after a random permutation of the vertices. This only achieves approximation ratio \( \Omega(n^{1-\delta}/\log n) \), which for sufficiently small \( \delta \) exceeds \( n^{1-\varepsilon} \) for every fixed \( \varepsilon > 0 \). Thus, greedy coloring does not perform well for all graphs. However, we exploit the fact that it performs much better for almost all graphs (and uniform hypergraphs): With Lemmas 1 and 2 below, we show that for a random hypergraph from \( H_d(n, p) \), GreedyColor finds with probability exponentially close to 1 solutions for Independent Set and Coloring obeying certain thresholds. We hence almost surely find a coloring with “few” colors and a “large” independent set. Here and below, we say that an event happens almost surely if it occurs with probability \( 1 - o(1) \). Furthermore, Corollary 1 below shows that with probability exponentially close to 1, GreedyColor achieves a much better approximation ratio than \( n^{1-\varepsilon} \) for small enough \( \varepsilon > 0 \). To conclude, greedy coloring performs relatively well for almost all graphs and uniform hypergraphs.

In our approximation algorithms, however, it does not suffice to perform well for almost all hypergraphs: We want to guarantee that our algorithms achieve a certain approximation ratio for all uniform hypergraphs, while we accept expected polynomial running time in contrast to worst-case efficiency. The general idea of algorithm ImprovedIndSet is to compute a greedy independent set by running the greedy coloring algorithm and choosing a largest color class. It then checks whether this greedy independent set is sufficiently good, i.e., has the desired ap-
approximation ratio, in which case it is output. Otherwise, an optimal solution is found by exhaustive search, i.e., the algorithm considers all possible subsets of the vertex set and checks whether the current subset is independent. In other words, we separate the inputs that are easy to handle (a greedy solution suffices) from the hard ones. This approach is often applied when designing an algorithm with polynomial expected running time over some probability space of inputs. In our case, the separation is done using a spectral technique: We compute the largest eigenvalues of some auxiliary matrices. Appendix A.2 contains a summary of important definitions and facts regarding eigenvalues, together with some proofs of properties we use to achieve our results.

In a meta-form, algorithm ApproxIndSet for random hypergraphs \( H \) from \( H_d(n, p) \) can be described as follows:

**Algorithm 2 Meta-IndSet(\( H, p \))**

1: Compute a greedy coloring \( C \) for \( H \) and let \( I \) be a largest color class in \( C \).
2: Compute an upper bound \( \lambda(H) \) on \( \alpha(H) \) with a spectral technique.
3: if \(|I| = \Omega((\log n)/p)^{1/(d-1)}) \) and \( \lambda(H) = O((n/p)^{1/2}) \) then output \( I \).
4: Find a maximum independent set by exhaustive search and output it.

In words, we check whether the greedy independent set is sufficiently large and whether the independence number of \( H \) is not too large. By choice of the thresholds for \(|I|\) and \( \lambda(H) \) in Step 3, outputting the greedy solution \( I \) guarantees the approximation ratio from Theorem 2. Exhaustive search even finds an optimal solution. A polynomial expected running time follows from the fact that we can show, that the exhaustive search with its exponential effort is executed only with exponentially small probability.

Analogously, we give two algorithms for Coloring, called ApproxColoring and ImprovedColoring. For ApproxColoring, we prove the following theorem.

**Theorem 3.** Fix an integer \( d \geq 2 \) and \( \varepsilon > 0 \). For probability \( p = p(n) \) with

\[
\frac{1}{n^{\frac{d-1}{2} - \varepsilon d}} \leq p \leq 3/4,
\]

ApproxColoring(\( H, p, \varepsilon \)) has approximation guarantee \( O(n \cdot (p/\log n)^{1/(d-1)}) \) and polynomial expected running time for random inputs \( H \) from \( H_d(n, p) \).

For the sake of clarity (and due to a technical detail in the algorithm), the analysis for ImprovedColoring is separated into the case \( d \geq 3 \) and the case \( d = 2 \). The following two theorems are proved.
Theorem 4. Fix an integer \( d \geq 3 \) and set \( \varepsilon := 1/(d \cdot 2^d) \). There is a constant \( c(d) > 0 \) such that for probability \( p = p(n) \) with \( c(d) \cdot (\ln n)/n^{1-1/d} \leq p \leq 3/4 \), \( \text{ImprovedColoring}(H, p, \varepsilon) \) has approximation guarantee \( O(n^{1/2} \cdot p^{-(d-3)/(2d-2)}/(\log n)^{(d-1)/d}) \) and polynomial expected running time for random hypergraphs \( H \) from \( H_d(n, p) \).

The auxiliary parameter \( \varepsilon \) is used only as a parameter that adapts the parametrized algorithm to different values of \( d \). We use it for the sake of a clear presentation. It has no influence on the range of possible edge probabilities or the approximation guarantee we achieve.

Theorem 5. Fix \( \varepsilon > 0 \). For probability \( p = p(n) \) with \( 2\varepsilon \cdot (\ln n)^{3/2}/n^{1/2-2\varepsilon} \leq p \leq 3/4 \), \( \text{ImprovedColoring}(H, p, \varepsilon) \) has approximation guarantee \( O((np)^{1/2}/\log n) \) and polynomial expected running time for random hypergraphs \( H \) from \( G(n, p) \).

Notice that our algorithms for \textsc{Independent Set} and \textsc{Coloring} have matching approximation guarantees, only the ranges of edge probabilities differ.

As mentioned, our results generalize the ones presented by Krivelevich and Vu [KV02] from \( G(n, p) \) to \( H_d(n, p) \). Our algorithms \text{ImprovedIndSet} and \text{ImprovedColoring} are extensions of corresponding ones in [KV02]. For \( d = 2 \), we also achieve some improvements. Firstly, we improve the lower bound on the possible edge probabilities \( p \) of the algorithm for \textsc{Independent Set} in [KV02] from \( p \geq 1/n^{1/2-\varepsilon} \) to \( p = \Omega((\log n)/n^{1/2}) \) (cf. Theorem 2), an improvement of factor \( \Omega(n^\varepsilon/\log n) \). This is achieved since the lower bound on the possible edge probabilities \( p \) in Lemma 1 below is smaller than the one of a corresponding Lemma in [KV02]. Roughly, Lemma 1 states that for a random hypergraph drawn from \( H_d(n, p) \), with probability exponentially close to 1, the greedy independent set has size \( \Omega((\log n)/n^{1/2}) \) (cf. Step 3 in Algorithm 2).

Secondly, Lemma 2 below states that for a random \( H_d(n, p) \) hypergraph, with probability at least \( 1 - e^{-n \ln n} \), the number of color classes used by greedy coloring is \( O(n \cdot (p/\log n)^{1/(d-1)}) \). The upper bound \( e^{-n \ln n} \) for the failure probability improves on the larger bound \( 2^{-2np/\ln n} \) of a corresponding lemma in [KV02]. Due to this, algorithm \text{ImprovedColoring} is a bit simpler than the corresponding one in [KV02]. The original algorithm performs seven steps, while we only need five.

Next, we state Lemmas 1 and 2 mentioned above. They are used in the proofs of Theorems 1–5 in Chapter 2, but their tail bounds on the size of the greedy independent set and the number of color classes used by greedy coloring for random hypergraphs from \( H_d(n, p) \) might be of their own interest. For a coloring \( C \), denote by \( \text{big}(C) \) its largest color class (break ties arbitrarily).
Lemma 1. Fix an integer $d \geq 2$ and $\varepsilon > 0$. There is a constant $c(d, \varepsilon) > 0$ such that for probability $p = p(n)$ with $c(d, \varepsilon) \cdot (\ln n)^d/n^{d-1-\varepsilon} \leq p \leq 3/4$, the following holds. Let $C$ be the coloring computed by $\text{GreedyColor}(H)$ for a random hypergraph $H$ from $H_d(n, p)$. Then,

$$\Pr[|\text{big}(C)| \leq ((d - 2)! \cdot \varepsilon \cdot (\ln n)/(2p))^{1/(d-1)}] \leq e^{-n \ln n}.$$  

Lemma 2. Fix an integer $d \geq 2$ and $\varepsilon > 0$. Let $p = p(n)$ be a probability with

$$\frac{4d! \varepsilon \cdot (\ln n)^{d-1}}{n^{d-1} \cdot (\ln n)^d} \leq p \leq 3/4,$$

and let $C$ be the coloring computed by $\text{GreedyColor}(H)$ for a random hypergraph $H$ from $H_d(n, p)$. Then, for the number $|C|$ of classes in $C$, we have

$$\Pr[|C| \geq 2n \cdot (p/(\varepsilon \cdot \ln n))^{1/(d-1)}] < e^{-n \ln n}.$$  

On the one hand, we have our results about $\text{ImprovedIndSet}$ and $\text{ImprovedColoring}$, showing that these algorithms achieve approximation guarantee $O(n^{1/2} \cdot p^{-(d-3)/(2d-2)}/(\log n)^{(d-1)/d})$ in polynomial expected running time over $H_d(n, p)$. On the other hand, we get as a simple corollary that by running $\text{GreedyColor}$ alone, we have an algorithm with worst-case efficiency that achieves a constant approximation ratio in expectation and with high probability. Therefore, we have a trade-off between the guarantees on running time and solution quality. This is summarized by Corollary 1. We mention that Schmidt-Pruzan, Shamir, and Upfal [SPSU85] proved that a modified greedy coloring algorithm achieves almost surely a constant approximation ratio for random uniform hypergraphs.

Corollary 1. Under the assumptions of Lemma 1 for $\text{Independent Set}$ and Lemma 2 for $\text{Coloring}$, the expected approximation ratio achieved by $\text{GreedyColor}(H)$ for a random hypergraph $H$ from $H_d(n, p)$ is $O(1)$. Furthermore, it achieves approximation ratio $O\left(n^{1/d} \cdot \left(\frac{p}{\log n}\right)^{\frac{d-1}{d}}\right)$ with probability at least $1 - 2 \cdot e^{-n \ln n}$.

Spectral Techniques  As mentioned, our algorithms are extensions of the ones presented by Krivelevich and Vu [KV02]. The algorithms use a spectral technique to compute an upper bound on the independence number of a given graph $G$ that was randomly drawn from $G(n, p)$. To be more precise, the largest eigenvalue of an auxiliary matrix is computed, yielding the desired upper bound $\lambda(G)$ on $\alpha(G)$ (cf. Step 2 in Algorithm 2). Spectral techniques, i.e., computing eigenvalues of suitable auxiliary matrices and interpreting their values, have proved to be very successful for solving hard combinatorial problems, especially solving problems on random
graphs. For a graph \( G = (V, E) \), \( V = \{1, \ldots, n\} \), and a value \( p \in (0, 1) \), Krivelevich and Vu [KV02] define the \( n \times n \)-matrix \( M(G, p) = (m_{ij})_{1 \leq i, j \leq n} \) by

\[
m_{ij} := \begin{cases} 
1 & \text{if } \{i, j\} \notin E \\
-(1 - p)/p & \text{otherwise.}
\end{cases}
\]

(1.10)

For an example, see Figure 1.2, which shows a graph \( G \) on four vertices, together with the resulting matrix \( M(G, p) \).

Let \( \lambda_1(M) \) denote the largest eigenvalue of \( M \). Then, for random graphs \( G \) from \( G(n, p) \), the following interesting lemma connecting the independence number \( \alpha(G) \) and the largest eigenvalue of the resulting random matrix \( M(G, p) \) is proved in [KV02]:

**Lemma 3.** For every graph \( G \) and every value \( p \in (0, 1) \), it holds that \( \alpha(G) \leq \lambda_1(M(G, p)) \). Furthermore, for probability \( p = p(n) = \omega(n^{-1}) \) and a random graph \( G \) from \( G(n, p) \), it holds that \( \Pr[\lambda_1(M(G, p)) \geq 4 \cdot (n/p)^{1/2}] \leq 2^{-np/8} \).

Given a probability \( p \) and a random graph \( G \) from \( G(n, p) \), one can compute the largest eigenvalue \( \lambda_1(M(G, p)) \) in polynomial time, since \( M \) is real and symmetric by construction (see e.g. Ralston [Ral85]). Due to Lemma 3, with probability at least \( 1 - 2^{-np/8} \), one can efficiently certify that \( \alpha(G) = O((n/p)^{1/2}) \). In our algorithms, we use this technique to efficiently certify with high probability that the independence number \( \alpha(H) \) of a random \( H(n, p) \) hypergraph \( H \) is at most \( O((n/p)^{1/2}) \) (cf. Step 3 in Algorithm 2). This is achieved by constructing some auxiliary graphs for \( H \) and computing the largest eigenvalues of the resulting matrices for the graphs.

**Small Edge Probabilities** Applying Theorem 1, we can approximate INDEPENDENT SET with algorithm ApproxIndSet for edge probabilities as small as \( p = 1/n^{d-1-\varepsilon} \) for arbitrarily small, fixed \( \varepsilon > 0 \). It is interesting to ask what we can
1.2. Results for Independent Set and Coloring

do for even smaller edge probabilities. Here, we can use the following idea: Spencer [Spe72] gave a lower bound \( l(d, n, m) \) on the independence number \( \alpha(H) \) that holds for every \( d \)-uniform hypergraph \( H \) with \( n \) vertices and \( m \) edges. Notice that the lower bound \( l(d, n, m) \) depends only on \( d \), \( n \), and \( m \), and not on the concrete edge set of the hypergraph. The smaller the number \( m \) of edges (for fixed \( d \), \( n \)), the larger the bound \( l(d, n, m) \) gets. For hypergraphs with “few” edges, a “large” independence number follows. Bertram-Kretzberg and Lefmann [BKL99] turned this into an algorithmic result by showing that for any \( d \)-uniform hypergraph \( H \) with \( n \) vertices and \( m \) edges, one can compute deterministically in worst-case polynomial time an independent set \( I \) of \( H \) with \( |I| \geq l(d, n, m) \).

Using the algorithm from [BKL99], which we call \texttt{EdgeNumIndSet} in the following, we can proceed as follows. For very small edge probabilities \( p \), a random \( H_d(n, p) \) hypergraph \( H \) contains with high probability “few” edges, and in this case, using \texttt{EdgeNumIndSet}(\( H \)), we can find in polynomial time a “large” independent set, which guarantees a certain approximation ratio. On the other hand, if \( H \) contains “many” edges, which happens only with exponentially small probability, we perform an exhaustive search step, finding an optimal solution. In the end, we again have a polynomial expected running time, since the execution probability of the exhaustive search step will be sufficiently small. In Section 2.5, we present an algorithm following this idea. We call it \texttt{SmallEdgePrIndSet}(\( H, p \)) for random hypergraphs \( H \) from \( H_d(n, p) \), and prove the following theorem.

\textbf{Theorem 6.} Fix an integer \( d \geq 2 \) and an arbitrary edge probability \( p = p(n), 0 < p < 1 \). Let \( H \) be a random hypergraph from \( H_d(n, p) \). Then, algorithm \texttt{SmallEdgePrIndSet}(\( H, p \)) has polynomial expected running time. For \( p \geq 1/n^{d-1} \), it has approximation guarantee \( O(n \cdot p^{1/(d-1)}) \). For \( p < 1/n^{d-1} \), it has approximation guarantee \( O(1) \).

Notice that for \( p \geq 1/n^{d-1} \), the approximation guarantee up to a factor of \( (\log n)^{1/(d-1)} \) matches the one achieved by \texttt{ApproxIndSet} (see Theorem 1). Notice also that for \( p = O(1/n^{d-1}) \), we have a constant approximation guarantee. Finally, we mention again that we can handle arbitrarily small edge probabilities.

1.2.2 Approximating Independent Set in Semi-Random Graphs

We start by introducing the model of semi-random graphs used in our analysis. As mentioned, we perform a probabilistic analysis in the spirit of smoothed analysis, i.e., we use a perturbation model of random inputs in which a malicious adversary chooses an arbitrary input which is then perturbed by small random modifications. As mentioned, this model was proposed by Spielman and Teng [ST09] in their survey paper on smoothed analysis, where they call it a “smoothed extension” of the \( G(n, p) \) model. We introduce it formally now.
Let a potential edge of a graph be a set \( \{v, w\} \) of two different vertices \( v, w \) in the graph. Thus, a graph with \( n \) vertices contains \( \binom{n}{2} \) potential edges. In our semi-random graph model, the adversary first chooses a graph \( G = (V, E) \). Then, for a flip probability \( \varepsilon > 0 \), a random graph \( \mathcal{G} = (V, \mathcal{E}) \) is produced by negating the existence of every potential edge in \( G \) independently with probability \( \varepsilon \). In other words, we insert the non-edges and remove the edges of \( G \) independently with probability \( \varepsilon \).

We denote the resulting probability distribution of \( \mathcal{G} \) by \( \mathcal{G}(G, \varepsilon) \). Formally, in the \( \mathcal{G}(G, \varepsilon) \) model, each of the \( \binom{n}{2} \) potential edges \( e \) of \( G \) has a probability \( p_e \) of finally appearing in the edge set \( \mathcal{E} \) of the semi-random graph \( \mathcal{G} = (V, \mathcal{E}) \), and

\[
p_e = \begin{cases} 
1 - \varepsilon & \text{if } e \in \mathcal{E} \text{ and } \varepsilon \\
\varepsilon & \text{if } e \notin \mathcal{E}.
\end{cases}
\tag{1.11}
\]

Consider the extreme case \( \varepsilon = 0 \). Here, we deterministically get \( \mathcal{G} = G \). Thus, the adversary has full power and can specify worst-case instances. For increasing flip probability \( \varepsilon \), the adversary loses power. When reaching \( \varepsilon = 1/2 \), every potential edge appears in \( \mathcal{G} \) with probability \( 1/2 \). Hence, we simply have a \( G(n, 1/2) \) model, and the adversarial graph \( G \) has no influence on our random graph. To summarize, the value of \( \varepsilon \) determines the “amount of randomness” in \( \mathcal{G} \). The reason why \( \mathcal{G}(G, \varepsilon) \) is a “smoothed extension” of \( G(n, p) \) is that for the empty graph \( G_0 = (V, \emptyset) \), we have \( \mathcal{G}(G_0, \varepsilon) = G(n, p) \) for \( n = |V| \) and \( p = \varepsilon \). Thus, \( G(n, p) \) is a special case of \( \mathcal{G}(G, \varepsilon) \).

Our Results

In Chapter 3, we present our algorithm \texttt{AdversaryIndSet} for INDEPENDENT SET in the \( \mathcal{G}(G, \varepsilon) \) model. Like our algorithm for random hypergraphs, it follows the approach of \texttt{Meta-IndSet} (Algorithm 2), i.e., one checks whether the greedy independent set in \( \mathcal{G} \) is large enough and whether the independence number \( \alpha(\mathcal{G}) \) is not too large. We prove the following theorem.

**Theorem 7.** Fix a graph \( G \) and a flip probability \( n^{-1/2} \leq \varepsilon \leq 1/2 \). Let \( \mathcal{G} \) be drawn from \( \mathcal{G}(G, \varepsilon) \). Then, \texttt{AdversaryIndSet}(\( \mathcal{G}, G, \varepsilon \)) has polynomial expected running time. Furthermore, if \( \varepsilon \) is sufficiently high, i.e., \( \frac{\ln(1/\varepsilon)}{\varepsilon} \leq \frac{\alpha^2}{|\mathcal{E}|} \), it has approximation guarantee \( O((n\varepsilon)^{1/2}) \). Otherwise, the approximation guarantee is \( O\left(\frac{|\mathcal{E}|}{n^{1/2}} \cdot \frac{\log(1/\varepsilon)}{\varepsilon^{1/2}}\right) \).

Analogously to the \( H_\mu(n, p) \) model, if we run \texttt{GreedyColor} alone, we get an algorithm with polynomial worst-case running time, but the approximation ratio then holds only in expectation and with high probability. This complements the performance of \texttt{AdversaryIndSet} and shows again a trade-off between guarantees on running time and approximation ratio.
Corollary 2. Fix a constant \( \delta \in (0, 1) \). Fix a graph \( G = (V, E) \) and a flip probability \( \varepsilon = \varepsilon(n) \) with \( n^{-(1-\delta)} \leq \varepsilon \leq 1/2 \). Let \( \mathcal{G} \) be drawn from \( \mathcal{G}(G, \varepsilon) \), and let \( I \) be the largest color class computed by \( \text{GreedyColor}(\mathcal{G}) \). Let \( \varepsilon \) be high if \( \frac{\ln(1/\varepsilon)}{\varepsilon} \leq \frac{n^2}{|E|} \) and small otherwise. Then, the expected approximation ratio \( E[ar(I)] \) is

\[
E[ar(I)] = \begin{cases} 
O(1) & \text{if } \varepsilon \text{ is high and } \\
O\left(\frac{|E|}{n^2} \cdot \frac{\log(1/\varepsilon)}{\varepsilon}\right) & \text{if } \varepsilon \text{ is small.}
\end{cases}
\]

Furthermore, with probability at least \( 1 - 2 \cdot e^{-n \ln n} \), the approximation ratio fulfills

\[
ar(I) = \begin{cases} 
O\left(\left(\frac{n^2 \ln n}{\log n}\right)^{1/2}\right) & \text{if } \varepsilon \text{ is high and } \\
O\left(\frac{|E|^{3/2}(\log n)^{3/2}}{n^{3/2} \cdot e^{1/2}}\right) & \text{if } \varepsilon \text{ is small.}
\end{cases}
\]

In our analysis of \( \text{AdversaryIndSet} \), we use tail bounds on the greedy independent set size and the largest eigenvalue of an auxiliary matrix we define for a random \( \mathcal{G}(G, \varepsilon) \) graph. These tail bounds may be of their own interest and are given in the next paragraphs.

A Tail Bound on the Greedy Independent Set Size of \( \mathcal{G}(G, \varepsilon) \) For convenience, we define an abbreviation for the threshold used for the greedy independent set in algorithm \( \text{AdversaryIndSet} \). For an adversarial graph \( G = (V, E) \) and \( \varepsilon, \delta > 0 \), let

\[
gis(G, \varepsilon, \delta) := \frac{\delta}{16} \cdot \min\left\{ \frac{\ln n}{\varepsilon}, \frac{n^2 \ln n}{|E| \ln(1/\varepsilon)} \right\}.
\]

(1.12)

Lemma 4 gives our tail bound on the size of the greedy independent set in a random \( \mathcal{G}(G, \varepsilon) \) graph. It states that with probability exponentially close to 1, the greedy independent set has size at least \( gis(G, \varepsilon) \). If, like here, we omit the parameter \( \delta \) in \( gis(G, \varepsilon, \delta) \), we assume that it is fixed in some way. The parameter is only used to specify the range of possible flip probabilities \( \varepsilon \). Recall that for a coloring \( C \), \( \text{big}(C) \) denotes a largest color class.

Lemma 4. Fix \( \delta > 0 \). Fix a graph \( G = (V, E) \) and let \( \varepsilon = \varepsilon(n) \) be a flip probability with \( n^{-(1-\delta)} \leq \varepsilon \leq 1/2 \). Let \( \mathcal{G} \) be a random graph from \( \mathcal{G}(G, \varepsilon) \), and let \( C \) be the coloring computed by \( \text{GreedyColor}(\mathcal{G}) \). Then,

\[
\Pr[|\text{big}(C)| < gis(G, \varepsilon, \delta)] \leq e^{-n \ln n}.
\]

We discuss the influence of the adversarial graph \( G = (V, E) \) and the flip probability \( \varepsilon \). Depending on \( |E| \) and \( \varepsilon \), either \( \frac{\ln n}{\varepsilon} \) or \( \frac{n^2 \ln n}{|E| \ln(1/\varepsilon)} \) is smaller in (1.12) and accordingly determines \( gis(G, \varepsilon) \). Intuitively, \( \frac{\ln n}{\varepsilon} \) gives the size of the independent set we can expect to find if \( |E| = 0 \), since in this case, \( \frac{n^2 \ln n}{|E| \ln(1/\varepsilon)} \) has infinite value,
and thus $\frac{\ln n}{\varepsilon}$ determines the minimum. In this case, we actually have a $G(n, \varepsilon)$ model, and also the resulting threshold $\text{gis}(G, \varepsilon) = \Theta\left(\frac{\log n}{\varepsilon}\right)$ asymptotically matches the one in Lemma 1 for the $G(n, \varepsilon)$ model (and also the one in a corresponding result in [KV02]). Furthermore, $\frac{\ln n}{\varepsilon}$ also gives what we can expect to find if the adversarial graph has little influence, i.e., $\varepsilon$ is sufficiently high: It is easy to see that if $\varepsilon$ is sufficiently high (assume $|E|$ fixed), $\frac{n^2 \ln n}{|E| \ln(1/\varepsilon)}$ in (1.12) is larger than $\frac{\ln n}{\varepsilon}$, and again, $\frac{\ln n}{\varepsilon}$ determines the minimum.

However, with decreasing $\varepsilon$ and increasing $|E|$, $G$ gains influence (i.e., the term corresponding to its influence decreases), while $\frac{\ln n}{\varepsilon}$ increases. If and only if $\frac{n^2 \ln n}{|E|} > \frac{\ln n}{|E|}$, the term for $G$ defines $\text{gis}(G, \varepsilon)$. This is also reflected in Theorem 7 and Corollary 2: Notice the distinction between high and small flip probabilities, based on whether $\frac{\ln(1/\varepsilon)}{\varepsilon} \leq \frac{n^2}{|E|}$ or not.

**A Tail Bound on the Largest Eigenvalue** In Section 1.2.1, we sketched the spectral technique used by Krivelevich and Vu [KV02] to upper bound the independence number of a random $G(n, p)$ graph. We adapt the technique to our $G(G, \varepsilon)$ model. To this end, we define a suitable matrix.

Let $G = (V, E)$ be an adversarial graph, $\varepsilon > 0$ be a flip probability, and $G = (V, E)$ be a random graph drawn from our $G(G, \varepsilon)$ model. Remember (cf. (1.11) in Section 1.2.2) that for a potential edge $e$ of $G$, we denote by $p_e$ the probability that $e \in E$, and either $p_e = \varepsilon$ (if $e \notin E$), or $p_e = 1 - \varepsilon$ (if $e \in E$). Now, let $A = A(G, G, \varepsilon)$ be the $n \times n$-matrix $(a_{ij})_{1 \leq i, j \leq n}$ with

$$a_{ij} = \begin{cases} 1 & \text{if } e = \{i, j\} \notin E \text{ and} \\ -(1 - p_e)/p_e & \text{if } e = \{i, j\} \in E. \end{cases}$$

(1.13)

Notice that $a_{ij}$ depends on whether $e = \{i, j\} \in E$ and whether $e \in E$, since the latter determines the probability $p_e$ that $e \in E$ as discussed above. Furthermore, we mention that (1.13) canonically extends the definition of the matrix used by Krivelevich and Vu in [KV02] (cf. its definition in (1.10)) to handle two types of edge probabilities in $G(G, \varepsilon)$, compared to only one type in $G(n, p)$.

Lemma 5 gives our tail bound on the largest eigenvalue of $A(G, G, \varepsilon)$ for a random graph $G$ drawn from $G(G, \varepsilon)$. It essentially transfers Lemma 3 to $G(G, \varepsilon)$. Besides the fact that we use Lemma 5 as a tool in our analysis, the lemma might be of its own interest, since the polynomial-time computability of $\lambda_1(A)$ yields a general technique for efficiently computing an upper bound on the independence number of a random $G(G, \varepsilon)$ graph that is small (in the lemma’s sense) with high probability.
1.2. Results for Independent Set and Coloring

**Lemma 5.** Fix a graph $G$ and $\varepsilon = \varepsilon(n) \leq 1/2$ with $\varepsilon = \Omega((\log n)^2/n)$. Let $\mathcal{G}$ be drawn from $\mathcal{G}(G, \varepsilon)$, and let $A := A(\mathcal{G}, G, \varepsilon)$. Then, $\alpha(\mathcal{G}) \leq \lambda_1(A)$. Furthermore,

$$E[\lambda_1(A)] \leq 2^7 \cdot (\log n) \cdot (n/\varepsilon)^{1/2}$$

and

$$\Pr[\lambda_1(A) \geq 2^8 \cdot (\log n) \cdot (n/\varepsilon)^{1/2}] \leq 4 \cdot \exp(-2^9 \cdot n \varepsilon \cdot (\log n)^2).$$

**Small Flip Probabilities** As for our results in random hypergraphs, it is interesting to ask what we can do for small flip probabilities $\varepsilon > 0$, i.e., below the range we can handle with Theorem 7. The theorem works only for $\varepsilon \geq n^{-1/2}$. For arbitrary flip probabilities $\varepsilon > 0$, we can achieve the following.

Theorem 6 above gives our analysis of algorithm $\text{SmallEdgePrIndSet}(H, p)$, with which we can approximate INDEPENDENT SET in random uniform hypergraphs from $H_d(n, p)$ for arbitrary edge probabilities $0 < p < 1$. Here, we have a random graph $\mathcal{G}$ from $\mathcal{G}(G, \varepsilon)$. By simply running $\text{SmallEdgePrIndSet}(\mathcal{G}, p^*)$, where we choose a suitable value $p^*$, we can achieve a similar result for arbitrarily small flip probabilities $\varepsilon$ in $\mathcal{G}(G, \varepsilon)$ as we achieved for small edge probabilities $p$ in $H_d(n, p)$.

We stress that $p^*$ (cf. (1.14) in the below theorem) is not an edge probability of a random $G(n, p^*)$ graph but a value we compute from the adversarial graph $G$ and the flip probability $\varepsilon$ to suit our needs: It is the average probability $\frac{1}{\binom{n}{2}} \cdot \sum e_p e$ of the potential edges $e$ in the random graph. In Section 3.5, we prove the following theorem:

**Theorem 8.** Fix a graph $G = (V, E)$ and an arbitrary flip probability $\varepsilon = \varepsilon(n)$, $0 < \varepsilon \leq 1/2$. Let $\mathcal{G}$ be drawn from $\mathcal{G}(G, \varepsilon)$. Let

$$p^* := \frac{|E| \cdot (1 - \varepsilon) + \left(\binom{n}{2} - |E|\right) \cdot \varepsilon}{\binom{n}{2}}. \quad (1.14)$$

Then, $\text{SmallEdgePrIndSet}(\mathcal{G}, p^*)$ has polynomial expected running time. If $p^* \geq 1/n$, it has approximation guarantee $O(np^*)$. If $p^* < 1/n$, it has approximation guarantee $O(1)$. 

1.2.3 Previous and Related Work

In the following, we mention some previous and related work for INDEPENDENT SET and COLORING. With respect to the notion of independence of a set of vertices, different generalizations from graphs to hypergraphs have been studied. For a $d$-uniform hypergraph $H = (V, E)$, the definition of an independent set used throughout this thesis is the one of a $\gamma$-independent set for $\gamma = d - 1$. A set $I \subseteq V$ is
\(\gamma\)-independent if for every edge \(e \in E\), it holds that \(|e \cap I| \leq \gamma\). Independent sets in our sense are also called \textit{weak independent sets}, since we only require that for no edge \(e \in E\), all of \(e\)'s vertices lie in \(I\). A stronger requirement would be that for every edge \(e \in E\), at most one vertex of \(e\) lies in \(I\). This is the notion of \(\gamma\)-independence for \(\gamma = 1\).

Since we defined a feasible coloring of the vertices in a hypergraph as a coloring \(C = \{C_1, \ldots, C_k\}\) such that every class \(C_i\) is independent, a coloring in our sense is feasible if and only if there is no monochromatic edge. Accordingly, our notion of the chromatic number of a hypergraph is the one of the \textit{weak chromatic number}. In contrast, if one demands \(\gamma\)-independence with \(\gamma = 1\) for the color classes, the \textit{strong chromatic number} results, where in each edge, the vertices must have pairwise different colors. For a discussion of the above-mentioned generalized notions of independent sets and colorings, see e.g. Schmidt-Pruzan, Shamir, and Upfal [SPSU85].

We briefly discuss the upper bound on the possible edge probabilities \(p\) in our results from Section 1.2.1, i.e., for the random hypergraphs from \(H_d(n, p)\). As mentioned above, the algorithms presented by Krivelevich and Vu [KV02] assume that the edge probability \(p\) is in the range \(n^{-1/2+\varepsilon} \leq p \leq 3/4\). For \textsc{Coloring}, this was later improved by Coja-Oghlan and Kuhtz [COK06] to \(c/n \leq p \leq 0.99\) for a certain constant \(c > 0\). It is not hard to see that our upper bound \(p \leq 3/4\) can be improved to \(p \leq 1 - \varepsilon\) for arbitrarily small, fixed \(\varepsilon > 0\) without affecting our results asymptotically. This is because the assumption \(p \leq 3/4\) is only needed to fulfill the estimation \(1 - p \geq e^{-2p}\), which we use several times in our proofs. Enlarging the range of \(p\) to \(p \leq 1 - \varepsilon\) for a constant \(\varepsilon > 0\) only changes the constant 2 in \(e^{-2p}\) to a larger one, i.e., for \(p \leq 1 - \varepsilon\) we have \(1 - p \geq e^{-dp}\) for a constant \(d > 0\) sufficiently large. One can verify that this modification to the calculations in our proofs does not affect our results asymptotically. A discussion of these inequalities can be found in Appendix A.1.

\section*{1.3 Results for Shortest Common Superstring}

The problem \textsc{Shortest Common Superstring} is motivated by applications like DNA sequencing (see e.g. Gusfield [Gus97]) and data compression. For example, to compress a set of \([0, 1]\)-strings, it suffices to store a single string \(t\) that contains every string to be compressed as a (closed) substring, together with starting and ending positions of the individual strings in \(t\). If \(t\) is shorter than the concatenation of the individual strings, a compression is achieved (neglecting the space used for storing the positions).

Formally, we consider strings over a finite \textit{alphabet} \(\Sigma\), which is simply a finite set. A \textit{string} \(s\) of \textit{length} \(l\) over \(\Sigma\) is a sequence \(s = s_1s_2\ldots s_l\) of \textit{letters} \(s_i \in \Sigma\). We
1.3. Results for Shortest Common Superstring

1.3.1. Results for Shortest Common Superstring

Denote $s$’s length by $|s|$. The set of all strings of length $t$ is denoted $\Sigma^t$, and we let $\Sigma^* := \bigcup_{t \geq 0} \Sigma^t$.

For strings $s, t$ with $|s| \leq |t|$, we say that $s$ is a substring of $t$ if $s$ is a contiguous subsequence of $t$, i.e., for some starting index $1 \leq j \leq |t| - |s| + 1$, we have $s_i = t_{j-1+i}$, $i = 1, \ldots, |s|$. In other words, $s$ appears in $t$, starting at some position $j$. Given a multiset $S = \{s^1, \ldots, s^n\}$ of strings over $\Sigma$, a string $t \in \Sigma^*$ is a superstring for $S$ if every $s^i \in S$ is a substring of $t$. Throughout Section 1.3 and Chapter 4, $n$ always denotes the number of strings in a multiset of strings under investigation. We can now define the problem Shortest Common Superstring.

**Problem Shortest Common Superstring**

**Input:** A multiset $S = \{s^1, \ldots, s^n\}$ of strings over a finite alphabet $\Sigma$

**Task:** Find an as short as possible superstring $t \in \Sigma^*$ for $S$.

Shortest Common Superstring is NP-hard, even under several restrictions on the lengths of the strings or the size of the underlying alphabet, see e.g. Gallant, Maier, and Storer [GMS80] and Middendorf [Mid94]. Due to the problem’s important applications, one is therefore interested in finding efficient approximation algorithms. Before we mention some results on the polynomial-time approximability of Shortest Common Superstring, we define some notions used in the following.

Given a string $s$, for $0 \leq k \leq |s|$, $s$’s prefix and suffix of length $k$ are the strings $s_1 \ldots s_k$ and $s_{|s|-k+1} \ldots s_{|s|}$, respectively. For two strings $s, t$, the overlap $\text{ov}(s, t)$ of $s$ and $t$ is the longest suffix of $s$ that is also a prefix of $t$. For example, $\text{ov}(\text{mankind, kindness}) = \text{kind}$. Notice that the overlap of two strings depends on the order in which they are given. Finally, let $s \oplus t$ denote the string resulting from maximally overlapping $s$ and $t$, i.e., the string resulting from concatenating $s$ with the suffix of $t$ of length $|t| - |\text{ov}(s, t)|$. For our two example strings, we get mankind $\oplus$ kindness = mankindness.

Next, we define some approximation-related terms. We consider two quality measures for a superstring $t$ of an input $S$. The first is $t$’s length, which we want to minimize. Given $S$ and $t$, let $t$’s length approximation ratio be $\text{ar}(t) := |t|/\text{opt}_t(S)$, where by $\text{opt}_t(S)$ we denote the length of a shortest superstring for $S$. The second quality measure is $t$’s compression, which we want to maximize. Let $\|S\| := \sum_{i=1}^{n} |s^i|$ be the total length of all strings in the input $S$. We measure the running times of algorithms with respect to this input length. Now, given an input $S$, the compression of a superstring $t$ is $c(t) := \|S\| - |t|$. Intuitively, the compression achieved by $t$ is the number of letters we save compared to the trivial solution given by the concatenation of all strings in $S$. The optimal compression of $S$ is $\text{opt}_t(S) := \|S\| - \text{opt}_t(S)$. Furthermore, the compression approximation ratio of $t$ is $\text{ar}_c(t) := \text{opt}_t(S)/c(t)$. Finally, our definitions in Section 1.1.3 yield that an algorithm has approximation guarantee (or factor) $f \geq 1$ if it always computes a superstring with approximation ratio at most $f$ (in the considered measure).
Previous Work and Known Results  We mention some known results about worst-case efficient approximation algorithms for Shortest Common Superstring. To the author's knowledge, the best efficient algorithm in the length measure achieves factor 2.5 and is due to Sweedyk [Swe99]. However, the algorithm and its analysis are quite complicated. Thus, it would be an interesting result to prove that the following simple, well-known greedy algorithm has factor 2 in the length measure. This is the famous greedy superstring conjecture, independently stated by Tarhio and Ukkonen [TU88] and Turner [Tur89].

Algorithm 3 GreedySCS(S)

1: while $|S| > 1$ do
   Choose two strings $s, t \in S$ that maximize $|ov(s, t)|$.
   Remove $s$ and $t$ from $S$.
   Put $s \oplus t$ in $S$.  \(\triangleright\) Overlap $s$ and $t$ maximally, replace them by the result.
2: Output the only string remaining in $S$.

As mentioned, in both [TU88] and [Tur89] it was conjectured that GreedySCS has factor 2 in the length measure, but to the author’s knowledge, no proof has been found yet. They did prove that it has factor 2 in the compression measure, but it is easy to see that achieving some constant factor in one measure does not imply any constant factor in the other.

The following example shows that the greedy algorithm cannot achieve a constant factor smaller than 2 for length: For $k \in \mathbb{N}$, consider the input $S = \{s_1 = ba^{k+1}, s_2 = a^{k+1}b, s_3 = a^k\}$. In the first step, the greedy algorithm replaces $s_1$ and $s_2$ by $s_1 \oplus s_2 = ba^{k+1}b$, which cannot be overlapped (by more than zero letters) with $s_3$. Hence, greedy ends up with the solution $ba^{k+1}ba^k$. However, the optimal solution is simply $ba^{k+1}b$. The greedy approximation ratio is therefore $\frac{3 + 2k}{3 + k}$, which is $2 - o(1)$ for increasing $k$. On the positive side, Blum, Jiang, Li, Tromp, and Yannakakis [BJL94] showed that GreedySCS has factor 4 for length. Later, Kaplan and Shafrir[KS05] showed that it achieves factor 3.5. Hence, GreedySCS has a worst-case approximation ratio between 2 and 3.5 in the length measure.

Unfortunately, Shortest Common Superstring seems to resist arbitrarily good approximation, i.e., no polynomial time approximation scheme (PTAS) exists: Since the problem is maxSNP-hard (see [BJL94]) in both the length and the compression measure, it follows that for both measures, a constant $f > 1$ exists such that there is no efficient approximation algorithm with factor $f$, unless $P = NP$. In this context, Ott [Ott99] gave the first explicit lower bounds for the factors achievable by algorithms with polynomial worst-case running time: Unless $P = NP$, there is no algorithm with factor $1 + 1/17245 \approx 1.000058$ in the length measure, and no algorithm with factor smaller than $1 + 1/11216 \approx 1.000089$ for the
compression measure. Later, Vassilevska [Vas05] improved these lower bounds to $1 + 1/1216 \approx 1.00082$ for length and $1 + 1/1071 \approx 1.00093$ for compression.

**Our Results** We show that one can beat the above inapproximability bounds, i.e., achieve factor $1 + \varepsilon$ for every fixed $\varepsilon > 0$, in polynomial expected running time for random inputs from our models. We perform both a classical average-case analysis (in the Bernoulli and mixing model) and a probabilistic analysis with semi-random inputs in a perturbation model. Since we can guarantee approximation ratios arbitrarily close to 1 in polynomial expected time, we might say that we have a “probabilistic PTAS” for *Shortest Common Superstring*. Furthermore, we prove a sharp tail bound on the optimal compression in the Bernoulli and mixing model, which improves a previous result by Frieze and Szpankowski [FS96]. Our improved result may be of its own interest.

Let us say a few words about the general idea of our approximation scheme. As for *Independent Set* and *Coloring*, the idea is to test whether the greedy solution is sufficiently good, which here means, has approximation ratio $1 + \varepsilon$. In this case, we output the greedy solution. Otherwise, an optimal solution is found in exponential time. To find an optimal solution, we present an algorithm following a dynamic programming approach, which computes a shortest superstring of an input $S$ in time $O(2^n \cdot \text{poly}(||S||))$. (Remember that $n$ denotes the number of strings in $S$, while $||S||$ is the total length of $S$’s strings.) To certify that the greedy solution is sufficiently good, we efficiently upper bound the optimal compression of the random input. The upper bound is given by the value of a maximum spanning tree in an auxiliary graph. We prove a tail bound on the value of the maximum spanning tree for random inputs from our models. This yields the desired upper bound on the failure probability of the greedy step, and also the above-mentioned tail bound on the optimal compression.

Furthermore, remember that the worst-case approximation ratio that GreedySCS achieves in the length measure is between 2 and 4. We show that, however, for random inputs from our models, it produces with probability exponentially close to 1 a solution with approximation ratio $1 + \varepsilon$ for any fixed $\varepsilon > 0$.

To sum up, we might say that on one hand, we have algorithmic results for *Shortest Common Superstring*: The probabilistic approximation scheme, the optimal algorithm following dynamic programming, and the analysis of the greedy approximation ratio for random inputs. On the other hand, we have a structural result about *Shortest Common Superstring*, namely the tail bound on the optimal compression for random inputs. In the next section, we introduce our models of random inputs and state previous results. Then, in Section 1.3.2, we state our results in detail. Proofs are deferred to Chapter 4.
1.3.1 Random Input Models and Previous Results

In the average-case analysis, we consider two models of random strings, the so-called Bernoulli and mixing models. Both are described below and result from different assumptions on a generic model of random inputs which we introduce now.

Generally, to produce a random multiset $S$ of strings, we fix the alphabet $\Sigma$ and the lengths $|s|$ of $S$’s strings $s$ in some way. The individual strings $s$ are then generated by a stationary and ergodic stochastic process $\{X_k\}_{k=-\infty}^{\infty}$ with random letters $X_k \in \Sigma$. For indices $r \leq s \in \mathbb{Z}$, let $X^r_s := \{X_k\}_{k=r}^{s}$ be the subsequence of $\{X_k\}$ between $r$ and $s$. We refer the reader to Cover and Thomas [CT06] for an introduction to such processes and related notions. For us, two features of such a process are important. Firstly, due to the stationarity of the process, the probability that a fixed string appears in $\{X_k\}$, starting at position $r$, is independent of $r$. More formally, for every string length $l \in \mathbb{N}$, every string $x \in \Sigma^l$, and every pair $a, b \in \mathbb{Z}$ of starting positions,

$$\Pr[X_{a+l-1} = x_1 \ldots x_l] = \Pr[X_{b+l-1}^{b+l-1} = x_1 \ldots x_l].$$

(1.15)

It follows from (1.15), that for every $l \in \mathbb{N}$, we can define the $l$-th order probability distribution $P(l)$ of $\{X_k\}$, which is defined as

$$P(l)(x) := \Pr[X_i = x_i, 1 \leq i \leq l], \ x \in \Sigma^l.$$ 

Therefore, $P(l)(x)$ is the probability distribution of the length-$l$ strings in our process. For brevity, let $p(x) := P(1)(x), x \in \Sigma$, denote the probability distribution of the letters in the alphabet $\Sigma$ at a fixed position in the process. Furthermore, let

$$p_{\text{min}} := \min_{x \in \Sigma} p(x) \quad \text{and} \quad p_{\text{max}} := \max_{x \in \Sigma} p(x)$$

denote the minimum and maximum probability of a letter in $\Sigma$, respectively.

The second important feature is that using the so called entropy $H$ of such a process, we can measure the “amount of randomness” in the process. This is defined below.

To produce a string $s' \in S$ of our random input, we run our stochastic process $\{X_k\}$ and choose the random outcome of $X_1^{|s'|}$, i.e., we let $s' := X_1 \ldots X_{|s'|}$. The individual strings $s'$ are produced independently of each other.

The Bernoulli model As mentioned, the Bernoulli and mixing model result from additional assumptions on the generic model described above. In the Bernoulli model, we assume that the letters in our process are chosen independently. This model is often a good starting point and due to assuming independence of letters easier to analyze than other models. However, assuming independence is in many
1.3. Results for Shortest Common Superstring

applications unrealistic: Consider e.g. DNA sequencing as an application. Here, we clearly have dependencies between letters, even at greater distances. Therefore, we also consider the mixing model, which can incorporate limited dependencies between the letters in a random string.

The mixing model In the mixing model (see e.g. Łuczak and Szpankowski [ŁS94]), we assume that the stochastic process \( \{X_k\} \) has the mixing property, which we define now.

If we run our stochastic process \( \{X_k\} \), we get an infinite random string. In this random experiment, for indices \( r \leq s \), let \( \mathcal{F}_{r,s} \) be the set of all events regarding (only) the random subsequence \( X_{r,s} \) of \( \{X_k\} \). In other words, an event \( A \) is contained in \( \mathcal{F}_{r,s} \) if and only if its occurrence depends only on the values our random experiment assigns to \( X_{r,s} \). For example, assuming that \( \Sigma = \{0, 1\} \), the event \( "X_7 = 0 \) and \( X_9 = 1" \) is in \( \mathcal{F}_{10}^{10} \), but the event \( "X_{12} = 0" \) is not in \( \mathcal{F}_{10}^{10} \), since it refers to a letter outside \( X_{10}^{10} \). Notice that two events \( A \) and \( B \) referring to disjoint subsequences of \( \{X_k\} \) can nevertheless have a nonempty intersection \( A \cap B \). For example, the above events \( A = "X_7 = 0 \) and \( X_9 = 1" \) and \( B = "X_{12} = 0" \) are from \( \mathcal{F}_{7}^{10} \) and \( \mathcal{F}_{12}^{12} \), respectively, but their intersection is \( A \cap B = "X_2 = 0 \) and \( X_0 = 1 \) and \( X_{12} = 0" \).

Now, the mixing property is that a function \( \alpha : \mathbb{N} \rightarrow [0, 1) \) with \( \lim_{g \to \infty} \alpha(g) = 0 \) exists such that for every index \( m \in \mathbb{Z} \), every distance \( g \in \mathbb{N} \), and all events \( A \in \mathcal{F}_{m}^{-\infty} \) and \( B \in \mathcal{F}_{m+g}^{\infty} \),

\[
(1 - \alpha(g)) \Pr[A] \Pr[B] \leq \Pr[A \cap B] \leq (1 + \alpha(g)) \Pr[A] \Pr[B] .
\]

(1.16)

Since without any dependencies between the events \( A \) and \( B \) it would hold that \( \Pr[A \cap B] = \Pr[A] \Pr[B] \), (1.16) in other words requires that the events \( A \) and \( B \) are “almost independent,” i.e., their joint probability is approximately \( \Pr[A] \Pr[B] \). Here, \( \alpha(g) \) determines the maximum factor allowed for the deviation. Since \( \lim_{g \to \infty} \alpha(g) = 0 \), (1.16) says that the dependencies vanish with increasing distance of letters, and the function \( \alpha \) characterizes their quantity. Obviously, the mixing model generalizes the Bernoulli model (if there are no dependencies between the letters, clearly (1.16) is fulfilled).

Previous Results in the Bernoulli and Mixing Model In the Bernoulli model, the entropy \( H \) is defined as

\[
H = - \sum_{x \in \Sigma} p(x) \ln p(x) .
\]

(1.17)

As mentioned, informally, the entropy measures the “amount of randomness” in the process. (See [CT06] for detailed explanations.) Alexander [Ale94] analyzed the optimal compression for random inputs in the Bernoulli model. Roughly, the
result can be described as follows: A random input $S_n$ with $n$ strings is partitioned into a set $S^l_n$ of long strings (in some sense) and a set $S^s_n$ of short ones. For 

$$v_n = \frac{|S^l_n| \ln |S^l_n|}{H} + \sum_{s \in S^s_n} |s|,$$  \hspace{1cm} (1.18)

it is shown that

$$\lim_{n \to \infty} E \left[ \left| \frac{\text{opt}_c(S_n)}{v_n} - 1 \right| \right] = 0 .$$  \hspace{1cm} (1.19)

Therefore, a bit simplified, we might say that the expected optimal compression for a random input $S_n$ with $n$ strings in the Bernoulli model is approximately $v_n$ for $n$ large enough. An intuitive interpretation of $v_n$’s value given by (1.18) is the following: Compared to the trivial solution for $S_n$, the concatenation of its strings, the shortest superstring saves $\sum_{s \in S^s_n} |s|$ letters for the short strings and $\frac{|S^l_n| \ln |S^l_n|}{H}$ letters for the long ones. In other words, the short strings compress totally while the contribution of the long ones depends only on their number and is independent of their actual lengths.

Equation (1.19) together with the definition of $v_n$ in (1.18) directly yields that if all strings are long (i.e., $S^l_n = S_n$), we have

$$\frac{(1 - \varepsilon) \cdot n \ln n}{H} \leq E[\text{opt}_c(S_n)] \leq \frac{(1 + \varepsilon) \cdot n \ln n}{H},$$  \hspace{1cm} (1.20)

for all $\varepsilon > 0$ and $n$ large enough. Later, Frieze and Szpankowski [FS96] and Yang and Zhang [YZ99] considered the mixing model and the Bernoulli model, respectively, and analyzed the optimal compression and the one produced by some greedy algorithms. In the mixing model, the entropy $H$ is defined as

$$H = \lim_{l \to \infty} - \frac{E \left[ \ln P^{(l)}(X^l_1) \right]}{l} .$$  \hspace{1cm} (1.21)

We mentioned that the mixing model generalizes the Bernoulli model. So, assume for a moment that we have a Bernoulli model, which we can also interpret as a mixing model. Then, (1.21) and (1.17) (the entropy in the Bernoulli model) yield the same value: For $l = 1$, we have

$$E \left[ \ln P^{(1)}(X^1_1) \right] = \sum_{x \in \Sigma^1} \ln(P^{(1)}(x)) \cdot P^{(1)}(x)$$

$$= \sum_{x \in \Sigma} \ln(p(x)) \cdot p(x)$$

$$= -H_B ,$$
where by $H_B$ we denote the entropy as defined in the Bernoulli model. Using this, by induction on $l \geq 1$, it is easy to show that

$$E[\ln P_l(X_l)] = -l \cdot H_B,$$

which yields with (1.21) that the entropy $H_m$ as defined in the mixing model is

$$H_m = \lim_{l \to \infty} -\frac{E[\ln P_l(X_l)]}{l} = \lim_{l \to \infty} -\frac{-l \cdot H_B}{l} = H_B.$$

Frieze and Szpankowski showed in [FS96] that for a set $S_n$ of $n$ random strings in the mixing model, assuming a minimum string length of $\Omega(\log n)$, for every $\varepsilon > 0$,

$$\Pr \left( \frac{(1 - \varepsilon) \cdot n \ln n}{H} \leq \text{opt}_c(S_n) \leq \frac{(1 + \varepsilon) \cdot n \ln n}{H} \right) = 1 - o(1),$$  

(1.22)

and the same holds for GreedySCS’s compression. Notice that (1.22) together with (1.20) yields a concentration result on $\text{opt}_c(S_n)$ in the Bernoulli model, stating that it is concentrated around its expectation.

Consider the upper bound on $\text{opt}_c(S_n)$ in (1.22). To be more precise, in [FS96] it was shown that

$$\Pr \left( \text{opt}_c(S_n) > \frac{(1 + \varepsilon) \cdot n \ln n}{H} \right) = O(1/n^\varepsilon).$$  

(1.23)

One of our results improves the upper bound $O(1/n^\varepsilon)$ in (1.23) to an exponentially small, tight upper bound. This is our above-mentioned improved tail bound on the optimal compression in the Bernoulli and mixing model (see Section 1.3.2 for details).

The Perturbation Model We introduce our last model of random inputs, the perturbation (adversary) model. We describe a generic model, in which for each letter, an individual probability distribution can be specified. Then, we show that our generic model can actually simulate the process that an adversary chooses a set of arbitrary strings which are then subject to a small random perturbation.

To produce a random input $S = \{s^1, \ldots, s^n\}$, we do the following: We specify the lengths $|s^i|$ of the strings in some way, and for each letter $s^i_j$ in $S$’s strings $s^i$, we specify an individual probability distribution $p^i_j : \Sigma \to [0, 1]$. (We thus have $\sum_{i=1}^n |s^i|$ distributions.) Then, the letters $s^i_j$ are drawn independently according to
their distribution $p'_j$. Clearly, this model generalizes the Bernoulli model, where all letters share the same distribution $p$.

Without restrictions, i.e., bounding the allowed probabilities away from 1, deterministic worst case inputs can be specified. In this scenario, due to the inapproximability bounds for our problem, we cannot expect to achieve our desired results. We hence demand that all probabilities are bounded away from 1. To be more precise, for a probability distribution $p'_j$, let

$$(p'_j)_{\text{max}} := \max_{x \in \Sigma} p'_j(x),$$

and let

$$p_{\text{max}} := \max_{s'_j} (p'_j)_{\text{max}}$$

be the maximum probability of any letter in $\Sigma$ at any position in the strings. Now, the restriction in our model is that for a fixed $\hat{\varepsilon} \in (0, 1/2]$, it holds that $p_{\text{max}} \leq 1 - \hat{\varepsilon}$. Notice that $\hat{\varepsilon}$ limits the amount of random noise in our model to a certain minimal value.

We discuss why this model of random inputs actually can incorporate perturbation models, i.e., why we can model the process that an adversary chooses an arbitrary set of strings which are then perturbed by small random modifications. Assume e.g. that this process works as follows: Fix a mutation probability $p_m \in (0, 1/2]$. The adversary chooses an arbitrary set of strings $s'$. Then, we mutate every letter $s'_j$ in the strings $s'$ independently with probability $p_m$. In case a letter $s'_j$ is mutated, we do the following. Let $x'_j \in \Sigma$ be the original value of $s'_j$. Then, we choose the new value uniformly from $\Sigma \setminus \{x'_j\}$. Clearly, for every letter $s'_j$, the resulting probability distribution $q'_j$ of the finally chosen value $x \in \Sigma$ has the form

$$q'_j(x) = \begin{cases} 
1 - p_m & \text{if } x = x'_j \\
\frac{p_m}{|\Sigma| - 1} & \text{if } x \neq x'_j.
\end{cases}$$

We now choose the probability distributions in our model to simulate the above mutation process, i.e., we choose all distributions $p'_j = q'_j$. Obviously, we have the same probability distribution of strings. Furthermore, for $\hat{\varepsilon} := p_m$, it holds that $p_{\text{max}} \leq 1 - \hat{\varepsilon}$. Thus, the distributions $p'_j$ are legal in our model.

### 1.3.2 Our Results

For a set $S$ of strings, let

$$\Delta(S) := \max_{1 \leq i, j \leq n} |s'_i| - |s'_j| + 1$$

measure the maximum length difference of two strings in $S$. (The $+1$-term is used to simplify the calculations in our later proofs.) We now state our tail bound on the optimal compression in the Bernoulli and mixing model.
A Tail Bound on the Optimal Compression  The following lemma gives the above-mentioned tail bound on the optimal compression in the Bernoulli model, improving (1.23).

**Lemma 6.** Fix \( \varepsilon > 0 \) and a Bernoulli model with \( H/|\ln p_{\text{max}}| < 1 + \varepsilon \). For a random input \( S \) with \( \Delta(S) = \text{polylog}(n) \), it holds that

\[
\Pr\left[ \text{opt}_n(S) > \frac{(1 + \varepsilon) \cdot n \ln n}{H} \right] = \exp(-\Omega(n \log n)) .
\]

We discuss the assumptions and the statement of the lemma. First, consider the restriction \( H/|\ln p_{\text{max}}| < 1 + \varepsilon \) on the possible Bernoulli models, i.e., the possible probability distributions \( p(x) \) of the values \( x \in \Sigma \). For a Bernoulli model with a uniform distribution on \( \Sigma \), since \( p(x) = |\Sigma|^{-1} \) for all \( x \in \Sigma \), we have

\[
\frac{H}{|\ln p_{\text{max}}|} = -\sum_{x \in \Sigma} p(x) \ln p(x) = -\ln(|\Sigma|^{-1}) = 1 .
\]  

(1.24)

If we deviate from the uniform distribution, then \( H/|\ln p_{\text{max}}| > 1 \): Fix a non-uniform distribution \( p(x) \), and fix \( x^* \in \Sigma \) with \( p(x^*) = p_{\text{min}} \). Clearly, \( p(x^*) < p_{\text{max}} \). Therefore, by the properties of the natural logarithm,

\[
\frac{\ln p(x^*)}{\ln p_{\text{max}}} > 1 \quad \text{and} \quad \frac{\ln p(x)}{\ln p_{\text{max}}} \geq 1, \ x \neq x^* .
\]

Using this, by definition of the entropy \( H \), it follows that

\[
\frac{H}{|\ln p_{\text{max}}|} = -\sum_{x \in \Sigma} p(x) \ln p(x) |\ln p_{\text{max}}| = \sum_{x \in \Sigma} p(x) \ln p(x) \ln p_{\text{max}} 
\]

\[
= \frac{p(x^*) \ln p(x^*)}{\ln p_{\text{max}}} + \sum_{x \neq x^*} \frac{p(x) \ln p(x)}{\ln p_{\text{max}}} 
\]

\[
> p(x^*) \sum_{x \neq x^*} \frac{p(x)}{\ln p_{\text{max}}} = 1 .
\]

(1.25)

Furthermore, we show that for \( p_{\text{min}} \to 0 \) and \( p_{\text{max}} \to 1 \) (one can show that \( p_{\text{min}} \to 0 \) while \( p_{\text{max}} \) stays bounded away from 1 does not suffice, and \( p_{\text{max}} \to 1 \) while \( p_{\text{min}} \) stays bounded away from 0 is impossible), we can make \( H/|\ln p_{\text{max}}| \to \infty \). Since we demand \( H/|\ln p_{\text{max}}| < 1 + \varepsilon \), the lemma hence applies to distributions which are “close enough” to the uniform one, depending on the value of \( \varepsilon \).

Let us consider a distribution \( p \) for \( |\Sigma| = 2 \), with \( p_{\text{min}} = \alpha \) and \( p_{\text{max}} = 1 - \alpha \). To show that for \( \alpha \to 0 \), we have \( H/|\ln p_{\text{max}}| \to \infty \), observe that with (1.25), we get

\[
\frac{H}{|\ln p_{\text{max}}|} \geq \frac{p_{\text{min}} \ln p_{\text{min}}}{|\ln p_{\text{max}}|} = \frac{\alpha \ln \alpha}{\ln(1 - \alpha)} .
\]
Thus, if \( \lim_{\alpha \to 0} \frac{\alpha \ln \alpha}{\ln(1-\alpha)} = \infty \), then also \( \lim_{\alpha \to 0} \frac{\alpha}{\ln p_{\max}} = \infty \) as claimed. But the latter can be easily seen using the rule of L’Hospital.

We return to our discussion of Lemma 6. With respect to its statement, observe that the upper bound \( \exp(-\Omega(n \log n)) \) on \( \Pr[\text{opt}(S) > (1 + \varepsilon)(n \ln n)/H] \) can only be improved by a constant factor in the exponent: Clearly, the smallest nonzero probability that any event regarding a random input \( S \) can have is the probability that the value \( x^* \in \Sigma \) with \( p(x^*) = p_{\min} \) is assigned to all letters in \( S \)’s strings. For inputs with \( ||S|| = O(n \log n) \), the probability of this event is \( p_{\min}^{\text{polylog}(n)} = \exp(-O(n \log n)) \). Therefore, our bound is tight up to a constant factor in the exponent. Finally, notice that Lemma 6 makes no assumption on the minimum or total length of the strings.

We generalize Lemma 6 to the mixing model. In the mixing model, for a length \( l \in \mathbb{N} \), let

\[
p_{\text{max}}^l := \max_{x \in \Sigma^l} p_l(x)
\]

be the maximum probability of a string of length \( l \). In Chapter 4, we prove that

\[
p_{\text{max}}^l = \exp(-\Omega(l)). \tag{1.26}
\]

Therefore, a constant \( c > 0 \) with \( p_{\text{max}}^l \leq e^{-cl} \) exists. For such \( c \), we have \( |\ln p_{\text{max}}^l|/l \geq |\ln e^{-cl}|/l = c \). Thus, given a mixing model, we can define

\[
c_{\text{mix}} := \sup \left\{ c > 0 \mid \exists l_0 \in \mathbb{N} : \forall l \geq l_0 : |\ln p_{\text{max}}^l|/l \geq c \right\}. \tag{1.27}
\]

Assume for a moment (cf. our discussion following (1.21)) that our mixing model is in fact a Bernoulli model, i.e., there are no dependencies between the letters. Then, \( p_{\text{max}}^l \) is the probability that all \( l \) letters in a random string of length \( l \) are chosen the value from \( \Sigma \) with the maximum probability, denoted \( p_{\max} \) in the Bernoulli model. Consequently, \( p_{\text{max}} = (p_{\max})^l \), and in (1.27), we have

\[
|\ln p_{\text{max}}^l|/l = |\ln ((p_{\max})^l)/l = |l \cdot \ln p_{\max}| = |\ln p_{\max}|.
\]

In consequence, we get \( c_{\text{mix}} = |\ln p_{\max}| \) if in fact we have a Bernoulli model. In other words, the definition of \( c_{\text{mix}} \) generalizes the definition of \( |\ln p_{\max}| \) from the Bernoulli to the mixing model. In our mixing model results, \( c_{\text{mix}} \) therefore plays the role that \( |\ln p_{\max}| \) has in the Bernoulli model.

The following lemma is our generalization of Lemma 6 to the mixing model. It differs only in the use of \( c_{\text{mix}} \) instead of \( |\ln p_{\max}| \).

**Lemma 7.** Fix \( \varepsilon > 0 \) and a mixing model with \( H/c_{\text{mix}} < 1 + \varepsilon \). For a random input \( S \) with \( \Delta(S) = \text{polylog}(n) \), it holds that

\[
\Pr[\text{opt}(S) > (1 + \varepsilon) \cdot n \ln n] = \exp(-\Omega(n \log n)).
\]
The Approximation Scheme  In Chapter 4, we present an algorithm called ShortestSupStr. This algorithm computes a shortest superstring for a given set of strings using a dynamic programming approach. We prove the following lemma.

**Lemma 8.** For every input $S$ for **Shortest Common Superstring**, algorithm ShortestSupStr($S$) computes a shortest superstring in time $O(2^n \cdot \text{poly}(||S||))$.

Using ShortestSupStr as a subroutine, we present our probabilistic approximation scheme ApproxSCS($S, c$) for a set $S$ of strings and a rational number $c > 0$, which is used to adjust the algorithm to the desired approximation guarantee in the length measure. We prove the following theorem.

**Theorem 9.** Fix a Bernoulli or perturbation model, $\varepsilon \in (0, 1]$, and $k \in \mathbb{N}$. Set $c := 2(1 + k)/|\ln p_{\max}|$. Then, for a random input $S$ with $||S|| \geq (2c/\varepsilon)(n \ln n)$ and $\Delta(S) \leq n^k$, ApproxSCS($S, c$) has factor $1 + \varepsilon$ in the length measure and polynomial expected running time.

Notice that the theorem makes no assumption on the minimum length of the strings. The assumed $\Omega(n \log n)$ lower bound on $||S||$ only demands a $\Omega(\log n)$ average length. Furthermore, by choosing the parameter $k \in \mathbb{N}$ suitably, we can adapt to any polynomial maximum length difference $\Delta(S)$ of two strings. As for Lemma 6, we can generalize Theorem 9 to the mixing model. Again, $|\ln p_{\max}|$ is replaced by $c_{\max}$ in the transition from the Bernoulli to the mixing model.

**Theorem 10.** Fix a mixing model, $\varepsilon \in (0, 1]$, and $k \in \mathbb{N}$. Set $c := 2(1 + k)/c_{\max}$. Then, for a random input $S$ with $||S|| \geq (2c/\varepsilon)(n \ln n)$ and $\Delta(S) \leq n^k$, ApproxSCS($S, c$) has factor $1 + \varepsilon$ in the length measure and polynomial expected running time.

Finally, we can prove that in all of our models, GreedySCS with probability exponentially close to 1 achieves almost optimal approximation ratio in the length measure:

**Theorem 11.** Fix a Bernoulli or perturbation model with parameters as in Theorem 9 or fix a mixing model with parameters as in Theorem 10. Then, for a random input $S$, it holds that $\Pr[\text{ar}(\text{GreedySCS}(S)) \leq 1 + \varepsilon] \geq 1 - \exp(-\Omega(n \log n))$. 

Observe that as argued above for Lemma 6, the upper bound $\exp(-\Omega(n \log n))$ on the probability that GreedySCS fails to produce approximation ratio $1 + \varepsilon$ can only be improved by a constant factor in the exponent. Therefore, we again have in this sense a tight upper bound.

Above, we mentioned the famous greedy superstring conjecture, stating that GreedySCS has approximation guarantee 2 in the length measure. Using our results, we can at least show that among all inputs $S$ with strings of certain lengths,
there is at most an exponentially small fraction for which GreedySCS fails to produce length approximation ratio 2. (Of course, the greedy conjecture states that no such inputs exist at all.) This is expressed by the following corollary.

**Corollary 3.** Fix an alphabet $\Sigma$ and $k \in \mathbb{N}$. Let $\tilde{S}$ be the set of all inputs $S$ for shortest common superstring with $||S|| \geq (4(1 + k)/\ln |\Sigma|)(n \ln n)$ and $\Delta(S) \leq n^k$. Then, the fraction of inputs from $\tilde{S}$ for which GreedySCS fails to produce length approximation ratio 2 is at most $\exp(-\Omega(n \log n))$.

In the proof of the corollary, we show that its statement holds not only for the set $\tilde{S}$ of all inputs with $||S|| \geq (4(1 + k)/\ln |\Sigma|)(n \ln n)$ and $\Delta(S) \leq n^k$. In fact, we can fix any sequence $l_1, \ldots, l_n$ of $n$ string lengths $l_i$ obeying the two bounds and consider the set $\tilde{S}'$ of all inputs $S' = \{s^1, \ldots, s^n\}$ with $|s^i| = l_i$, $i = 1, \ldots, n$. For the sake of a clear presentation, however, the corollary is presented in the above way.

### 1.3.3 Further Notes on Previous and Related Work

We give some further notes on previous and related work, whose mentioning we deferred to this section, since above we wanted to motivate our results as clearly and concisely as possible.

Despite the hardness results on approximating shortest common superstring in polynomial time and the poor worst-case approximation ratio achieved by GreedySCS, the problem seems more tractable in practice. For example, Romero, Brizuela, and Tchernykh [RBT04] experimentally analyzed the approximation ratio achieved by some worst-case efficient algorithms for random inputs. In the experiments, different random input models were used, among them a Bernoulli model over an alphabet of size four and a model which randomly selects substrings from a real-world DNA sequence. Even though for the algorithms, only approximation guarantees of 3 and 4 were proved in the length measure, in the experiments the average approximation ratios were much better, sometimes as low as 1.014. Therefore, very good solution qualities were delivered by the algorithms on the average.

With respect to the greedy superstring conjecture, claiming that GreedySCS achieves factor 2 in the length measure, Weinard and Schnitger [WS06] achieved a partial success. They showed that if the input has a certain property, namely, causes the greedy algorithm to merge strings in a certain order, the greedy approximation ratio is at most 2. (Actually, a somewhat stronger result is proved, stating that the greedy superstring is longer than the shortest superstring by at most the length of a shortest cycle cover in an auxiliary graph, which itself always has at most the length of a shortest superstring.)

We note that Ma [Ma08] has performed a smoothed analysis showing that GreedySCS has factor $1 + o(1)$ in expectation. Hence, our analysis in the spirit
of smoothed analysis is not the first analysis in this style for Shortest Common Superstring. However, the approach in our probabilistic approximation scheme is different from the one followed by Ma: While in [Ma08], a worst-case efficient algorithm with \textit{expected} solution quality is studied, we demand \textit{guaranteed} approximation ratio $1 + \epsilon$ and polynomial running time only in expectation.

Above, we mentioned that currently, the best efficient approximation algorithm for Shortest Common Superstring in the length measure is due to Sweedyk [Swe99] and achieves factor 2.5. This result stands at the end of a long sequence of papers, in which algorithms with increasingly better approximation guarantee were presented. We mention some of these results. The first algorithm with a proven factor was presented by Li [Li90]. It achieves factor $O(\log n)$. As mentioned, Blum, Jiang, Li, Tromp, and Yannakakis [BJL+94] proved that GreedySCS achieves factor 4. They also gave a modified algorithm with factor 3. Later, Teng and Yao [TY93] presented an algorithm with factor 2.89, and Armen and Stein achieved factor $2 + \frac{\sqrt{2}}{5}$ in [AS94] and factor $2 + \frac{\sqrt{3}}{3}$ in [AS96].

1.4 Summary of Our Results

In this section, we briefly summarize the results stated in Sections 1.2 and 1.3, pointing out the commonalities between the different results for our three problems.

In addition to presenting average-case efficient approximation algorithms for Independent Set, Coloring, and Shortest Common Superstring in random and semi-random input models, we also analyze the performance of well-known greedy algorithms, i.e., algorithms with worst-case efficiency, for random inputs. Furthermore, we analyze some structural properties of random inputs for our problems. Hence, we can say that generally, our results fall into one of the following three categories:

\textbf{Average-Case Efficient Algorithms} We present approximation algorithms for Independent Set, Coloring, and Shortest Common Superstring, which guarantee to achieve certain approximation ratios and have polynomial expected running time for random and semi-random inputs from certain models. For Independent Set, we consider the $H_d(n, p)$ model of random uniform hypergraphs and the $\mathcal{G}(G, \epsilon)$ model of semi-random graphs. For Coloring, we perform our analysis in the $H_d(n, p)$ model. For Shortest Common Superstring, we consider the Bernoulli and mixing model as models of random inputs and a perturbation model as a model of semi-random inputs.

\textbf{Worst-Case Efficient Algorithms} We analyze the performance of greedy coloring for Independent Set and Coloring, and greedy superstring for Shortest
COMMON SUPERSTRING. These algorithms guarantee a polynomial running time, and we analyze their expected behavior.

For INDEPENDENT SET, we show that the greedy algorithm achieves a constant approximation ratio in expectation in both the $H_d(n, p)$ model and the $G(G, \varepsilon)$ model. For COLORING, we show this fact in the $H_d(n, p)$ model. We also give tail bounds, showing that certain approximation ratios are achieved with probability exponentially close to 1.

For SHORTEST COMMON SUPERSTRING, we show that in all considered models, the greedy algorithm achieves almost optimal length approximation ratio $1+\varepsilon$ with probability exponentially close to 1. This tail bound is tight up to a constant factor in the exponent.

**Structural Properties of Random Inputs** We analyze the behavior of some properties of random inputs for our problems. In the $G(G, \varepsilon)$ semi-random graph model, for a random graph $G$, we investigate the distribution of the largest eigenvalue $\lambda_1(A(G))$ of our auxiliary matrix $A(G)$. We determine the expected value of $\lambda_1$ and we also obtain a tail bound on $\lambda_1$, stating that it is “small” with high probability. Since $\lambda_1$ can be computed in polynomial time and since it upper bounds the independence number of $G$, one can efficiently compute an upper bound on $\alpha(G)$ which is small with high probability due to our results.

Also, we investigate the distribution of the optimal compression of a set of random strings in the Bernoulli and mixing model and a perturbation model. For the optimal compression, we obtain an exponentially small upper bound on the probability that it exceeds $1+\varepsilon$ times its expectation. Again, the bound is tight up to a constant factor in the exponent.

To conclude, our analyses yield an important insight into either the behavior of an algorithm for “typical” instances of a problem, or into the structural properties of such inputs.
1.5 Bibliographical Notes

We give some bibliographical notes on the results presented in this thesis.

- The results on approximating INDEPENDENT SET and COLORING in random uniform hypergraphs, as given in Section 1.2.1 and proved in Chapter 2, were presented in a preliminary version at a conference:


  Furthermore, a journal version [Plo10] is submitted to a journal.

- The results on approximating SHORTEST COMMON SUPERSTRING, as given in Section 1.3 and proved in Chapter 4, were presented in preliminary form at a conference:

Chapter 2

Independent Set and Coloring for Random Hypergraphs

Contents

2.1 Greedy Coloring of Random Hypergraphs .......................... 49
2.2 Approximating the Independence Number ......................... 53
2.3 Approximating the Chromatic Number .............................. 61
2.4 The Expected Behavior of Greedy Independent Set and Coloring 64
2.5 Small Edge Probabilities ............................................. 68
2.6 Completing the Proof of Lemma 2 ................................. 73
2.7 Conclusions and Open Problems ................................. 75

In this chapter, we prove our results for approximating INDEPENDENT SET and COLORING in random uniform hypergraphs, as stated in Section 1.2.1. Since the analyses of our algorithms rely on the tail bounds for the size of the greedy independent set (Lemma 1) and the number of colors used by greedy coloring (Lemma 2), we start by proving these results in Section 2.1. Sections 2.2 and 2.3 contain our approximation algorithms for INDEPENDENT SET and COLORING, respectively, together with their analyses. In Section 2.4, we prove Corollary 1, giving our analysis of the expected behavior of the greedy independent set and coloring for random uniform hypergraphs. In Section 2.5, we prove Theorem 6, which gives our analysis of algorithm SmallEdgePrIndSet, which we can use to approximate INDEPENDENT SET for very small edge probabilities, below the ones we can handle with our other algorithms. Section 2.6 contains a proof of two inequalities claimed in the proof of Lemma 2. The proof of the inequalities is deferred to Section 2.6 since it is technical and lengthy. Finally, in Section 2.7, we draw conclusions and discuss some open problems.

2.1 Greedy Coloring of Random Hypergraphs

In this section, we perform our analysis of the greedy independent set and greedy coloring for random hypergraphs drawn from $H_d(n, p)$. We start with our tail bound
on the greedy independent set given in Lemma 1, which we restate here for conve-
nience.

Fix an integer \( d \geq 2 \) and \( \varepsilon > 0 \). There is a constant \( c(d, \varepsilon) > 0 \) such that for probability \( p = p(n) \) with \( c(d, \varepsilon) \cdot (\ln n)^d/n^{d-1-\varepsilon} \leq p \leq 3/4 \), the following holds. Let \( C \) be the coloring computed by \( \text{GreedyColor}(H) \) for a random hypergraph \( H \) from \( H_d(n, p) \). Then,

\[
\Pr\left[ |\text{big}(C)| \leq ((d - 2)! \cdot \varepsilon \cdot (\ln n)/(2p))^{1/(d-1)} \right] \leq e^{-n \ln n}.
\]

**Proof of Lemma 1.** We set the constant \( c(d, \varepsilon) \) in the lemma to

\[
c(d, \varepsilon) := (d - 2)! \cdot 8^{d-1} \cdot \varepsilon/2.
\]

Furthermore, we abbreviate the threshold for \( |\text{big}(C)| \) by

\[
s := ((d - 2)! \cdot \varepsilon \cdot (\ln n)/(2p))^{1/(d-1)},
\]

and set \( t := n/(2s) \). Let \( \mathcal{B} \) denote the bad event that \( |\text{big}(C)| \leq s \). Then, the lemma states that \( \Pr[\mathcal{B}] \leq e^{-n \ln n} \). If \( \mathcal{B} \) happens, there are at least \( n/s > t \) color classes in the coloring \( C \). In this case, let \( C^* := \{C_1, \ldots, C_t\} \) contain the first \( t \) color classes of \( C \).

We call a set \( D = \{D_1, \ldots, D_t\} \) of pairwise disjoint classes \( D_i \subseteq V \) with \( |D_i| \leq s \) for \( i = 1, \ldots, t \) a partial vertex coloring. A vertex \( v \) is bad if for every class \( D_i \in D \), there is an edge \( e \in E \) with \( v \in e \) and \( e \setminus \{v\} \subseteq D_i \). We call \( D \) bad if every vertex \( v \in (V \setminus (D_1 \cup \ldots \cup D_t)) =: \overline{D} \) is bad.

Consider the set \( C^* \) in case that \( \mathcal{B} \) happens. All classes in \( C^* \) are of size at most \( s \). Furthermore, every vertex \( v \in C^* \) is not assigned to any of the classes in \( C^* \) by algorithm GreedyColor. Thus, for every such vertex and every class \( C_i \in C^* \), there has to be an edge \( e \in E \) with \( v \in e \) and \( e \setminus \{v\} \subseteq C_i \). We conclude that if \( \mathcal{B} \) happens, \( C^* \) is a bad partial vertex coloring and \( \Pr[\mathcal{B}] \leq \Pr[\exists D \in \overline{D} : D \text{ is bad}] \) for the set \( \overline{D} \) of all partial vertex colorings.

We estimate the probability that a fixed partial vertex coloring \( D \in \overline{D} \) is bad. The probability that a fixed vertex \( v \in \overline{D} \) is bad equals

\[
\prod_{i=1}^{t} \left( 1 - (1 - p)^{|D_i|/(|E|)} \right),
\]

since \( (1 - p)^{|D_i|/(|E|)} \) is the probability that there is no edge \( e \in E \) with \( v \in e \) and \( e \setminus \{v\} \subseteq D_i \). Since \( |D_i| \leq s \) for all \( D_i \in D \), we obtain \( |\overline{D}| \geq n - t \cdot s = n/2 \). With
2.1. Greedy Coloring of Random Hypergraphs

\[ 1 + x \leq e^x \text{ for } x \in \mathbb{R} \text{ and } \binom{n}{b} \leq \frac{a^b}{b!} \text{ for } a, b \in \mathbb{N}, \]

we infer that

\[
\Pr[D \text{ is bad}] = \prod_{v \in D} \Pr[v \text{ is bad}] \leq \left( \prod_{i=1}^{t} \left( 1 - (1 - p)^{|D_i|} \right) \right)^{n/2} \]

\[
\leq \exp \left( -\frac{n}{2} \cdot \sum_{i=1}^{t} (1 - p)^{|D_i|} \right) \]

\[
\leq \exp \left( -\frac{n}{2} \cdot \sum_{i=1}^{t} (1 - p)^{|D_i|} \right) \]

\[
\leq \exp \left( -\frac{tn}{2} \cdot n^{-\varepsilon/(d-1)} \right), \quad (2.1)
\]

again using \(|D_i| \leq s\) for all \(D_i\). To get (2.1), observe the following: With \(1 - x \geq e^{-2x}\) for \(0 \leq x \leq 3/4\) and \(0 \leq p \leq 3/4\), by choice of \(s\) we have

\[
s \leq ((d - 2)! \cdot e \cdot (\ln n)/(2p))^1/(d-1) \]

\[
\Rightarrow \exp \left( -2ps^{-1}/((d - 1)!) \right) \geq n^{-\varepsilon/(d-1)} \]

\[
\Rightarrow (1 - p)^{s^{-1}/((d - 1)!)} \geq n^{-\varepsilon/(d-1)} .
\]

Since there are at most \(\sum_{i=1}^{s} \binom{n}{i}\) possible choices for each of the \(t\) color classes of a partial vertex coloring \(D \in \tilde{D}\), (2.1) yields

\[
\Pr[\mathcal{B}] \leq \Pr[\exists D \in \tilde{D} : D \text{ is bad}] \leq \sum_{D \in \tilde{D}} \Pr[D \text{ is bad}] \leq \left( \sum_{i=1}^{s} \binom{n}{i} \right)^{t} \cdot \exp \left( -\frac{tn}{2} \cdot n^{-\varepsilon/(d-1)} \right) \]

\[
\leq n^{(s+1)t} \cdot \exp \left( -\frac{tn}{2} \cdot n^{-\varepsilon/(d-1)} \right), \quad (2.2)
\]

using that \(\sum_{i=1}^{s} \binom{n}{i} \leq n^{s+1}\) for \(s \leq (n - 1)/2\). The latter holds by choice of \(p\). Notice that in (2.2), we have \(n^{(s+1)t} = n^{(s+1)n/(2s)} \leq n^n = e^{n \ln n}\). Furthermore, our choice of \(p\) in the lemma is equivalent to \((tn/2) \cdot n^{-\varepsilon/(d-1)} \geq 2n \ln n\). Therefore, with (2.2), we get

\[
\Pr[|\text{big} (C)| \leq s] = \Pr[\mathcal{B}] \leq n^{(s+1)t} \cdot \exp \left( -\frac{tn}{2} \cdot n^{-\varepsilon/(d-1)} \right) \]

\[
\leq e^{(n \ln n) - 2n \ln n} = e^{-n \ln n} ,
\]

which completes our proof. \(\square\)
Next, we give the proof for Lemma 2, giving the tail bound on the number of color classes used by GreedyColor. The lemma states the following.

Fix an integer \( d \geq 2 \) and \( \varepsilon > 0 \). Let \( p = p(n) \) be a probability with
\[
\frac{4d^4 \varepsilon \cdot (\ln n)^{2d^4}}{n^{d+1-\varepsilon d^4/n}} \leq p \leq \frac{3}{4} ,
\]
and let \( C \) be the coloring computed by GreedyColor(\( H \)) for a random hypergraph \( H \) from \( H_d(n, p) \). Then, for the number \( |C| \) of classes in \( C \), we have
\[
\Pr\left[ |C| \geq 2n \cdot (p/(\varepsilon \cdot \ln n))^{1/(d-1)} \right] < e^{-\ln n}.
\]

**Proof of Lemma 2.** Let \( k_0 := n \cdot (p/(\varepsilon \cdot \ln n))^{1/(d-1)} \). The lemma states that \( \Pr[|C| \geq 2k_0] < e^{-\ln n} \). For \((k-1)(d-1)+1 \leq j \leq n\), let \( B_j^k \) denote the event that vertex \( j \) gets color \( k \), i.e., it is assigned to class \( C_k \) (obviously, a vertex \( j \leq (k-1)(d-1) \) cannot get color \( k \)). Furthermore, let \( A_j^k \) be the event that for coloring the first \( j \) vertices, the algorithm uses at least \( k \) colors. Finally, let \( A_k \) be the event "\(|C| \geq k\)". We estimate the conditional probability \( \Pr[A_k^k | A_k^k] \). Since \( A_k^k = \bigcup_{j=k(d-1)+1}^n B_j^k \), we infer that
\[
\Pr[A_k^k | A_k^k] \leq \sum_{j=k(d-1)+1}^n \Pr[B_j^k | A_k^k] .
\]
Furthermore, since \( B_j^k \subseteq A_{j-1}^k \subseteq A_k^k \), it follows that \( \Pr[B_j^k | A_k^k] \leq \Pr[B_j^k | A_{j-1}^k] \), and hence,
\[
\Pr[A_k^k | A_k^k] \leq \sum_{j=k(d-1)+1}^n \Pr[B_j^k | A_{j-1}^k] . \tag{2.3}
\]
We estimate \( \Pr[B_j^k | A_{j-1}^k] \) for fixed \( j \). Vertex \( j \) gets color \( k+1 \) only if for all classes \( C_i, 1 \leq i \leq k \), which exist at the time of coloring \( j \) since \( A_{j-1}^k \) occurs, there is an edge connecting \( j \) with \( d-1 \) vertices in \( C_i \). Therefore,
\[
\Pr[B_j^k | A_{j-1}^k] \leq \prod_{i=1}^k \left( 1 - (1 - p)^{C_i^j} \right)
\]
\[
\leq \exp \left( - \sum_{i=1}^k (1 - p)^{C_i^j} \right)
\]
\[
\leq \exp \left( - \sum_{i=1}^k (1 - p)^{(C_i^j)^{(d-1)}/((d-1)!)} \right)
\]
\[
= \exp(-\sigma) , \tag{2.4}
\]
where we used
\[
\sigma := \sum_{i=1}^{k} (1 - p)^{(d-1)/(d-1)} .
\]  
(2.5)

For the range of \( p \) in the lemma, it holds that
\[
k_0 \sigma / 2 \geq n \ln n \quad \text{and} \quad \sigma / 2 \geq \ln n .
\]  
(2.6)

The proof of (2.6) is deferred to Section 2.6. (The hard part is proving the left-hand side inequality, the right-hand side one then follows easily since \( k_0 = o(n) \).) Using \( \ln n \leq \sigma / 2 \) from (2.6), with (2.3) and (2.4) we get
\[
\Pr[|A| = k_0 + 1] \leq n \cdot ((d-2)! \cdot \ln n / (2p))^{1/(d-1)}
\]
and
\[
\Pr[|B| = k_0 + 1] \leq 2(2k_0) \cdot \exp(-\sigma / 2) \leq \exp(-n \ln n) ,
\]
which completes our proof.

\[ \square \]

### 2.2 Approximating the Independence Number

Lemma 1 immediately yields the following approximation algorithm for \textsc{Independent Set} with approximation guarantee \( O(n \cdot (p/ \log n)^{1/(d-1)}) \) and polynomial expected running time for random inputs from \( H_d(n, p) \). Notice that in addition to the random hypergraph \( H \) from \( H_d(n, p) \), it is given the edge probability \( p \) used to create \( H \), and a parameter \( \varepsilon > 0 \) which is used to adapt the algorithm to different lower bounds for the possible edge probabilities \( p \).

**Algorithm 4** \textsc{ApproxIndSet}(\(H, p, \varepsilon\))

1. Run \textsc{GreedyColor}(\(H\)) and let \( C \) be the coloring it computes.
2. If \( |\text{big}(C)| > ((d-2)! \cdot \varepsilon \cdot (\log n) / (2p))^{1/(d-1)} \) then output \text{big}(C).
3. Find a largest independent set by exhaustive search and output it.
Theorem 1 gives our analysis of \textsc{ApproxIndSet}. We restate it here for convenience.

\begin{quote}
Fix an integer \(d \geq 2\) and \(\varepsilon > 0\). Let \(p = p(n)\) be a probability with \(1/n^{d-1-2\varepsilon} \leq p \leq 3/4\). Then, \textsc{ApproxIndSet}(H, p, \varepsilon) has approximation guarantee \(O(n \cdot (p/\log n)^{1/(d-1)})\) and polynomial expected running time for random inputs \(H\) from \(H_d(n, p)\).
\end{quote}

\textbf{Proof of Theorem 1.} We start with the expected running time of the algorithm and show that each step’s expected running time, the product of its effort and its execution probability, is polynomial.

Steps 1–2 have polynomial effort and hence also polynomial expected running time, since algorithm \textsc{GreedyColor} has polynomial worst-case running time: For \(n - 1\) vertices \(v\), it checks for at most \(n\) color classes \(C_i\) whether there is an edge among the \(O(n^d)\) edges of \(H\) which consists of \(v\) and \(d - 1\) vertices in \(C_i\). Thus, \textsc{GreedyColor} performs a polynomial amount of \(O(n^{d+2})\) tests, each of which clearly can be performed in polynomial time.

In Step 3, we test all \(2^n\) subsets \(V' \subset V\) for independence and output the largest independent set \(V'\) found. Therefore, Step 3 has effort \(O(poly(n) \cdot 2^n)\). Since Step 3 is executed only if \(|\text{big}(C)| \leq ((d - 2)! \cdot \varepsilon \cdot (\ln n)/(2p))^{1/(d-1)}\), Lemma 1 yields an execution probability of at most \(e^{-n \ln n} \leq 2^{-n}\), and thus the expected running time of Step 3 is \(O(poly(n) \cdot 2^n \cdot 2^{-n}) = poly(n)\). Lemma 1 is applicable, since our choice of \(1/n^{d-1-2\varepsilon} \leq p \leq 3/4\) in the theorem implies that it is legal with respect to Lemma 1, i.e., \(c(d, \varepsilon) \cdot (\ln n)^d/n^{d-1-\varepsilon} \leq p \leq 3/4\) for the constant \(c(d, \varepsilon)\) in the lemma.

We turn to the approximation guarantee and show that in any case, we output a solution with approximation ratio \(O(n \cdot (p/\log n)^{1/(d-1)})\). Step 3’s optimal solution has approximation ratio 1. If Step 2 outputs \text{big}(C),

\[|\text{big}(C)| > ((d - 2)! \cdot \varepsilon \cdot (\ln n)/(2p))^{1/(d-1)} = \Omega\left(\left((\log n)/p\right)^{1/(d-1)}\right).\]

Trivially, \(\alpha(H) \leq n\). Thus, the approximation ratio is

\[
\frac{\alpha(H)}{|\text{big}(C)|} = O\left(\frac{n}{(\log n)/p}^{1/(d-1)}\right) = O(n \cdot (p/\log n)^{1/(d-1)}),
\]

which finishes our proof of the approximation guarantee. \(\square\)

\textbf{Improving the Approximation Guarantee} In the following, we improve the approximation guarantee of algorithm \textsc{ApproxIndSet} for a smaller range of edge probabilities \(p\), i.e., for \(p = \Omega((\log n)/n^{1-1/d})\). This is achieved by adding some steps which try to certify that \(\alpha(H) = O((n/p)^{1/2})\). Then, we can use \(O((n/p)^{1/2})\) instead of the trivial upper bound \(n\) on \(\alpha(H)\) in the analysis. By choice of \(p\), it
holds that \((n/p)^{1/2} = o(n)\), so in consequence this improves the approximation guarantee. The resulting algorithm ImprovedIndSet (Algorithm 5) is very similar to the corresponding one in [KV02]. We start with some definitions used in the algorithm and its analysis, together with some statements regarding the introduced notions.

**Definition 4.** For a \(d\)-uniform hypergraph \(H = (V, E)\), \(d \geq 2\), and a set \(S \subseteq V\) with \(|S| = d - 2\), the projection graph of \(H\) on \(S\) is the graph \(G_{\text{pro}}(H, S) := (V \setminus S, E')\), such that \([u, w] \in E'\) if and only if \((S \cup \{u, w\}) \in E\).

Algorithm ImprovedIndSet exploits the following lemma, stating that we can upper bound the independence number of a hypergraph by upper bounding the independence numbers of its projection graphs.

**Lemma 9.** For every \(d\)-uniform hypergraph \(H = (V, E)\), \(d \geq 2\), we have

\[
\alpha(H) \leq \max_{S \subseteq V, |S| = d-2} \alpha(G_{\text{pro}}(H, S)) + d - 2.
\]

As an example for the above notions, consider Figure 2.1. It shows a 3-uniform hypergraph \(H = (V, E)\) with \(V = \{1, \ldots, 5\}\) and three edges \(\{1, 2, 3\}\), \(\{2, 3, 4\}\), and \(\{1, 4, 5\}\). Since \(d = 3\), for this hypergraph, the maximum in (2.7) is taken over the five sets \(S = \{1\}, \ldots, \{5\}\) containing one vertex each. The resulting projection graphs \(G_{\text{pro}}(H, \{i\})\), \(i = 1, \ldots, 5\), are also shown in Figure 2.1.

![Figure 2.1: A 3-uniform Hypergraph with its Projection Graphs.](image)

**Proof of Lemma 9.** Fix a \(d\)-uniform hypergraph \(H\). First, we prove that for every subset \(S \subseteq V\) with \(|S| = d - 2\), the size of a largest independent set \(I \subseteq V\) such
that \( S \subseteq I \) is at most \( \alpha(G_{\text{pro}}(H, S)) + d - 2 \): Fix \( S \) and let \( I \) be such an independent set. Clearly, \( I \setminus S \) is independent in the projection graph \( G_{\text{pro}}(H, S) \), yielding \( \alpha(G_{\text{pro}}(H, S)) \geq |I \setminus S| \), or equivalently, \( |I| \leq \alpha(G_{\text{pro}}(H, S)) + d - 2 \) as claimed.

Now, fix a maximum independent set \( I^* \) in \( H \). Clearly, \( |I^*| \geq d - 1 \), and we can choose some \( S^* \subseteq I^* \) with \( |S^*| = d - 2 \). Then,

\[
\alpha(H) = |I^*| \leq \alpha(G_{\text{pro}}(H, S^*)) + d - 2 \leq \max_{S \subseteq V, |S| = d - 2} \alpha(G_{\text{pro}}(H, S)) + d - 2 ,
\]

which finishes our proof. \( \square \)

Consider Lemma 9 for a graph \( H \), i.e., the case \( d = 2 \). Since a set \( S \subseteq V \) has cardinality \( d - 2 \) if and only if \( S = \emptyset \), there is only one projection graph \( G_{\text{pro}}(H, S) \) considered in the maximum in (2.7), namely \( G_{\text{pro}}(H, \emptyset) \). Furthermore, \( G_{\text{pro}}(H, \emptyset) \) by definition simply is \( H \) itself.

**Definition 5.** For a hypergraph \( H = (V, E) \) and \( S \subseteq V \), let the non-neighborhood of \( S \) be \( \overline{N}(S) := \{ v \in V \setminus S : \text{there is no edge } (\{v\} \cup T) \text{ in } E \text{ with } T \subseteq S \} \).

In words, given a hypergraph \( H = (V, E) \), the non-neighborhood of a set \( S \subseteq V \) contains exactly the vertices \( v \) outside of \( S \) for which there is no edge in \( E \) that connects \( v \) with some vertices exclusively in \( S \). As an example, consider Figure 2.2. It shows the above 3-uniform hypergraph. For \( S = \{1, 2\} \), we have \( \overline{N}(S) = \{4, 5\} \), since for these two vertices, there is no edge that contains, besides the vertex itself, only vertices from \( S \). For the vertex 3, this is not the case (since \( \{1, 2, 3\} \) is such an edge).

![Figure 2.2: A 3-uniform Hypergraph with a Set S and its Non-Neighborhood.](image)

Again, the following lemma is implicitly used by algorithm \texttt{ImprovedIndSet} to upper bound the independence number of the given hypergraph.

**Lemma 10.** For every hypergraph \( H = (V, E) \) and all \( a, b \in \mathbb{N} \), the following holds: If for all sets \( S \subseteq V \) with \( |S| = a \) it holds that \( |\overline{N}(S)| \leq b \), then \( \alpha(H) \leq a + b \).
2.2. Approximating the Independence Number

Proof. Fix a hypergraph $H$ and $a, b \in \mathbb{N}$, and assume that $\alpha(H) > a + b$. Then, we can choose an independent set $I \subseteq V$ with $|I| > a + b$, and then fix a set $S \subseteq I$ with $|S| = a$. Clearly, for each vertex $v \in I \setminus S$, there is no edge connecting $v$ and some vertices exclusively from $S$, since $I$ is independent. Thus, $|\overline{N}(S)| \geq |I \setminus S| = (a+b) - a = b$. Therefore, not for all subsets $S$ with $|S| = a$, we have $|\overline{N}(S)| \leq b$. $\square$

We are now ready to present algorithm ImprovedIndSet, improving on the approximation guarantee of ApproxIndSet for high enough edge probabilities.

Algorithm 5 ImprovedIndSet$(H, p)$

1: Run GreedyColor$(H)$ and let $C$ be the coloring it computes. if $|\text{big}(C)| \leq ((d - 2)! \cdot (\ln n)/(6p))^{1/(d-1)}$ then go to Step 5.
2: For all $S \subseteq V$, $|S| = d - 2$, compute $\lambda_1(M(G_{pro}(H, S), p))$. Let $m$ be the maximum of the computed eigenvalues. if $m \leq 4 \cdot (n/p)^{1/2}$ then output big($C$).
3: Set $s' := (d - 1) \cdot (4 \cdot (\ln n)/p)^{1/(d-1)}$. For all $S' \subseteq V$, $|S'| = s'$, compute $|\overline{N}(S')|$. if $|\overline{N}(S')| \leq (n/p)^{1/2}$ for all tested subsets $S'$ then output big($C$).
4: Check all subsets $S'' \subseteq V$ with $|S''| = 2 \cdot (n/p)^{1/2}$. if none of them is independent then output big($C$).
5: Find a largest independent set by exhaustive search and output it.

Step 2 implicitly exploits (2.7) by upper bounding $\alpha(G_{pro}(H, S))$ for all sets $S$ with $|S| = d - 2$, using the largest eigenvalue of $M(G_{pro}(H, S), p)$ as an upper bound. (Confer Lemma 3.) We discuss the case that the algorithm is run on a random graph, i.e., we have $d = 2$. Here, only $G_{pro}(H, \emptyset)$ is considered in Step 2, which as argued above is simply the graph $H$ itself. Consequently, Step 2 is except parametrization the same as the corresponding step in the algorithm presented by Krivelevich and Vu in [KV02]. The same holds for the other steps. (For Step 3, in case $d = 2$, we mention that our definition of the non-neighborhood matches the one in [KV02].) For random graphs, our algorithm is except parametrization therefore the same as the one presented by Krivelevich and Vu in [KV02], and we can say that we actually extended the latter from $G(n, p)$ to $H_r(n, p)$.

We prove Theorem 2, restated below, which gives our analysis of ImprovedIndSet.

Fix an integer $d \geq 2$. There is a constant $c(d) > 0$ such that for probability $p = p(n)$ with $c(d) \cdot (\ln n)/n^{1-1/d} \leq p \leq 3/4$, ImprovedIndSet$(H, p)$ has approximation guarantee $O(n^{1/2} \cdot p^{-(d-3)/(2d-2)} / (\log n)^{1/(d-1)})$ and polynomial expected running time for random inputs $H$ from $H_r(n, p)$.

Proof of Theorem 2. We start with the approximation guarantee. Step 5 outputs an optimal solution and hence achieves approximation ratio 1. If big($C$) is output in
any other step, its size is

\[ |\text{big}(C)| > ((d - 2)! \cdot (\ln n)/(6p))^{1/(d-1)} \]

since otherwise Step 1 switches to exhaustive search. In case Step 2 outputs \text{big}(C), we have

\[ m = \max_{S \subseteq V, |S| = d - 2} \lambda_1(M(G_{\text{pro}}(H, S), p)) \leq 4 \cdot (n/p)^{1/2}. \]

Lemmas 3 and 9 yield that for a fixed set \( S \), \( \alpha(H) \leq 4 \cdot (n/p)^{1/2} + d - 2 = O((n/p)^{1/2}). \) Thus, if we output \text{big}(C) in Step 2, its approximation ratio is

\[ \frac{\alpha(H)}{|\text{big}(C)|} = O\left(\frac{(n/p)^{1/2}}{(\ln n)/p^{1/(d-1)}}\right) = O\left(n^{1/2} \cdot p^{-(d-3)/(2d-2)}/(\ln n)^{1/(d-1)}\right) \]

as claimed in the theorem. If we output \text{big}(C) in Step 3, all subsets \( S' \subseteq V \) with \( |S'| = (d - 1) \cdot (4 \cdot (\ln n)/p)^{1/(d-1)} \) have a non-neighborhood with at most \((n/p)^{1/2}\) vertices. Thus, Lemma 10 yields

\[ \alpha(H) \leq (d - 1) \cdot (4 \cdot (\ln n)/p)^{1/(d-1)} + (n/p)^{1/2} = O((n/p)^{1/2}) \]

the same upper bound on \( \alpha(H) \) as in Step 2, so again approximation ratio \( O(n^{1/2} \cdot p^{-(d-3)/(2d-2)}/(\ln n)^{1/(d-1)}) \) follows. Finally, if Step 4 outputs \text{big}(C), clearly \( \alpha(H) \leq (n/p)^{1/2}. \) Again, the same approximation ratio follows.

**The Expected Running Time** We turn to the running time. Let \( T_i \) be the random variable for the time spent in Step \( i \) and \( F_i \) be the probability that it fails. We show that \( E[T_i] \) is polynomial for all steps \( i = 1, \ldots, 5 \). Steps 1–2 even have polynomial effort: Step 2 computes the largest eigenvalue of \( \binom{n}{d-2} \leq n^{d-2} \) matrices, and each computation runs in polynomial time as mentioned in Section 1.2.1.

Step 3 is executed only if \( F_2 \) occurs, i.e., among the \( \binom{n}{d-2} \) sets \( S \) considered in Step 2, there is one with \( \lambda_1(M(G_{\text{pro}}(H, S), p)) > 4 \cdot (n/p)^{1/2}. \) Since in \( H \), the edges of cardinality \( d \) are chosen independently with probability \( p \), the graphs \( G_{\text{pro}}(H, S) \) are random graphs according to \( H_2(n - (d - 2), p) = G(n - (d - 2), p). \) Thus, Lemma 3 yields that for a fixed set \( S \),

\[ \Pr\left[ \lambda_1(M(G_{\text{pro}}(H, S), p)) > 4 \cdot (n/p)^{1/2} \right] \leq 2^{-(n-(d-2))p/8}. \]

Therefore, with a union bound over all tested sets, Step 3’s execution probability is at most

\[ \Pr[F_2] = \Pr\left[ \exists S \subseteq V, |S| = d - 2: \lambda_1(M(G_{\text{pro}}(H, S), p)) > 4 \cdot (n/p)^{1/2} \right] \leq \binom{n}{d-2} \cdot 2^{-(n-(d-2))p/8} \leq n^{d-2} \cdot 2^{-np/9} = \text{poly}(n) \cdot 2^{-np/9}. \]  

(2.8)
Step 3’s effort is $O\left(\text{poly}(n) \cdot \left(\binom{n}{s} \right)\right)$, since it tests $\binom{n}{s}$ subsets. Since $\left(\binom{n}{s} \right) \leq n^{s^{'}} = 2^{s^{'-\log n}}$, with (2.8) its expected running time is

$$
E[T_3] = O\left(\text{poly}(n) \cdot \left(\binom{n}{s} \right) \cdot 2^{-np/9}\right) = O\left(\text{poly}(n) \cdot 2^{(s^{'-\log n})-np/9}\right) = O\left(\text{poly}(n) \cdot 2^{(d-1)\left(\frac{1}{4}(\ln n)/p\right)^{1/(d-1)}(\log n)-np/9}\right). \quad (2.9)
$$

We set the constant $c(d)$ in the theorem to $c(d) := (9(d-1) \cdot 4^{1/(d-1)}/\ln 2)^{1-1/d}$. Then, our choice of $p$ yields

$$(d-1) \cdot (4 \cdot (\ln n)/p)^{1/(d-1)} \cdot (\log n) - np/9 \leq 0,$$

i.e., the exponent in (2.9) is at most zero, and we can conclude that $E[T_3] = O(\text{poly}(n))$.

We turn to Step 4’s execution probability, which is at most $\Pr[F_3]$. For a fixed set $S'$ of size $s'$ considered by Step 3, we have

$$
\Pr\left[|\overline{N}(S')| > (n/p)^{1/2}\right] \leq \binom{n}{n/p^{1/2}} \cdot (1 - p)^{\binom{s'}{d-1}(n/p)^{1/2}},
$$

since the number of potential non-neighborhoods with a size of $(n/p)^{1/2}$ is $\binom{n}{n/p^{1/2}}$, and the probability that there is no edge connecting any of the $(n/p)^{1/2}$ vertices of such a set with some $d-1$ vertices in the set $S'$ is $(1 - p)^{\binom{s'}{d-1}(n/p)^{1/2}}$. Since there are $\binom{n}{s'}$ subsets of size $s'$, with $1 - x \leq e^{-x}$ for $x \in \mathbb{R}$ and $\binom{s'}{d-1} \geq \left(\frac{s}{d-1}\right)^{d-1}$, we conclude that

$$
\Pr[\text{Step 4 is executed}] \leq \Pr[F_3] = \Pr\left[\exists S' \subseteq V, |S'| = s' : |\overline{N}(S')| > (n/p)^{1/2}\right] \\
\leq \binom{n}{s'} \cdot \binom{n}{n/p^{1/2}} \cdot (1 - p)^{\binom{s'}{d-1}(n/p)^{1/2}} \\
\leq \exp\left((s' \cdot \ln n) + (n/p)^{1/2} \cdot (\ln n) - p \cdot \left(\frac{s'}{d-1}\right)^{d-1} \cdot (n/p)^{1/2}\right). \quad (2.10)
$$

By choice of $s'$, we have

$$
p \cdot (s'/(d-1))^{d-1} \cdot (n/p)^{1/2} = 4 \cdot (\ln n) \cdot (n/p)^{1/2}.
$$

Furthermore, $s' = o((n/p)^{1/2})$: For $d \geq 3$, this is trivial, and for $d = 2$, we have

$$
s' = o((n/p)^{1/2}) \Rightarrow p = \omega((\ln n)^{2}/n),
$$

which holds by choice of $p$. Now, (2.10) yields

$$
\Pr[\text{Step 4 is executed}] = \exp\left(o((n/p)^{1/2}) \cdot (\ln n) - 3 \cdot (n/p)^{1/2} \cdot \ln n\right) \\
\leq \exp\left(-(\ln n) \cdot 2 \cdot (n/p)^{1/2}\right).
$$
Step 4’s failure probability is 

$$\Pr[F_4] = \Pr \left[ \exists S'' \subseteq V, |S''| = 2 \cdot (n/p)^{1/2} : S'' \text{ is independent} \right] \leq \left( 2^{2(n/p)^{1/2}} \cdot (1 - p) \right) \cdot \left( \frac{2 \cdot (n/p)^{1/2}}{d} \right)^d \exp \left( \ln n \cdot 2 \cdot (n/p)^{1/2} - p \cdot \left( \frac{2 \cdot (n/p)^{1/2}}{d} \right)^d \right),$$

(2.11)

since the probability that a subset of size $2 \cdot (n/p)^{1/2}$ is independent is $(1 - p)^{2^{2(n/p)^{1/2}}}$. It can be easily seen that for all $d \geq 2$, we have $p \cdot \left( \frac{2 \cdot (n/p)^{1/2}}{d} \right)^d \leq n$. For $d = 2$, this is equivalent to $n \geq n$, and for $d \geq 3$, this is equivalent to $p \leq d^{2d/(2-d)} \cdot n$, where the latter is trivially fulfilled. By choice of $p$, it follows that 

$$\ln n \cdot 2 \cdot (n/p)^{1/2} \leq 2 \cdot ((\ln n)/c(d))^{1/2} \cdot n^{1-1/(2d)} = o(n),$$

which implies 

$$\ln n \cdot 2 \cdot (n/p)^{1/2} = o \left( \frac{p \cdot \left( \frac{2 \cdot (n/p)^{1/2}}{d} \right)^d}{d} \right),$$

and hence, with (2.11) we get 

$$\Pr[F_4] \leq \exp \left( - (1 - o(1)) \cdot p \cdot \left( \frac{2 \cdot (n/p)^{1/2}}{d} \right)^d \right) \leq \exp \left( - (1 - o(1)) \cdot n \right) \leq 2^{-n}.$$ 

(2.12)

(2.13)

Therefore, the probability that Step 5 is executed since Step 4 fails is at most $2^{-n}$. The only other way that Step 5 can be executed is that Step 1 finds that $|\big(C| \leq \left( (d - 2)! \cdot (\ln n)/(6p) \right)^{1/(d-1)}$. Using $\varepsilon = 1/3$, Lemma 1 yields that the probability of this event is at most $e^{-\varepsilon n} \leq 2^{-n}$. The lemma is applicable, since $p \in [c(d) \cdot (\ln n)/n^{1-1/d}, 3/4]$ and $d \geq 2$ in our theorem yield that $p$ is in the lemma’s legal range $[c(d, 1/3) \cdot (\ln n)^d/n^{d-1/(d-1)}, 3/4]$ of edge probabilities. Since both events leading to the execution of Step 5 have probability at most $2^{-n}$, and since its effort is $O(\text{poly}(n) \cdot 2^n)$, its expected running time is $O(\text{poly}(n) \cdot 2^n \cdot 2^{-n}) = O(\text{poly}(n))$. □
2.3 Approximating the Chromatic Number

Analogously to Lemma 1 for Independent Set, Lemma 2 yields the following algorithm for Coloring.

Algorithm 6 ApproxColoring(H, p, ε)

1: Run GreedyColor(H) and let C be the coloring it computes.
2: if |C| < 2n · (p/(ε · ln n))^{1/(d-1)} then output C.
3: Find an optimal coloring by exhaustive search and output it.

The effort of the exhaustive search step (Step 3) is higher than in our algorithms for Independent Set. We test for all partitions C’ of V into nonempty subsets whether C’ is a feasible coloring, and output a feasible coloring with the fewest colors. Since there are at most n^n = e^{n ln n} partitions to be tested, the effort of Step 3 is O(poly(n) · e^{n ln n}). However, since Lemma 2 yields an upper bound of e^{-n ln n} for the execution probability of Step 3, we still have a polynomial expected running time of O(poly(n) · e^{n ln n} · e^{-n ln n}) = O(poly(n)). Theorem 3 gives our analysis of ApproxColoring:

Fix an integer d ≥ 2 and ε > 0. For probability p = p(n) with

\[ \frac{1}{n^{\frac{d-1}{2}} \cdot e^{\frac{d}{2}}} \leq p \leq \frac{3}{4}, \]

ApproxColoring(H, p, ε) has approximation guarantee O(n · (p/\log n)^{1/(d-1)}) and polynomial expected running time for random inputs H from H_d(n, p).

We omit the proof of this theorem, which is completely analogous to the one of Theorem 1. The difference is that we use Lemma 2 instead of Lemma 1. It is easy to see that by choice of p in the theorem, p is also legal for Lemma 2 so that we can apply it.

Improving the Approximation Guarantee As done above for ApproxIndSet, we can improve the approximation guarantee of algorithm ApproxColoring for a smaller range of edge probabilities. The analysis of the improved algorithm ImprovedColoring (Algorithm 7 below) for the case d ≥ 3 is given by Theorem 4. Here, the lower bound on p is Ω((\log n)/n^{1−1/d}), matching the achieved bound in Theorem 2 for Independent Set. For the graph case d = 2, the range of possible edge probabilities is slightly smaller than what we achieved for Independent Set.
The reason is that in the analysis, we use Lemma 2, which for $d = 2$ demands
\[
p \geq \frac{2\epsilon \cdot (\ln n)^{3/2}}{n^{1/2-2\epsilon}} = \omega((\log n)/n^{1/2}) ,
\]
while the lower bound $p = \Omega((\log n)/n^{1-1/d})$ from Theorem 2 in case $d = 2$ is $p = \Omega((\log n)/n^{1/2})$. The graph case is handled by Theorem 5.

Algorithm ImprovedColoring uses an algorithm called Eppstein($H$) as a subroutine. The latter is discussed below. At the moment, it suffices to know that it computes an optimal coloring of a graph.

Algorithm 7 ImprovedColoring($H, p, \epsilon$)

1: Run GreedyColor($H$) and let $C$ be the coloring it computes. if $|C| \geq 2n \cdot (p/(\epsilon \cdot \ln n))^{1/(d-1)}$ then go to Step 5.
2: For all $S \subseteq V$, $|S| = d - 2$, compute $\lambda_1(M(G_{pro}(H, S), p))$. Let $m$ be the maximum of the computed eigenvalues. if $m \leq 4 \cdot (n/p)^{1/2}$ then output $C$.
3: Set $s' := (d - 1) \cdot (4 \cdot (\ln n)/p)^{1/(d-1)}$. For all $S' \subseteq V$, $|S'| = s'$, compute $|\overline{N}(S')|$. if $|\overline{N}(S')| \leq (n/p)^{1/2}$ for all tested subsets $S'$ then output $C$.
4: Check all subsets $S'' \subseteq V$ with $|S''| = 2 \cdot (n/p)^{1/2}$. if none of them is independent then output $C$.
5: if $d = 2$ then

   Run Eppstein($H$) to find an optimal coloring and output it.

else

   Find and optimal coloring by exhaustive search and output it.
end if

Notice that ImprovedColoring is very similar to algorithm ImprovedIndSet. A notable difference is Step 5, which distinguishes between $d = 2$ and $d \geq 3$. The reason is the following: For $d = 2$, (2.12) only yields a failure probability of $e^{-(1-o(1))n}$ for Step 4. Executing exhaustive search in Step 5 would take time $\Omega(e^{(1-o(1))n})$. This would result in an exponential expected running time of

\[
\Omega(\exp((1 - o(1))n \ln n) \cdot \exp(-(1 - o(1))n)) = \Omega(\exp((1 - o(1))n \ln n)) .
\]

We hence use an algorithm presented by Eppstein [Epp03], which computes an optimal coloring of a graph $H$ with $n$ vertices in time

\[
O\left(\frac{4}{3} + \frac{3^{4/3}/4}{n}\right) = O(2.4151n) = O\left(e^{0.8818n}\right) .
\]

We denote the algorithm by Eppstein($H$) in the following. For convenience, we restate Theorems 4 and 5. Theorem 4 states the following.
2.3. Approximating the Chromatic Number

Fix an integer $d \geq 3$ and set $\varepsilon := 1/(d \cdot 2^d)$. There is a constant $c(d) > 0$ such that for probability $p = p(n)$ with $c(d) \cdot (\ln n)/n^{1-1/d} \leq p \leq 3/4$, $\text{ImprovedColoring}(H, p, \varepsilon)$ has approximation guarantee $O(n^{1/2} \cdot p^{-(d-3)/(2d-2)})/((\log n)^{1/(d-1)})$ and polynomial expected running time for random inputs $H$ from $H_d(n, p)$.

Theorem 5 claims the following.

Fix $\varepsilon > 0$. For probability $p = p(n)$ with $2\varepsilon \cdot (\ln n)^{3/2}/n^{1/2-2\varepsilon} \leq p \leq 3/4$, $\text{ImprovedColoring}(H, p, \varepsilon)$ has approximation guarantee $O(n^{1/2} / \log n)$ and polynomial expected running time for random inputs $H$ from $G(n, p)$.

Proof of Theorems 4 and 5. First, we prove Theorem 4 ($d \geq 3$). Steps 1–4 of $\text{ImprovedIndSet}$ have the same efforts and execution probabilities as in algorithm $\text{ImprovedIndSet}$. Thus, the proof of Theorem 2 shows a polynomial expected running time for these steps. Step 5 has effort $O(\text{poly}(n) \cdot e^{\varepsilon \ln n})$ as mentioned above. It is only executed if Step 1 finds $|C| \geq 2n \cdot (p/(\varepsilon \cdot \ln n))^{1/(d-1)}$ or if Step 4 fails. The former happens with probability less than $e^{-n \ln n}$ due to Lemma 2. For the latter probability, (2.12) yields an upper bound of

$$\exp\left(-\left(\left(1 - o(1)\right) \cdot p \cdot \left(\frac{2 \cdot (n/p)^{1/2}}{d}\right)^d\right)\right) = \exp\left(-\Omega(p \cdot (n/p)^{d/2})\right)$$

$$= \exp\left(-\Omega(n^{d/2})\right) \leq \exp(-n \ln n) \ , \quad (2.14)$$

since $d \geq 3$. Hence, Step 5 has execution probability $O(e^{-n \ln n})$ and expected running time $O(\text{poly}(n) \cdot e^{\varepsilon \ln n} \cdot e^{-n \ln n}) = O(\text{poly}(n))$. It is easy to see that for our choice of $\varepsilon = 1/(d \cdot 2^d)$, the lower bound on $p$ in Lemma 2 is asymptotically smaller than the lower bound on $p$ in our Theorem. Thus, we can apply the lemma.

We turn to the approximation guarantee. First, observe that for every hypergraph $H$, it holds that $\chi(H) \geq n/\alpha(H)$, since all color classes in a coloring are independent and thus of size at most $\alpha(H)$. Assume that one of the Steps 2–4 outputs $C$. The proof of Theorem 2 shows that in this case, $\alpha(H) = O((n/p)^{1/2})$, and hence

$$\chi(H) = \Omega\left(\frac{n}{(n/p)^{1/2}}\right) = \Omega((np)^{1/2}) \ ,$$

Furthermore,

$$|C| < 2n \cdot (p/(\varepsilon \cdot \ln n))^{1/(d-1)} = O(n \cdot (p/ \log n)^{1/(d-1)}) \ ,$$
since otherwise Step 1 switches to exhaustive search. Therefore, we achieve approximation ratio
\[
\frac{|C|}{\chi(H)} = O\left(n \cdot \left(\frac{p}{\log n}\right)^{1/(d-1)}\right) = O(n^{1/2} \cdot \frac{p^{-(d-3)/(2d-2)}}{(\log n)^{1/(d-1)}}).
\]

We turn to the proof of Theorem 5 (i.e., the case \(d = 2\)). For the case \(d = 2\), Theorem 2 again yields a polynomial expected running time for Steps 1–4. As discussed, the lower bound on \(p\) in the Theorem is the one of Lemma 2 for \(d = 2\). Hence, the lemma is applicable and shows that Step 5 is reached from Step 1 with probability less than \(e^{-n \ln n}\). The failure probability of Step 4 is at most \(e^{-(1-o(1))n}\) by (2.13). Thus, Step 5’s execution probability is \(O(e^{-(1-o(1))n})\). As mentioned, Step 5’s effort is \(O(e^{0.8818n})\). An expected running time of \(O(e^{-(1-o(1))n+0.8818n}) = O(1)\) follows. The approximation guarantee can be analyzed as above. \(\square\)

### 2.4 The Expected Behavior of Greedy Independent Set and Coloring

In this section, we prove Corollary 1, giving our analysis of the expected behavior of GreedyColor when run on a random \(H_d(n, p)\) hypergraph. In the proof of the corollary, we exploit Lemmas 1 and 2, which give \(e^{-n \ln n}\) upper bounds on the probability that GreedyColor fails to produce a “good” independent set and coloring, respectively. Using this, we can prove the statement in Corollary 1 which says that GreedyColor with probability exponentially close to 1 achieves a certain approximation ratio for Independent Set and Coloring. To be able to prove this statement, we need exponentially small (i.e., also \(e^{-n \ln n}\)) tail bounds on the independence and chromatic number of \(H_d(n, p)\). These are given in Lemma 11 and Corollary 4, which we present and prove now.

**Lemma 11.** Fix \(d \geq 2\) and let \(H\) be a random hypergraph drawn from \(H_d(n, p)\). With probability at least \(1 - 1/n\),
\[
\alpha(H) = O\left(\left(\frac{\log n}{p}\right)^{\frac{1}{d-1}}\right).
\]
Furthermore, if \(p \geq 2d \cdot (\ln n)/n^{d-1}\), with probability at least \(1 - e^{-n \ln n}\),
\[
\alpha(H) = O\left(\left(\frac{n \log n}{p}\right)^{\frac{1}{d}}\right).
\]

Notice that (2.16) is optimal up to a constant factor: Assume we want to prove that
\[
Pr\left[\alpha(H) > f \cdot \left(\frac{n \ln n}{p}\right)^{\frac{1}{d}}\right] \leq e^{-n \ln n}
\]
(2.17)
2.4. The Expected Behavior of Greedy Independent Set and Coloring

for an as small as possible value \( f > 0 \). With (2.16), we achieve this for a constant \( f > 0 \). Now, assume that \( f = f(n) = o(1) \). We show that here, it is impossible to achieve an upper bound of \( e^{-n \ln n} \) on the probability in 2.17. Therefore, (2.16) cannot be improved by more than a constant factor. To prove our claim, let \( H = (V, E) \) be drawn from \( H_d(n, p) \), and fix a set \( V' \subseteq V \) with \( |V'| = 2f \cdot \left( \frac{n \ln n}{p} \right)^{\frac{1}{2}} \), and abbreviate \( t := |V'| \). For \( p \leq 3/4 \), we can infer with \( 1 - p \geq e^{-2p} \) and \( \binom{t}{d} \leq t^d \) that

\[
\Pr\left[ \alpha(H) > f \cdot \left( \frac{n \ln n}{p} \right)^{\frac{1}{2}} \right] \geq \Pr[V' \text{ is independent}]
\]

\[
= (1 - p)^{\binom{t}{d}}
\]

\[
\geq \exp(-2p \cdot t^d)
\]

\[
= \exp\left(-2p \cdot \left( 2f \cdot \left( \frac{n \ln n}{p} \right)^{\frac{1}{2}} \right)^d \right)
\]

\[
= \exp(-2^{d+1} \cdot f^d \cdot n \ln n)
\]

\[
= \exp(-o(1) \cdot n \ln n)
\]

by choice of \( f = o(1) \). Therefore, for \( f = o(1) \), it is impossible to prove (2.17), which requires an upper bound of \( e^{-n \ln n} \) on the probability that \( \alpha(H) > f \cdot \left( \frac{n \ln n}{p} \right)^{\frac{1}{2}} \).

**Proof of Lemma 11.** We analyze \( \Pr[\alpha(H) > t] \) for the random hypergraph \( H = (V, E) \) from \( H_d(n, p) \). Here, we write the threshold \( t \) in the form

\[
t = f \cdot \left( \frac{\ln n}{p} \right)^{\frac{1}{d-1}}
\]

(2.18)

for a factor \( f > 0 \). If \( \alpha(H) > t \), then among the \( \binom{n}{t} \) subsets of \( V \) with size \( t \), there is one such that none of its \( \binom{d}{\frac{t}{d}} \) subsets of size \( d \) is an edge of \( H \). With \( \binom{n}{t} \leq e^{\ln n} \), \( 1 - p \leq e^{-p} \), and \( \binom{d}{\frac{t}{d}} \geq (t/d)^d \), we get

\[
\Pr[\alpha(H) > t]
\]

\[
\leq \binom{n}{t} \cdot (1 - p)^{\binom{d}{\frac{t}{d}}}
\]

\[
\leq \exp(t \cdot (\ln n) - p \cdot (t/d)^d)
\]

\[
= \exp\left( (\ln n) \cdot f \cdot \left( \frac{\ln n}{p} \right)^{\frac{1}{d-1}} - p \cdot \left( \frac{f}{d} \cdot \left( \frac{\ln n}{p} \right)^{\frac{1}{d-1}} \right)^d \right)
\]

\[
= \exp\left( f \cdot \frac{(\ln n)^{d/(d-1)}}{p^{1/(d-1)}} - \left( \frac{f^d}{d} \cdot \frac{(\ln n)^{d/(d-1)}}{p^{1/(d-1)}} \right) \right)
\]

\[
= \exp\left( \left( f - \frac{f^d}{d} \right) \cdot \frac{(\ln n)^{d/(d-1)}}{p^{1/(d-1)}} \right).
\]

(2.19)
Obviously, if we choose \( f = c \) for a large enough constant \( c > 1 \) (i.e., our choice depends only on \( d \)), then \( f - \left( \frac{\ln n}{p} \right)^{1/d} = -\Omega(1) \). Then, (2.19) yields

\[
\Pr \left[ \alpha(H) > c \cdot \left( \frac{\ln n}{p} \right)^{1/d} \right] \leq \exp\left( -\Omega(1) \cdot \frac{(\ln n)^{d/(d-1)}}{p^{1/(d-1)}} \right) \\
\qquad \leq \exp(- \ln n) = 1/n .
\]

This yields the claim in the lemma that with probability at least \( 1 - 1/n \), we have \( \alpha(H) = O\left( \left( \frac{\log n}{p} \right)^{2/d} \right) \).

We turn to the second statement in the lemma. Observe that

\[
f - \left( \frac{f}{d} \right)^d \leq -\frac{1}{2} \cdot \left( \frac{f}{d} \right)^d \quad (2.20)
\]

if and only if

\[
f \geq 2^{1/(d-1)} d^{d/(d-1)} . \quad (2.21)
\]

We set

\[
f := d 2^{1/d} \cdot n^{1/d} \cdot \left( \frac{p}{\ln n} \right)^{1/(d(d-1))} . \quad (2.22)
\]

Then, standard calculations show that assuming \( p \geq 2d \cdot (\ln n)/n^{d-1} \) as in the lemma, (2.21) holds, which in turn implies that (2.20) holds. Using our definition of \( f \) in (2.22), our threshold \( t \) defined in (2.18) becomes

\[
t = f \cdot \left( \frac{\ln n}{p} \right)^{1/d} = 2^{1/d} \cdot \left( \frac{n \ln n}{p} \right)^{1/d} . \quad (2.23)
\]

Plugging (2.23) and (2.20) into (2.19), we get

\[
\Pr \left[ \alpha(H) > 2^{1/d} \cdot \left( \frac{n \ln n}{p} \right)^{1/d} \right] \leq \exp\left( -\frac{1}{2} \cdot \left( \frac{f}{d} \right)^d \cdot \frac{(\ln n)^{d/(d-1)}}{p^{1/(d-1)}} \right) . \quad (2.24)
\]

Standard calculations show that for the exponent in (2.24), we have

\[
-\frac{1}{2} \cdot \left( \frac{f}{d} \right)^d \cdot \frac{(\ln n)^{d/(d-1)}}{p^{1/(d-1)}} \leq -n \ln n \Leftrightarrow f \geq d 2^{1/d} \cdot n^{1/d} \cdot \left( \frac{p}{\ln n} \right)^{1/(d(d-1))} ,
\]

where the latter is fulfilled by our definition of \( f \) in (2.22). We conclude with (2.24) that

\[
\Pr \left[ \alpha(H) > 2^{1/d} \cdot \left( \frac{n \ln n}{p} \right)^{1/d} \right] \leq \exp(- n \ln n) . \quad (2.25)
\]

This proves our claim in the lemma that with probability at least \( 1 - e^{-n \ln n} \), we have \( \alpha(H) = O\left( \left( \frac{n \log n}{p} \right)^{2/d} \right) \).

\( \square \)
2.4. The Expected Behavior of Greedy Independent Set and Coloring

Above, we argued that for every hypergraph \( H \), the chromatic number fulfills \( \chi(H) \geq n/\alpha(H) \). Therefore, Lemma 11 immediately yields the following corollary for the behavior of the chromatic number of a random \( H_d(n, p) \) hypergraph.

**Corollary 4.** Fix \( d \geq 2 \) and let \( H \) be a random hypergraph drawn from \( H_d(n, p) \). With probability at least \( 1 - 1/n \),

\[
\chi(H) = \Omega\left(n \cdot \left(\frac{p}{\log n}\right)^{\frac{1}{2d-1}}\right).
\] (2.26)

Furthermore, if \( p \geq 2d \cdot (\ln n)/n^{d-1} \), with probability at least \( 1 - e^{-n \ln n} \),

\[
\chi(H) = \Omega\left(n^{1/2} \cdot \left(\frac{p}{\log n}\right)^{\frac{1}{2}}\right).
\] (2.27)

We are now in position to prove Corollary 1, which states the following.

**Proof of Corollary 1.** We start with the expected approximation ratio of \( \text{GreedyColor} \) for \( \text{Independent Set} \). Let \( I \) be the greedy independent set for a random hypergraph \( H \) from \( H_d(n, p) \), i.e., let \( C := \text{GreedyColor}(H) \) and let \( I := \text{big}(C) \). Under its assumptions, Lemma 1 yields that with probability at least \( 1 - e^{-n \ln n} \),

\[
|I| = \Omega\left(\left(\frac{\log n}{p}\right)^{1/(d-1)}\right).
\] (2.28)

Furthermore, Lemma 11 shows that with probability at least \( 1 - 1/n \), (2.15) holds, i.e., \( \alpha(H) = O\left(\left(\frac{\log n}{p}\right)^{\frac{1}{d-1}}\right) \). The approximation ratio of the greedy independent set is \( \text{ar}(I) = \alpha(H)/|I| \). Clearly, if both (2.28) and (2.15) hold, we have \( \text{ar}(I) = O(1) \). Otherwise, we still have \( \text{ar}(I) \leq n = O(n) \), since no solution can have an approximation ratio larger than \( n \). This happens with probability at most \( e^{-n \ln n} + 1/n = O(1/n) \) due to our above discussion. We get

\[
\text{E}[\text{ar}(I)] = O(1) \cdot \text{Pr}[(2.28) \text{ and (2.15) hold}]
+ O(n) \cdot \text{Pr}[(\text{not both of (2.28) and (2.15) hold}]
= O(1) + O(n) \cdot O(1/n) = O(1)
\]

completing our proof of a constant approximation ratio in expectation for \( \text{Independent Set} \). We turn to the proof that \( I \) has approximation ratio \( O\left(n^{1/d} \cdot \left(\frac{p}{\log n}\right)^{\frac{1}{d-1}}\right) \).
with probability exponentially close to 1. Notice that edge probabilities which are legal with respect to Lemma 1 are also legal with respect to Lemma 11, since the lower bound on \( p \) in the latter lemma is smaller than the one in the former lemma. Therefore, under the assumptions of Lemma 1, Lemma 11 yields that with probability at least \( 1 - e^{-n \ln n} \), we have \( \alpha(H) = O\left( \left( \frac{n \log n}{p} \right)^{\frac{1}{d}} \right) \), i.e., (2.16) holds. With (2.28), we conclude that with probability at least \( 1 - 2 \cdot e^{-n \ln n} \),

\[
\alpha(H) = O\left( \left( \frac{n \log n}{p} \right)^{\frac{1}{d}} \right) = O\left( n^{1/d} \cdot \left( \frac{p \log n}{\log n} \right)^{\frac{1}{d-1}} \right)
\]
as claimed. For Coloring, the proofs are completely analogous. Here, we use Lemma 2 instead of Lemma 1 and Corollary 4 instead of Lemma 11. Lemma 2 shows that with high probability, the number of color classes in the greedy coloring \( C \) is small enough, while Corollary 4 shows that the chromatic number is large enough with high probability to yield our desired approximation ratios. \( \square \)

### 2.5 Small Edge Probabilities

In Section 1.2.1, we mentioned that for very small edge probabilities, we can approximate \textsc{Independent Set} using an algorithm by Bertram-Kretzberg and Lefmann [BKL99], which we call \textsc{EdgeNumIndSet}, as a tool. The performance of \textsc{EdgeNumIndSet} is described by the following theorem, proved in [BKL99]. For completeness, we sketch the algorithm and the proof here.

**Theorem 12.** There is a deterministic, polynomial worst-case running time algorithm \textsc{EdgeNumIndSet} with the following properties. Given any \( d \)-uniform hypergraph \( H = (V, E) \), \( d \geq 2 \), with \( |E| \geq |V|/d \), it finds an independent set of size

\[
\Omega\left( \frac{|V|^{d/(d-1)}}{|E|^{(d-1)/(d-1)}} \right).
\]

Otherwise, i.e., \( |E| < |V|/d \), it finds an independent set of size \( \Omega(n) \).

**Proof.** We describe a randomized algorithm computing an independent set of the claimed size, and then show that the algorithm can be derandomized to yield the deterministic algorithm \textsc{EdgeNumIndSet}. Our randomized algorithm works as follows.

Let \( q \in (0, 1) \) denote a probability\(^1\). The value of \( q \) will be determined later. Given \( H = (V, E) \), we randomly choose a set \( V' \subseteq V \) of vertices by independently

\(^{1}\)We chose to denote the probability by \( q \) instead of \( p \) to make clear that it is not to be confused with the edge probability \( p \) in our \( H_d(n, p) \) model. We do not consider random hypergraphs here but a randomized algorithm.
including every vertex from $V$ with probability $q$. For convenience, let $m = |E|$. (As always, $n = |V|$.) We get a random subset $V' \subseteq V$ with expected size $E[|V'|] = nq$.

Given the random set $V'$ of vertices, let $E'$ be the set of edges $e \in E$ such that $e \subseteq V'$, i.e., $H' = (V', E')$ is the subhypergraph of $H$ induced by $V'$. If $E' = \emptyset$, we are done, since we can output $V'$ as an independent set of $H$. If this is not the case, we can do the following: As long as $E' \neq \emptyset$, we choose an edge $e \in E'$ and remove one of its vertices from $V'$. This way, the number of vertices in $V'$ decreases by one, and the number of edges in $E'$ decreases by at least one. Hence, after at most $|E'|$ steps removing a single vertex from $V'$, it holds that $E' = \emptyset$, and we can output the (reduced) set $V'$ as an independent set of $H$.

The above arguments show that, given the random set $V'$, we find an independent set $I$ of size $|I| \geq |V'| - |E'|$, and by linearity of expectation we can conclude that the expected size of our solution is

$$E[|I|] \geq E[|V'|] - E[|E'|] = nq - mq^d =: f(q),$$

since each of the $m$ edges in $H$ lies in the random edge set $E'$ with probability $q^d$. Standard analysis shows that $f(q)$ is maximal for

$$q = \left(\frac{n}{dm}\right)^{1/(d-1)},$$

(2.30)

For $m \geq n/d$, as assumed in the first statement of the theorem, we can use this value as a legal probability $q \leq 1$ in the following. Otherwise, i.e., $m < n/d$, it is trivial to find an independent set of size $\Omega(n)$ as claimed in the theorem: For every edge, we choose one of its vertices and remove it from $V$. Afterwards, the reduced set $V$ is an independent set of the original hypergraph $H$, and still $|V| = \Omega(n)$.

Let us return to the non-trivial case $m \geq n/d$. Using (2.30) as the value for $q$ (now, we fix $q$ that way), we can conclude with (2.29) that our algorithm finds an independent set $I$ of expected size

$$E[|I|] \geq n \cdot \left(\frac{n}{dm}\right)^{1/(d-1)} - m \cdot \left(\frac{n}{dm}\right)^d = \frac{n^{d/(d-1)}}{m^{1/(d-1)}} \cdot \left(\frac{1}{d^{1/(d-1)}} - \frac{1}{d^d}ight) = \Omega \left(\frac{n^{d/(d-1)}}{m^{1/(d-1)}}\right),$$

(2.31)

which is the lower bound our theorem claims to hold for the size of the found independent set. Therefore, our randomized algorithm in expectation finds an independent set of the required size.

It remains to show that our algorithm can be derandomized to yield a deterministic version finding an independent set that is guaranteed to have at least the size
given by (2.31). To this end, let \( q_i, i = 1, \ldots, n \), denote the probability that vertex \( i \in V \) is included in \( V' \). We still include the vertices independently in \( V' \), but each vertex \( i \) has its own probability. We give a modified version of (2.29). It is easy to see that for our new process,
\[
E[|V'|] - E[|E'|] = \sum_{i \in V} q_i - \sum_{e \subseteq \{i, \ldots, j\} \subseteq E} q_i \cdot \cdots \cdot q_j. \tag{2.32}
\]
Now, we start with \( q_1 = \ldots = q_n = q \) for the value \( q \) from (2.30). This way, our new process still yields a random set \( V' \) such that \( E[|V'|] - E[|E'|] = \Omega\left(\frac{n^{d/(d-1)}}{m^{1/(d-1)}}\right) \), and using our “vertex removal” technique from above, we can find an independent set \( I \) in the end which in expectation has the required size.

Our derandomization now works as follows: Observe that (2.32) is linear in every \( q_i \) (for fixed \( q_j, j \neq i \)). Therefore, given the original value of \( q_i \), we can try \( q_i = 0 \) and \( q_i = 1 \) and be sure that for one of these values, (2.32) is at least as large as for the original value. In consequence, we can consider \( q_1, \ldots, q_n \) one by one, and for every \( q_i \), we can choose a value from \( \{0, 1\} \) in the described way without decreasing (2.32), i.e., in the end, we still have \( E[|V'|] - E[|E'|] = \Omega\left(\frac{n^{d/(d-1)}}{m^{1/(d-1)}}\right) \). But since all probabilities \( q_i \) are 0 or 1, the set \( V' \) is chosen deterministically, and we conclude that for our solution \( I \) which we output in the end, \( |I| \geq |V'| - |E'| = \Omega\left(\frac{n^{d/(d-1)}}{m^{1/(d-1)}}\right) \), as claimed in the theorem.

Finally, it is obvious that the algorithm can be implemented to run in worst-case polynomial time.

Exploiting Theorem 12, we can approximate Independent Set with the following algorithm SmallEdgePrIndSet\((H, p)\) for random hypergraphs \( H \) from \( H_d(n, p) \) and arbitrarily small edge probabilities \( p = p(n) \).

**Algorithm 8 SmallEdgePrIndSet\((H, p)\)**

1. Let \( \mu := \binom{n}{d} \cdot p \) denote the expected number of edges.
2. if \( p \geq 1/n^{d-1} \) then \( t := 18d^4\mu \) else \( t := 18n \) end if.
3. if \(|E| < t \) then run EdgeNumIndSet\((H)\) and output the computed independent set.
4. Find a largest independent set by exhaustive search and output it.

In Section 1.2.1, we presented Theorem 6, giving our analysis of SmallEdgePrIndSet’s performance. For convenience, we restate the theorem here.

*Fix an integer \( d \geq 2 \) and an arbitrary probability \( p = p(n) \). Let \( H \) be a random hypergraph from \( H_d(n, p) \). Then, algorithm SmallEdgePrIndSet\((H, p)\) has polynomial expected running time. For \( p \geq 1/n^{d-1} \), it has approximation guarantee \( O(n \cdot p^{1/(d-1)}) \). For \( p < 1/n^{d-1} \), it has approximation guarantee \( O(1) \).*
Proof of Theorem 6. We start with the approximation guarantee. First, consider $p \geq 1/n^{d-1}$. If Step 4 outputs its solution, it is optimal. Otherwise, Step 3 outputs \texttt{EdgeNumIndSet}(H), and $|E| < t$. With $\mu = \binom{n}{d} \cdot p \leq n^d \cdot p$ and $t = O(\mu)$, it follows that

$$|E| = O(n^d \cdot p) .$$

Therefore, using Theorem 12, it follows that the independent set $I$ found by \texttt{EdgeNumIndSet}(H) has size

$$|I| = \Omega\left(\frac{n^d}{|E|^{1/(d-1)}}\right) = \Omega\left(\frac{n^d}{(n^d \cdot p)^{1/(d-1)}}\right) = \Omega\left(\frac{1}{p^{1/(d-1)}}\right) .$$

With $\alpha(H) \leq n$, we conclude that we achieve an approximation ratio of

$$\frac{\alpha(H)}{|I|} = O\left(\frac{n}{1/p^{1/(d-1)}}\right) = O(n \cdot p^{1/(d-1)}) ,$$

as claimed in the theorem.

Now, assume $p < 1/n^{d-1}$. Here, if Step 3 outputs its solution, $|E| = O(n)$. Hence, \texttt{EdgeNumIndSet}(H) finds a solution $I$ of size

$$|I| = \Omega\left(\frac{n^d}{|E|^{1/(d-1)}}\right) = \Omega\left(\frac{n^d}{n^{1/(d-1)}}\right) = \Omega(n) .$$

With $\alpha(H) \leq n$, we immediately get a constant approximation ratio $O(1)$ as claimed.

We turn to the expected running time of the algorithm. Clearly, Steps 1–3 have polynomial worst-case running time. It hence suffices to upper bound the execution probability of the exhaustive search step, Step 4, by $2^{-n}$, since then, its expected running time is $O(\text{poly}(n) \cdot 2^n \cdot 2^{-n}) = O(\text{poly}(n))$.

To upper bound the probability that the number of edges deviates from its expectation, we use the following well-known Chernov bound: Let $X_1, \ldots, X_k$ be independent random variables $X_i \in \{0, 1\}$ with $\Pr[X_i = 1] = q_i$ for $i = 1, \ldots, k$. Let $X = \sum_{i=1}^k X_i$. Then, $\text{E}[X] = \sum_{i=1}^k q_i$, and for all $0 \leq \delta < 1$ and all $s \geq \text{E}[X]$, it holds that

$$\Pr[X \geq (1 + \delta) \cdot s] \leq e^{-s \delta^2/3} . \quad (2.33)$$
In case \( q_i = q \) for \( i = 1, \ldots, k \), the random variable \( X \) is \( B(k, q) \) binomially distributed with \( k \) trials and success probability \( q \). Our Chernov bound (2.33) hence includes the special case of binomially distributed random variables.

Consider \( p \geq 1/n^{d-1} \). To apply (2.33), notice that \(|E|\) is \( B\left(\binom{n}{d}, p\right) \) binomially distributed with \( k \) trials and success probability \( q \). Our Chernov bound (2.33) hence includes the special case of binomially distributed random variables. Let \( s = 12d^d \cdot \mu \). Since \( s > \mu = E[|E|] \), it is legal with respect to (2.33). Furthermore, set \( \delta = 1/2 \). Then, all parameters are suitable to apply our Chernov bound. Furthermore, for \( p \geq 1/n^{d-1} \), the threshold for \(|E|\) tested in the algorithm is \( t = 18d^d \cdot \mu \). Using our parameters, we get

\[
(1 + \delta) \cdot s = (1 + 1/2) \cdot 12d^d \cdot \mu = 18d^d \mu = t .
\]

If Step 4 is executed, then \(|E| \geq t = (1 + \delta) \cdot s\). Therefore, applying (2.33), we get

\[
\Pr[\text{Step 4 is executed}] \\
\leq \Pr[|E| \geq (1 + \delta) \cdot s] \\
\leq e^{-s \delta^2/3} \\
= e^{-12d^d \binom{n}{d} \cdot p^{1/12}} \quad (2.34) \\
\leq e^{-d^d \cdot (n/d)^d \cdot 1/n^{d-1}} \quad (2.35) \\
= e^{-n} \leq 2^{-n} .
\]

To get (2.34), we used the definition of \( s = 12d^d \cdot \mu \) and \( \mu = \binom{n}{d} \cdot p \), together with \( \delta^2/3 = 1/12 \). For (2.35), we used \( \binom{n}{d} \geq (n/d)^d \) together with \( p \geq 1/n^{d-1} \). We conclude that for \( p \geq 1/n^{d-1} \), Step 4’s execution probability is at most \( 2^{-n} \), i.e., sufficiently small to achieve polynomial expected running time in this step.

We turn to the case \( p < 1/n^{d-1} \). Then,

\[
\mu = \binom{n}{d} \cdot p \leq n^d \cdot p < n^d / n^{d-1} = n .
\]

Let \( s = 12n \geq 12\mu \) and \( \delta = 1/2 \). Then, \( s \) and \( \delta \) are legal for (2.33). For \( p < 1/n^{d-1} \), the threshold \( t \) is \( t = 18n \). Hence, if Step 4 is executed, \(|E| \geq t = 18n = (1 + \delta) \cdot s\). Now, (2.33) yields

\[
\Pr[\text{Step 4 is executed}] \\
\leq \Pr[|E| \geq (1 + \delta) \cdot s] \\
\leq e^{-s \delta^2/3} \\
= e^{-12n^{1/12}} \\
= e^{-n} \leq 2^{-n} ,
\]

which completes our proof. \( \square \)
2.6 Completing the Proof of Lemma 2

In the proof of Lemma 2, we claimed that (2.6) holds, i.e., for \( \sigma = \sum_{i=1}^{k}(1 - p)^{|C_i|/(d-1)!} \) (cf. (2.5)), we have \( k_0 \sigma / 2 \geq n \ln n \) and \( \sigma / 2 \geq \ln n \). Below, we prove this. Set

\[
f(x) := (1 - p)^{x^{d-1}/((d-1)!)}.
\]

Then, \( \sigma = \sum_{i=1}^{k} f(|C_i|) \). For a convex function \( g(x) \) with domain \( D, x_1, \ldots, x_m \in D \), and \( s = \sum_{i=1}^{m} x_i \), Jensen’s inequality (Theorem 14 in Appendix A.1) yields

\[
\sum_{i=1}^{m} g(x_i) \geq m \cdot g(s/m).
\]

Using this, we lower bound \( \sigma \). We show that \( f(x) \) is convex for \( x \) large enough, i.e., the second derivative \( f''(x) \geq 0 \). We have

\[
f'(x) = (1 - p)^{x^{d-1}/((d-1)!)} \cdot \frac{\ln(1 - p)}{(d-2)!} \cdot x^{d-2}.
\]

For \( d = 2 \), it follows that \( f''(x) = (\ln(1 - p))^2 \cdot (1 - p)^s \geq 0 \) for all \( x \in \mathbb{R} \), and thus \( f(x) \) is convex over \( \mathbb{R} \). Now consider \( d \geq 3 \). Here (since we deal with class sizes, in the following we can assume \( x \geq 1 \)),

\[
f''(x) = \frac{\ln(1 - p)}{(d-2)!} \cdot (1 - p)^{x^{d-1}/((d-1)!)} \cdot \left( \frac{\ln(1 - p)}{(d-2)!} \cdot x^{2d-4} + (d-2)x^{d-3} \right),
\]

and since \( \frac{\ln(1 - p)}{(d-2)!} < 0 \), we have \( f''(x) \geq 0 \) if and only if

\[
\frac{\ln(1 - p)}{(d-2)!} \cdot x^{2d-4} + (d-2)x^{d-3} \leq 0
\]

\[
\Leftrightarrow \frac{\ln(1 - p)}{(d-2)!} \cdot x^{d-1} \leq -(d-2)
\]

\[
\Leftrightarrow -\frac{p}{(d-2)!} \cdot x^{d-1} \leq -(d-1) \quad \text{(since } \ln(1 - p) \leq -p\text{)}
\]

\[
\Leftrightarrow x^{d-1} \geq \frac{(d-1)!}{p}
\]

\[
\Leftrightarrow x \geq \left( \frac{(d-1)!}{p} \right)^{1/(d-1)}.
\]

We conclude that for all \( d \geq 2 \), \( f(x) \) is convex for \( x \geq \left( \frac{(d-1)!}{p} \right)^{1/(d-1)} \).
We lower bound \( \sigma = \sum_{i=1}^{k} f(|C_i|) \). Let a class \( C_i \) be large if \( |C_i| \geq \left( \frac{d-1}{p} \right)^{1/(k-1)} \) and small otherwise, and denote the corresponding sets of classes by \( L \) and \( S \), respectively. We split our sum in two and get

\[
\sigma = \sum_{i=1}^{k} f(|C_i|) = \sum_{C_i \in L} f(|C_i|) + \sum_{C_i \in S} f(|C_i|) .
\]  

(2.37)

Let \( s := \sum_{C_i \in L} |C_i| \). Since \( f(x) \) is convex for \( x \geq \left( \frac{d-1}{p} \right)^{1/(k-1)} \) and \( |C_i| \geq \left( \frac{d-1}{p} \right)^{1/(k-1)} \) for all \( C_i \in L \), with (2.36), we get

\[
\sum_{C_i \in L} f(|C_i|) \geq |L| \cdot f(s/|L|) = |L| \cdot (1 - p)^{\frac{s}{|L|} / ((d-1)/p)} .
\]  

(2.38)

We consider the small classes. Since \( f(x) \) is strictly decreasing and \( |C_i| < \left( \frac{d-1}{p} \right)^{1/(k-1)} \) for all \( C_i \in S \),

\[
\sum_{C_i \in S} f(|C_i|) 
\geq \sum_{C_i \in S} f \left( \left( \frac{d-1}{p} \right)^{1/(k-1)} \right) 
= (k - |L|) \cdot f \left( \left( \frac{d-1}{p} \right)^{1/(k-1)} \right) 
= (k - |L|) \cdot (1 - p)^{1/p} 
\geq (k - |L|) \cdot e^{-2p/p} = (k - |L|) \cdot e^{-2} ,
\]

(2.39)

using \( 1 - p \geq e^{-2p} \) since \( p \leq 3/4 \). Now, using (2.38) and (2.39), with (2.37) we get

\[
\sigma = \sum_{i=1}^{k} f(|C_i|) \geq \left| L \right| \cdot \frac{(1 - p)^{s/|L| / ((d-1)/p)} + (k - |L|) \cdot e^{-2}}{\sigma_1} .
\]

We show the first inequality, i.e., \( k_0 \sigma/2 \geq n \ln n \), from (2.6). First, consider the case \( |L| < k/2 \). Then,

\[
k_0 \sigma/2 \geq k_0 \sigma_2/2 > k_0 (k/2) \cdot e^{-2}/2 = \Omega(k_0^2) ,
\]

since \( k \geq k_0 \) throughout our proof. With \( k_0 = \Omega(n \cdot (p/\ln n)^{1/(d-1)}) \) and clearly \( p = \omega((\ln n)^{(d+1)/2}/n^{(d-1)/2}) \) by choice of \( p \), we infer

\[
k_0 \sigma/2 = \Omega \left( n^2 \cdot \left( \frac{p}{\ln n} \right)^{\frac{2}{d-1}} \right) 
= \omega \left( n^2 \cdot \left( \frac{\ln n}{n} \right)^{\frac{d+1}{2}} \right) 
= \omega(n \ln n) \geq n \ln n
\]
as claimed. Now assume $|L| \geq k/2$. With $k \geq k_0$, $1 - p \geq e^{-2p}$ since $p \leq 3/4$, and $s \leq n$, we get

$$k_0 \sigma / 2 \geq k_0 \sigma_1 / 2 \geq (k_0 k/4) \cdot (1 - p)^{(s/(k/2))^{d-1}/((d-1)!)} \quad \text{(by definition of } \sigma_1)$$

$$\geq (k_0^2/4) \cdot (1 - p)^{(2s/k_0)^{d-1}/((d-1)!)} \geq (k_0^2/4) \cdot \exp \left( -2p(2n/k_0)^{d-1}/((d-1)! \right)$$

$$= (k_0^2/4) \cdot \exp \left( -\frac{(\ln n) e^{2d}}{(d-1)!} \right)$$

$$= \frac{n^2}{4} \cdot \left( \frac{p}{\epsilon \ln n} \right)^{\frac{2}{d-1}} \cdot n^{\frac{e^{2d}}{(d-1)!}}, \quad (2.40)$$

using $k_0 = n \cdot (p/(\epsilon \cdot \ln n))^{1/(d-1)}$ for (2.40). By choice of

$$p \geq \frac{4^{d-1} \cdot (\ln n)^{\frac{d+1}{2}}}{n^{\frac{d+1}{2} \cdot \frac{e^{2d}}{(d-1)!}}}$$

in Lemma 2 (whose proof we complete here), we can infer for (2.40) that

$$\left( \frac{p}{\epsilon \ln n} \right)^{\frac{2}{d-1}} \geq 4 \cdot \frac{(\ln n)^{\frac{d+1}{2}}}{n^{\frac{d+1}{2} \cdot \frac{e^{2d}}{(d-1)!}}} \frac{2}{d-1}$$

$$= 4 \cdot \frac{(\ln n)^{\frac{d+1}{2}}}{n^{\frac{d+1}{2} \cdot \frac{e^{2d}}{(d-1)!}}}$$

$$= \frac{4 \ln n}{n} \cdot n^{\frac{e^{2d}}{(d-1)!}}. \quad (2.41)$$

Now, with (2.40) and (2.41) we get

$$k_0 \sigma / 2 \geq \frac{n^2}{4} \cdot \frac{4 \ln n}{n} \cdot n^{\frac{e^{2d}}{(d-1)!}} \cdot n^{\frac{e^{2d}}{(d-1)!}} = n \ln n$$

as claimed in (2.6). The second inequality in (2.6) follows from this, since

$$\sigma / 2 \geq \frac{n \ln n}{k_0} = \frac{n \ln n}{n \cdot (p/(\epsilon \cdot \ln n))^{1/(d-1)}} = \frac{\ln n}{(d-1)!} \geq \ln n.$$
On the one hand, we presented approximation algorithms which guarantee approximation ratio $O(n^{1/2} \cdot p^{-(d-3)/(2d-2)} / (\log n)^{1/(d-1)})$ and have polynomial expected running time over $H_d(n, p)$. On the other hand, we showed that GreedyColor, which guarantees polynomial running time, has constant approximation ratio in expectation. Furthermore, it achieves approximation ratio $O(n^{1/d} \cdot (p / \log n)^{1/(d-1)})$ with probability at least $1 - 2 \cdot e^{-n \ln n}$. Therefore, in some sense we have a trade-off between guarantees on running time and solution quality.

Also, we achieved some improvements for $d = 2$, i.e., in the $G(n, p)$ model: Firstly, we improved the lower bound on the possible edge probabilities $p$ of the algorithm for Independent Set in [KV02] from $p \geq 1/n^{1/2-\varepsilon}$ with $\varepsilon > 0$ fixed, to $p \geq c \cdot (\ln n)/n^{1/2}$, with $c > 0$ constant. This improvement of factor $\Theta(n^\varepsilon / \log n)$ is achieved since the lower bound on $p$ in Lemma 1, analyzing the distribution of the greedy independent set size, is smaller than the lower bound on $p$ in a corresponding lemma in [KV02]. Secondly, Lemma 2 states that for a random $G(n, p)$ graph, greedy coloring uses $O(np / \log n)$ colors with probability at least $1 - e^{-n \ln n}$. Hence, the probability that GreedyColor fails to produce a coloring with a small (in this sense) number of colors is at most $e^{-n \ln n}$. This upper bound improves on the larger bound $2^{-2np/\ln n}$ of a corresponding lemma in [KV02]. Due to this, we were able to simplify the algorithm for Coloring in [KV02]: This algorithm performs seven steps, while our algorithm ImprovedColoring only needs five.

**Open Problems** In the analysis of algorithms ImprovedIndSet and ImprovedColoring, we used that two bounds hold with high probability: A bound for the optimum and a bound for the value of the computed solution. Lemmas 1 and 2 give the bounds for the solution value for Independent Set and Coloring, respectively. The former lemma requires that the edge probability $p$ is of the form $p = \Omega(1/n^{d-1-\varepsilon})$, while the latter lemma requires that it is of the form $p = \Omega(1/n^{(d-1)/2-\varepsilon})$. While for large values of $d$, these lower bounds on $p$ are asymptotically much smaller than $1/n$, our bounds for the optimum do not have this property: In the analysis of ImprovedIndSet and ImprovedColoring, we exploit Lemma 3, presented by Krivelevich and Vu [KV02]. The lemma gives a tail bound for the largest eigenvalue of the auxiliary matrix defined in [KV02] for a random $G(n, p)$ graph. In our algorithms, given a random $H_d(n, p)$ hypergraph $H$, we compute some projection graphs for $H$ which are roughly distributed like $G(n, p)$, and computing its largest eigenvalue, could improve our lower bound on $p$. It would be interesting to improve the lower bound on $p$ to an as small value as in Lemma 1, i.e., $p = \Omega(1/n^{d-1-\varepsilon})$. 
to enlarge the range of possible edge probabilities for ImprovedIndSet and ImprovedColoring.

Another open problem is how to eliminate the need of knowing the edge probability $p$ underlying the given random hypergraph in our algorithms. In many applications, no natural value for $p$ even exists, since the considered hypergraph was not actually produced by a $H_d(n, p)$ model. Then, it is not clear, which value “the” edge probability should have. It would be therefore desirable to have an algorithm which automatically adapts to different values of $p$. 
In this chapter, we prove the results stated in Section 1.2.2 on approximating Independent Set for our semi-random model of perturbed graphs. In Section 3.1, we prove Lemma 4, which gives our tail bound on the size of the greedy independent set for perturbed graphs. Section 3.2 contains a proof for Lemma 5, which gives our tail bound on the largest eigenvalue of the auxiliary matrix we use to upper bound the independence number of a given perturbed graph. In Section 3.3, we combine our results from Sections 3.1 and 3.2 to present our approximation algorithm for Independent Set on our semi-random graph model. In Section 3.4, we prove Corollary 2, which gives our analysis of the expected behavior of GreedyColor for our semi-random graphs. In Section 3.5, we show what we can achieve for very small flip probabilities, and finally, in Section 3.6, we draw conclusions and discuss some open problems.

Throughout Chapter 3, when we write \( \ln x \), we mean the natural logarithm of \( x \), and when we write \( \log x \), we mean the base-2 logarithm of \( x \). (Of course, if \( \log x \) appears as a factor of a term in \( O \)-notation, the base does not matter. However, sometimes we care for the exact value of such a term, not only its asymptotical behavior, cf. e.g. Lemma 5.)
3.1 Greedy Coloring of Perturbed Graphs

We prove Lemma 4, which gives our tail bound on the size of the greedy independent set, i.e., the largest color class $\text{big}(C)$ in the coloring $C$ computed by GreedyColor (Algorithm 1). Remember that, given an adversary graph $G$ and a flip probability $\varepsilon > 0$, a semi-random graph $G$ is produced by our model $G(G, \varepsilon)$ by flipping each potential edge in $G$ independently with probability $\varepsilon$. Furthermore, remember that in (1.12), we defined the size of the independent set we can expect to find with high probability by $\text{gis}(G, \varepsilon, \delta) = \delta \frac{\ln n}{\varepsilon^2} \cdot \min\left\{\ln \frac{n}{\varepsilon}, 2\ln n \right\}$, where $\delta$ determines the range of the legal flip probabilities. Lemma 4 states the following.

Fix a constant $\delta \in (0, 1)$. Let $G = (V, E)$ be a graph and let $\varepsilon$ be arbitrary with $n^{-1-\delta} \leq \varepsilon \leq 1/2$. Let $G$ be a random graph from $G(G, \varepsilon)$, and let $C$ be the coloring computed by GreedyColor($G$). Then,

$$\Pr[|\text{big}(C)| < \text{gis}(G, \varepsilon, \delta)] \leq e^{-n \ln n}.$$ 

Proof of Lemma 4. In Section 2.1, we proved Lemma 1, which gives our tail bound on the greedy independent set size for random hypergraphs. We adapt the proof of the lemma. The difference is that here, we have two types of edge probabilities instead of one, which we have to consider in the proof.

Let $G = (V, E)$ be the random graph drawn from $G(G, \varepsilon)$. Again, let $s := \text{gis}(G, \varepsilon, \delta)$ denote the threshold for the largest color class, let $t := n/(2s)$, and let $\mathcal{B}$ denote the bad event that $|\text{big}(C)| \leq s$. As before, we prove that $\Pr[\mathcal{B}] \leq e^{-n \ln n}$.

The proof of Lemma 1 works unchanged except the part which determines the probability that a fixed partial vertex coloring $D = (D_1, \ldots, D_t)$ of color classes $D_i$ with at most $s$ vertices each is bad. Remember that $D$ is bad if and only if for every vertex $v \in \overline{D} = V \setminus (D_1 \cup \ldots \cup D_t)$ and every color class $D_i \in D$, there is an edge $\{v, w\} \in E$ connecting $v$ with some vertex $w \in D_i$. Having determined the probability that $D$ is bad, we can complete our proof using $\Pr[\mathcal{B}] \leq \Pr[\exists \overline{D}: D \text{ is bad}]$ for the set $\overline{D}$ of all partial vertex colorings.

Let us estimate the probability that a fixed partial vertex coloring $D = (D_1, \ldots, D_t)$ is bad. Again, by choice of parameters, the total number of vertices in $\overline{D}$ is at least $n/2$. For a vertex $v \in \overline{D}$ and a class $D_i$, let $v_D$ be the number of vertices $w \in D_i$ such that the edge $\{v, w\}$ is contained in the original (non-random) edge set $E$ of the adversary graph $G$. Then, the number of vertices in $D_i$ to which $v$ is not adjacent in the graph $G$ is $|D_i| - v_D$. For vertices $v \neq w$, the edge $e = \{v, w\}$ is in the random edge set $E$ with probability $1 - \varepsilon$ if $e \in E$ and with probability $\varepsilon$ if $e \notin E$. We observe that for a fixed vertex $v$ and a fixed class $D_i$, the probability that
3.1. Greedy Coloring of Perturbed Graphs

the semi-random graph $G$ contains an edge connecting $v$ to some vertex in $D_i$ is

$$1 - (1 - \varepsilon)^{|D_i| - v(v,i)}e^{v(v,i)} .$$

We exploit that the edges are drawn independently. This and $1 - x \leq e^{-x}$ for $x \in \mathbb{R}$ yields

$$\Pr[D \text{ is bad}] \leq \prod_{v \in D} \prod_{i = 1}^t (1 - (1 - \varepsilon)^{|D_i| - v(v,i)}e^{v(v,i)})$$

$$\leq \exp \left( - \sum_{v \in D} \sum_{i = 1}^t (1 - \varepsilon)^{|D_i| - v(v,i)}e^{v(v,i)} \right)$$

$$\leq \exp \left( - \sum_{v \in D} \sum_{i = 1}^t (1 - \varepsilon)e^{-v(v,i)}e^{v(v,i)} \right) ,$$

(3.1)

where we used $|D_i| \leq s$ for all classes $D_i \in D$ to derive (3.1). Consider the term $(1 - \varepsilon)e^{-v(v,i)}$ in the exponent of (3.1). Let $f(x) := (1 - \varepsilon)e^{-x}$ for $x \in \mathbb{R}$. The partial derivative of $f$ with respect to $x$ is

$$f'(x) = f(x) \cdot \ln(\varepsilon/(1 - \varepsilon)) \leq 0$$

since $\varepsilon \leq 1/2$, and its second derivative is

$$f''(x) = f(x) \cdot \ln^2(\varepsilon/(1 - \varepsilon)) \geq 0 .$$

(3.2)

Hence, $f(x)$ is a convex function. Without loss of generality, assume that $|D| = n/2$. For brevity, let $v_1, \ldots, v_{n/2}$ be the values $v(v,i)$, $v \in D$, $i = 1, \ldots, t$, in some arbitrary order. Clearly,

$$\hat{v} := \sum_{j = 1}^{n/2} v_j \leq |E| .$$

Jensen’s inequality for convex functions (see Theorem 14 in Appendix A.1) yields

$$\frac{1}{n/2} \cdot \sum_{j = 1}^{n/2} f(v_j) \geq f \left( \frac{1}{n/2} \cdot \sum_{j = 1}^{n/2} v_j \right) = f \left( \frac{\hat{v}}{n/2} \right) \geq f \left( \frac{2|E|}{n} \right),$$

(3.3)

using $\hat{v} \leq |E|$ and that $f(x)$ is monotonically decreasing. Using (3.1) and (3.3), we get

$$\Pr[D \text{ is bad}] \leq \exp \left( - \sum_{j = 1}^{n/2} f(v_j) \right)$$

$$\leq \exp \left( - \frac{n}{2} f \left( \frac{2|E|}{n} \right) \right)$$

$$= \exp \left( - \frac{n}{2} \cdot (1 - \varepsilon)^{-2|E|/n} e^{2|E|/n} \right) .$$

(3.4)
We show that the absolute value of the exponent in (3.4) is at least $2n \ln n$. For brevity, let $a := (1 - \varepsilon)^{-2|E|/(tm)}$ and $b := \varepsilon^{2|E|/(tn)}$. Then it suffices to show that $t \cdot a \cdot b \geq 4 \ln n$, or equivalently

$$
\ln t + \ln a + \ln b \geq \ln(4 \ln n) .
$$

The choice of $s \leq \frac{\delta \ln n}{4e}$ and $\varepsilon \geq n^{-(1-\delta)}$ in the lemma gives us

$$
s \leq \frac{\delta n^{1-\delta} \ln n}{16} .
$$

This yields

$$
\ln t = \ln \left( \frac{n}{2s} \right) \\
\geq \ln \left( \frac{8n^\delta}{\delta \ln n} \right) \\
= (\delta \ln n) - o(\ln n) \\
\geq (\delta/2) \cdot \ln n .
$$

Using $s \leq \frac{\delta \ln n}{4e}$ again, we can infer that

$$
\ln a = \ln \left( (1 - \varepsilon)^{-2|E|/(tm)} \right) \\
\geq \ln((1 - \varepsilon)^s) \\
\geq \ln(e^{-2\varepsilon s}) \\
= -2\varepsilon s \\
\geq -(\delta/8) \ln n ,
$$

where we used that $1 - x \geq e^{-2x}$ for $x \leq 1/2$. Moreover, $s \leq \frac{\delta n^{1-\delta} \ln n}{16|E| \ln(1/\varepsilon)}$ yields

$$
\ln b = \ln \left( \varepsilon^{2|E|/(tn)} \right) \\
= \frac{2|E|}{\ln n} \cdot \ln \varepsilon \\
= \frac{4|E|s}{n^2} \cdot \ln \varepsilon \\
\geq \frac{4|E| \ln \varepsilon \cdot \delta n^2 \ln n}{n^2} \cdot \frac{\delta n^2 \ln n}{16|E| \ln(1/\varepsilon)} \\
= -(\delta/4) \ln n .
$$

Finally, (3.7), (3.8), and (3.9) lead to

$$
(\ln t) + (\ln a) + (\ln b) \geq (\delta/2 - \delta/8 - \delta/4) \ln n \\
= (\delta/8) \ln n \\
= \omega(\ln n) \\
\geq \ln(4 \ln n) .
$$
This proves (3.5). Combining (3.5) with (3.4) yields

\[ \Pr[D \text{ is bad}] \leq \exp(-2n \ln n). \quad (3.10) \]

We have \( s = o(n) \leq (n - 1)/2 \) for sufficiently large \( n \) by (3.6). In the proof of Lemma 1, we showed that under this condition, the total number \( |\hat{D}| \) of partial vertex colorings fulfills \( |\hat{D}| \leq e^{\ln n} \). Using this and (3.10), we take a union bound over all possible partial vertex colorings to get \( \Pr[B] \leq e^{-n \ln n}. \)

\[ \square \]

### 3.2 Upper Bounding the Independence Number

In this section, we prove Lemma 5, which we use to upper bound the independence number of the given semi-random graph \( G \) in our approximation algorithm AdversaryIndSet (Algorithm 9). In proving our results, we exploit some properties of eigenvalues of real, symmetric matrices. For precise definitions of terms, some known facts, and proofs of some properties, we refer the reader to Appendix A.2.

Our proof is an adaption of the spectral technique used by Krivelevich and Vu [KV02], who compute an upper bound on \( \alpha(G) \) for \( G \) drawn from \( G(n, p) \), using the largest eigenvalue of a suitable auxiliary matrix \( M \) (see our description in Section 1.2.1 for details). Recall that in Section 1.2.2, we gave the following definition (cf. (1.13)):

For an adversarial graph \( G = (V, E) \), a flip probability \( \varepsilon > 0 \), and a random graph \( G = (V, E) \) drawn from \( G(G, \varepsilon) \), the \( n \times n \)-matrix \( A = A(G, \varepsilon) = (a_{ij})_{1 \leq i, j \leq n} \) is defined as

\[
a_{ij} = \begin{cases} 
1 & \text{if } e = \{i, j\} \notin E \text{ and } \\
-(1 - p_e)/p_e & \text{if } e = \{i, j\} \in E.
\end{cases} \quad (3.11)
\]

Again, notice that \( a_{ij} \) depends on whether \( e = \{i, j\} \in E \) and whether \( e \in E \), since the latter determines the probability \( p_e \) that \( e \in E \) as discussed above. We restate Lemma 5 and prove it.

**Fix a graph \( G \) and \( \varepsilon = \varepsilon(n) \leq 1/2 \) with \( \varepsilon = \Omega((\log n)^2/n). \) Let \( \mathcal{G} \) be drawn from \( \mathcal{G}(G, \varepsilon) \), and let \( A := A(\mathcal{G}, \varepsilon) \). Then, \( \alpha(\mathcal{G}) \leq \lambda_1(A) \). Furthermore,**

\[
E[\lambda_1(A)] \leq 2^7 \cdot (\log n) \cdot (n/\varepsilon)^{1/2}
\]

and

\[
\Pr[\lambda_1(A) \geq 2^8 \cdot (\log n) \cdot (n/\varepsilon)^{1/2}] \leq 4 \cdot \exp(-2^9 \cdot n \varepsilon \cdot (\log n)^2).
\]
A proof of the first claim of the lemma, $\alpha(G) \leq \lambda_1(A)$ for all perturbed graphs $G$, is presented in the appendix, Section A.2.1. Below, we give proofs for the other two claims.

Our final goal in proving Lemma 5 is to prove its third claim, i.e., that with high probability, $\lambda_1(A(G))$ is $\mathcal{O}((\log n) \cdot (n/\epsilon)^{1/2})$ for a random $G(n, \varepsilon)$ graph $G$. This statement is exploited in the analysis of algorithm $\text{AdversaryIndSet}$. As mentioned, we apply the technique that Krivelevich and Vu [KV02] used to prove their result for $G(n, p)$ graphs. It is based on estimating the probability that the largest eigenvalue deviates significantly from its expectation. While in [KV02], a known result on $E[\lambda_1]$ of (the matrix for) $G(n, p)$ graphs by Füredi and Komlós [FK81] is used, we first must determine $E[\lambda_1(A(G))]$, which we do in Section 3.2.1. In Section 3.2.2, we transfer the result about the deviation of $\lambda_1$ from its expectation to our model.

### 3.2.1 The Expectation of the Largest Eigenvalue

For a matrix $A \in \mathbb{R}^{n \times n}$, the trace of $A$ is defined as

$$\text{tr}(A) = \sum_{i=1}^{n} a_{ii},$$

the sum of $A$’s main diagonal entries. To upper bound $E[\lambda_1(A(G))]$, we use the trace method (see Wigner [Wig55]) for estimating the eigenvalues of a matrix, which was also used by Füredi and Komlós in [FK81]. Roughly, the method is based on estimating the trace of a high power $A^k$ of $A$. More precisely, if $A$ is real and symmetric and $k \in \mathbb{N}$ is even, one can show that

$$\text{tr}(A^k) = \sum_{i=1}^{n} \lambda_i(A^k) \geq \lambda_1(A^k). \quad (3.12)$$

We give a proof of (3.12) in Section A.2.

Consider $\lambda_1(A)$. It is easy to see that its $k$-th power $\lambda_1(A)^k$ is an eigenvalue of $A^k$, and hence we have $\lambda_i(A^k) \geq \lambda_1(A)^k$. Together with (3.12), we get for a real symmetric matrix $A$ and even $k \in \mathbb{N}$

$$\lambda_1(A)^k \leq \text{tr}(A^k). \quad (3.13)$$

Consider $A = A(G, G, \varepsilon)$ for a random graph $G$ from $G(G, \varepsilon)$, which by definition is real and symmetric. For $k$ even, applying (3.13) yields $E[\lambda_1(A)^k] \leq E[\text{tr}(A^k)]$. Together with $E[\lambda_1(A)] \leq E[\text{tr}(A^k)]$ due to Jensen’s inequality for random variables (see Theorem 15 in Appendix A.1), we get $E[\lambda_1(A)]^k \leq E[\text{tr}(A^k)]$ or equivalently $E[\lambda_1(A)] \leq E[\text{tr}(A^k)]^{1/k}$. We obtain the following lemma.
Lemma 12. Fix an adversary graph $G$, a flip probability $\varepsilon > 0$, and $k \in \mathbb{N}$ even. Let $A := A(G, G, \varepsilon)$ for a random graph $G$ drawn from $G(G, \varepsilon)$. Then, $E[\lambda_1(A)] \leq E[\text{tr}(A^k)]^{1/k}$.

Using Lemma 12, we upper bound $E[\lambda_1(A(G))]$ for a random perturbation graph $G$ by upper bounding $E[\text{tr}(A(G)^k)]$ and taking the $k$-th root of the determined bound. This proves the claim about $E[\lambda_1(A(G))]$ in Lemma 5.

Fix an adversary graph $G$ on $n$ vertices, a flip probability $1/n \leq \varepsilon \leq 1/2$, and let $k \in \mathbb{N}$ be even. Let $G$ be a random graph from $G(G, \varepsilon)$ and $A := A(G, G, \varepsilon)$. For the sake of clarity, we abbreviate $B := A^k$. By definition of the matrix product, we have

$$b_{ij} = \sum_{l_1=1}^{n} \cdots \sum_{l_{k-1}=1}^{n} a_{l_1i}a_{l_1l_2} \cdots a_{l_{k-1}j}.$$ 

Together with $\text{tr}(A^k) = \sum_{l_0=1}^{n} b_{l_0l_0}$ and linearity of expectation, we get

$$E[\text{tr}(A^k)] = E \left[ \sum_{l_0=1}^{n} b_{l_0l_0} \right] = E \left[ \sum_{l_0=1}^{n} \sum_{l_1=1}^{n} \cdots \sum_{l_{k-1}=1}^{n} a_{l_0l_1}a_{l_1l_2} \cdots a_{l_{k-1}l_0} \right] = \sum_{l_0=1}^{n} \cdots \sum_{l_{k-1}=1}^{n} E[a_{l_0l_1}a_{l_1l_2} \cdots a_{l_{k-1}l_0}] = \sum_{l \in L} E[a_{l_0l_1}a_{l_1l_2} \cdots a_{l_{k-1}l_0}] , \tag{3.14}$$

abbreviating the set of the considered sequences of indices by $L := \{1, \ldots, n\}^k$. We fix a sequence $\vec{l} \in L$ and estimate the corresponding summand

$$E[a_{l_0l_1}a_{l_1l_2} \cdots a_{l_{k-1}l_0}] \tag{3.15}$$

in (3.14). In the product, different entries $a_{ij}$ of $A$ appear. For $i \leq j$, since $A$ is symmetric, we identify the equally-valued entries $a_{ij}$ and $a_{ji}$ and consider $a_{ij}$ as the representative for both. Let $a_{i_1j_1}, \ldots, a_{i_mj_m}$ be the representative entries in (3.15) with multiplicities $r_1, \ldots, r_m \geq 1$, respectively. Since the edges $\{i, j\}$ in our random graph $G$ are inserted independently, the same holds for their corresponding representative entries $a_{ij}$. Using linearity of expectation for products of independent random variables, we get for (3.15)

$$E[a_{l_0l_1}a_{l_1l_2} \cdots a_{l_{k-1}l_0}] = E \left[ \prod_{s=1}^{m} a_{i_sj_s}^{r_s} \right] = \prod_{s=1}^{m} E \left[ a_{i_sj_s}^{r_s} \right] ,$$
and using this, we can rewrite (3.14) as

\[ E[\text{tr}(A^k)] = \sum_{l \in L} E[a_{l_0,l_1} \ldots a_{l_{k-1}l_0}] = \sum_{l \in L} \prod_{s=1}^{m} E[a_{i,s}^{r_s}] \]  \hspace{1cm} (3.16)

To achieve our goal of upper bounding \( E[\text{tr}(A^k)] \), we upper bound the right hand side term in (3.16).

First, consider the sequences \( \tilde{l} \in L \) for which all appearing representative entries lie on the main diagonal. Clearly, this happens if and only if \( l_0 = \ldots = l_{k-1} = i \) for some \( i \in \{1, \ldots, n\} \). For such \( \tilde{l} \), the corresponding summand in (3.16) is

\[ \prod_{s=1}^{m} E[a_{i,s}^{r_s}] = E[a_{i,i}^1] = E[1^k] = 1 , \]

since by definition of \( A \), main diagonal entries always have value 1. We conclude that the \( n \) summands corresponding to the sequences of the form \( l_0 = \ldots = l_{k-1} = i \) contribute \( n \) to (3.16).

Now, consider the sequences \( \tilde{l} \in L \) choosing at least one off-diagonal entry. If an off-diagonal representative entry \( a_{i,j} \) with multiplicity \( r_s = 1 \) appears, the corresponding summand \( \prod_{s=1}^{m} E[a_{i,s}^{r_s}] \) in (3.16) becomes zero: By definition of \( A(\mathcal{G}) \) in (1.13), it holds for the entry that

\[ E[a_{i,j}^{r_s}] = E[a_{i,j}^1] = 1 \cdot (1 - p_e) - \frac{1-p_e}{p_e} \cdot p_e = 0 \]

for the potential edge \( e = \{i, j\} \). Thus, \( \prod_{s=1}^{m} E[a_{i,s}^{r_s}] = 0 \), and we can ignore this summand in (3.16). We conclude that it suffices to consider the set \( L' \) of sequences with at least one off-diagonal entry and every such entry appearing at least twice, which we do in the following.

We view a sequence \( \tilde{l} \in L' \) as a closed walk \( l_0, l_1, \ldots, l_{k-1}, l_k = l_0 \) of length \( k \) in a complete undirected graph on \( n \) vertices \( 1, \ldots, n \). A step \((l_j, l_{j+1})\) is identical if \( l_j = l_{j+1} \) (we stay at vertex \( l_j \)) and real otherwise. Notice that the entry \( a_{l_jl_{j+1}} \) belonging to a step \((l_j, l_{j+1})\) is off-diagonal if and only if the step is real. Let \( k' \) be the number of real steps and \( m' \) be the number of different (undirected) edges passed in the walk (no edge is passed in identical steps). Then, \( 2 \leq k' \leq k \) and \( 1 \leq m' \leq k'/2 \), since every of the \( m' \) edges is passed at least twice: It corresponds to an off-diagonal representative entry in \( A \), each of which is chosen at least twice by \( \tilde{l} \in L' \).

We count the number of different closed walks of length \( k' \), given values for \( k' \) and \( m' \). For the positions of the \( k - k' \) identical steps, we have \( \binom{k}{k-k'} \leq 2^k \) choices. It remains to choose a closed walk of length \( k' \) with real steps only, with each of the \( m' \) passed edges appearing at least twice.
3.2. Upper Bounding the Independence Number

Friedman, Goerdt, and Krivelevich [FGK05] proved an upper bound on the number of such walks (called “duplicated” in [FGK05]). The arguments are as follows. For a walk \( l_0, l_1, \ldots, l_{k-1}, l_k = l_0 \), let \( e_1, \ldots, e_k \) be the sequence of passed edges in the graph. (Thus, \( e_1 = \{l_0, l_1\}, e_2 = \{l_1, l_2\} \), and so on.) Every possible closed walk of length \( k' \), with only real steps and each of the \( m' \) passed edges appearing at least twice, is generated at least once by the following process:

1. Choose the \( m' \) positions where each of the \( m' \) different edges passed in the walk appears for the first time in \( e_1, \ldots, e_k \). We have at most \( \binom{k'}{m'} \leq 2^{k'} \) possible choices here.

2. For the remaining \( k' - m' \) edges \( e_i \) in \( e_1, \ldots, e_k \), choose which of the preceding first occurrences of an edge is to be used as \( e_i \). We have at most \( (m')^{k' - m'} \leq (k')^{k'} \) possible choices here.

3. Choose the vertices of the edges chosen in Step 1 of the process. Since these \( m' \) edges must induce a connected subgraph, we have at most \( n^{m'+1} \) possible choices here.

It follows that the total number of walks is at most

\[
2^{k'} \cdot (k')^{k'} \cdot n^{m'+1} \leq 2^k \cdot k^k \cdot n^{m'+1} ,
\]

where we used \( m' \leq k' \leq k \). Together with the \( 2^k \) choices for the positions of the identical steps, we infer that the total number of different walks given \( k', m' \) is at most

\[
2^k \cdot 2^k \cdot k^k \cdot n^{m'+1} = 2^{2k} \cdot k^k \cdot n^{m'+1} . \tag{3.17}
\]

For a walk \( \vec{l} \in L' \) with parameters \( k', m' \), we estimate its summand \( \prod_{s=1}^{n} \mathbb{E}[a_{i_sj_s}] \) in (3.16). Since diagonal entries \( a_{i_ij_j} \) always have value 1, their factors \( \mathbb{E}[a_{i_ij_j}] = 1 \) can be omitted. For an off-diagonal representative entry \( a_{i_ij_j} \) and the corresponding potential edge \( e = \{i_s, j_s\} \), by (1.13)

\[
\mathbb{E}[a_{i_ij_j}] = 1^r \cdot (1 - p_e) + \left(-\frac{1 - p_e}{p_e}\right)^r \cdot p_e \\
\leq 1 + \frac{1}{p_e^{r-1}} \\
\leq \frac{2}{p_e^{r-1}} \\
\leq \frac{2}{\varepsilon^{r-1}} , \tag{3.18}
\]

since \( \varepsilon \leq p_e \leq 1 - \varepsilon \) for all potential edges \( e \). Observe that our estimation \( p_e \geq \varepsilon \) used for all potential edges \( e \) in the final inequality in (3.18) neglects the potential
edges $e$ which are actually contained in the adversary graph $G$. For such an $e$, we have $p_e = 1 - \varepsilon > \varepsilon$, which could potentially improve (3.18) and hence our final result. Nevertheless, asymptotically we lose nothing: Assume that $G$’s edges form a clique of size $n/2$. Then, $|E| = \Theta(n^2)$ but $G$ still contains an independent set of size $n/2$. This part of our random graph $\mathcal{G}$ behaves as $G(n/2, \varepsilon)$. Thus, we cannot expect to finally get a better bound than for $G(n/2, \varepsilon)$, for which our approach asymptotically yields the same result.

We return to our proof. W.l.o.g., assume the off-diagonal representative entries $a_{i,j}$ have indices $s = 1, \ldots, m'$. Then,

$$\sum_{s=1}^{m'} (r_s - 1) = \left( \sum_{s=1}^{m'} r_s \right) - m' = k' - m', \quad (3.19)$$

since each of the $k'$ real steps corresponds to an off-diagonal entry. Now, (3.19) together with (3.18) yields

$$\prod_{s=1}^{m} E[a_{i,s,j,s}] = \prod_{s=1}^{m'} E[a_{i,s,j,s}] \leq \prod_{s=1}^{m'} \frac{2}{\varepsilon^{r_s-1}} = \frac{2^{m'}}{\varepsilon^{\sum_{s=1}^{m'} (r_s-1)}} = \frac{2^{m'}}{\varepsilon^{k'-m'}}. \quad (3.20)$$

We can now estimate the contribution of the sequences $\mathbf{i} \in L'$ to (3.16). For given parameters $k', m'$, remember that the number of corresponding walks $\mathbf{i}$ is at most $2^{2k} \cdot k^k \cdot n^{m'+1}$ due to (3.17). We now sum up over all possibilities for $k', m'$ and get using (3.20)

$$\sum_{\mathbf{i} \in L'} \prod_{s=1}^{m} E[a_{i,s,j,s}] \leq \sum_{k' \geq 2} \sum_{m' = 1}^{k'/2} 2^{2k} \cdot k^k \cdot n^{m'+1} \cdot \frac{2^{m'}}{\varepsilon^{k'-m'}} \leq \sum_{k' \geq 2} \sum_{m' = 1}^{k'/2} 2^{3k} \cdot k^k \cdot n \cdot \left( \frac{n}{\varepsilon^{k/2}} \right) \leq 2^{4k} \cdot k^k \cdot n \cdot \left( \frac{n}{\varepsilon^{k/2}} \right), \quad (3.21)$$

using that $2 \leq k' \leq k$ and $1 \leq m' \leq k'/2$, which implies

$$2^{m'} \leq 2^k \quad \text{and} \quad \frac{n^{m'+1}}{\varepsilon^{k'-m'}} \leq n \cdot \left( \frac{n}{\varepsilon^{k/2}} \right). \quad (3.22)$$

and

$$2^{m'} \leq 2^k \quad \text{and} \quad \frac{n^{m'+1}}{\varepsilon^{k'-m'}} \leq n \cdot \left( \frac{n}{\varepsilon^{k/2}} \right). \quad (3.23)$$
For the latter inequality in (3.23), observe that

\[
\frac{n^{m'+1}}{\varepsilon^{k'-m'}} \leq n \cdot \left(\frac{n}{\varepsilon}\right)^{k/2}
\]

\[
\Leftrightarrow n^{m'-k/2} \leq \varepsilon^{k'-m'-k/2}.
\] (3.24)

With \(k' \leq k\) and \(0 < \varepsilon < 1\), it follows that

\[
\varepsilon^{k'-m'-k/2} \geq \varepsilon^{-(m'-k/2)}.
\]

Using this, to fulfill (3.24), it suffices to fulfill

\[
n^{m'-k/2} \leq \varepsilon^{-(m'-k/2)},
\]

or equivalently,

\[
(n\varepsilon)^{m'-k/2} \leq 1.
\] (3.25)

Since \(m' \leq k'/2 \leq k/2\), we have \(m' - k/2 \leq 0\) in the exponent of (3.25), and hence, (3.25) follows from \(\varepsilon \geq 1/n\), we assumed above.

Furthermore, to get (3.22), we used in that in (3.21), we have at most \(k^2 \leq 2^k\) summands.

We can now upper bound (3.16): Above, we showed that the contribution of the sequences \(\vec{l} \in L\) choosing only main diagonal entries is \(n\). The remaining sequences are the ones in \(L'\), and their contribution is given by (3.22). Using (3.16), we get

\[
E[\text{tr}(A^k)] = \sum_{\vec{l} \in L} \prod_{s=1}^{m} E[a_{l_{i,j},i,j}]
\]

\[
\leq n + 2^{4k} \cdot k^k \cdot n \cdot \left(\frac{n}{\varepsilon}\right)^{k/2}
\]

\[
\leq 2^{5k} \cdot k^k \cdot n \cdot \left(\frac{n}{\varepsilon}\right)^{k/2},
\] (3.26)

using that clearly \(n \leq 2^{4k} \cdot k^k \cdot n \cdot (n/\varepsilon)^{k/2}\). Now, we set \(k := 2[\log n]\) and apply Lemma 12 to (3.26), which yields

\[
E[\lambda_1(A)] \leq E\left[\text{tr}(A^k)\right]^{1/k}
\]

\[
\leq \left(2^{5k} \cdot k^k \cdot n \cdot \left(\frac{n}{\varepsilon}\right)^{k/2}\right)^{1/k}
\]

\[
= 2^{5} \cdot k \cdot n^{1/k} \cdot \left(\frac{n}{\varepsilon}\right)^{1/2}
\]

\[
\leq 2^7 \cdot (\log n) \cdot \left(\frac{n}{\varepsilon}\right)^{1/2}.
\] (3.27)

To derive (3.27), we used that \(n^{1/k} = n^{1/(2[\log n])} \leq 2^{1/2}\) and \(k = 2[\log n] \leq 2^{3/2}(\log n)\), yielding

\[
2^5 \cdot k \cdot n^{1/k} \leq 2^5 \cdot 2^{3/2} \cdot (\log n) \cdot 2^{1/2} = 2^7 \cdot \log n.
\]
We have proved the claim $E[\lambda_1(A)] \leq 2^7 \cdot (\log n) \cdot (n/\varepsilon)^{1/2}$ in Lemma 5. Observe that the lemma assumes $\varepsilon = \Omega((\log n)^2/n)$ while $\varepsilon \geq 1/n$ suffices in the above arguments. The reason is that for $\varepsilon = 1/n$, the upper bound on $E[\lambda_1(A)]$ becomes $\Theta(n \log n)$ and is hence trivial for upper bounding the independence number. For the range of $\varepsilon$ in the lemma, it is at most $O(n)$.

### 3.2.2 A Tail Bound on the Largest Eigenvalue

In this section, we prove that under the assumptions of Lemma 5,

$$
\Pr[\lambda_1(A) \geq 2^8 \cdot (\log n) \cdot (n/\varepsilon)^{1/2}] \leq 4 \cdot \exp(-2^9 \cdot n \varepsilon \cdot (\log n)^2)
$$

for the matrix $A = A(G, G, \varepsilon)$ of a random graph $G$ drawn from $G(G, \varepsilon)$, as claimed in the lemma. This finishes our proof of Lemma 5.

As mentioned, Krivelevich and Vu proved in [KV02] that for a random graph $G = (V, E)$ from $G(n, p)$,

$$
\Pr[\lambda_1(M) \geq 4(n/p)^{1/2}] \leq 2^{-np/8}
$$

for their auxiliary matrix $M = M(G, p)$. We adapt their proof to our matrix $A(G)$. The original proof and the one for our matrix $A$ are almost identical, and we could simply refer the reader to [KV02]. However, the proof in [KV02] is very concise, and for the sake of self-containedness and readability, we sketch it here.

The proof in [KV02] is based on estimating the probability that $\lambda_1$ largely deviates from its median, using Talagrand’s inequality [Tal95]. In a form that is convenient in our notation, the inequality states the following:

**Theorem 13 (Talagrand’s inequality).** Let $T_1, \ldots, T_r$ be independent random variables, and let $S$ be the product space (with the product measure) generated by $T_1, \ldots, T_r$. For $B \subseteq S$ and $t \geq 0$, let

$$
B_t = \left\{ \tilde{x} \in S \mid \forall \tilde{a} \in \mathbb{R}^r: \exists \tilde{y} \in B: \sum_{x_i \neq y_i} |a_i| \leq t \cdot \left( \sum_{i=1}^r a_i^2 \right)^{1/2} \right\}.
$$

Then,

$$
\Pr[\bar{B}_t] \cdot \Pr[B] \leq e^{-t^2/4}.
$$

Recall our definition (cf. (1.13)) of the matrix $A = A(G, G, \varepsilon)$ such that $a_{ij} = 1$ for $\{i, j\} \notin E$, and $a_{ij} = \frac{1-p_e}{p_e}$ for $\{i, j\} \in E$. (The crucial difference in our proof compared to the one in [KV02] is the following: Since $p_e$ can be either $\varepsilon$ or $1-\varepsilon$, there are two types of entries $a_{ij}$ in our matrix $A$. Since in $G(n, p)$, every potential edge has the same probability of appearing, there is only one type of entry in $M$.)
3.2. Upper Bounding the Independence Number

We want to apply Talagrand’s inequality to get our upper bound on the probability that $\lambda_1(A)$ largely deviates from its median. First, notice that a random matrix $A(G)$ is specified by the above-diagonal entries $a_{ij}$, $1 \leq i < j \leq n$, which are independent random variables since the edges in our random graph are chosen independently. Hence, we can apply Talagrand’s inequality, with the above-diagonal entries in the role of $T_1, \ldots, T_r$. Let $m$ be the median of $\lambda_1(A(G))$, and let $B$ be the set of all matrices $A$ for which $\lambda_1(A) \leq m$. Then, $\Pr[B] = 1/2$.

Let $d > 0$ and consider the event “$\lambda_1(A) \geq m + d$,” i.e., $\lambda_1(A)$ deviates from its median $m$ by at least $d$ to above. Let $A^0$ be a matrix with $\lambda_1(A^0) \geq m + d$. By the Rayleigh-Ritz theorem (see Theorem 16 in Section A.2.1), we can infer that there is a vector $\vec{x} \in \mathbb{R}^n$ of length 1 such that

$$m + d \leq \vec{x}^T A^0 \vec{x} .$$

(3.30)

Using that $\vec{x}$ has unit length, we get

$$\vec{x}^T A^0 \vec{x} = \sum_{1 \leq i < j \leq n} x_i x_j a_{ij}^0$$

$$= 1 + \sum_{1 \leq i < j \leq n} 2x_i x_j a_{ij}^0 ,$$

which together with (3.30) yields

$$m + d \leq 1 + \sum_{1 \leq i < j \leq n} 2x_i x_j a_{ij}^0 .$$

(3.31)

On the other hand, again by the Rayleigh-Ritz theorem, for every matrix $A^1 \in B$, since $\lambda_1(A^1) \leq m$ by definition of $B$, we have

$$m \geq 1 + \sum_{1 \leq i < j \leq n} 2x_i x_j a_{ij}^1 .$$

(3.32)

With (3.31) and (3.32), we conclude that

$$\sum_{1 \leq i < j \leq n} 2x_i x_j \cdot (a_{ij}^0 - a_{ij}^1) \geq d ,$$

and hence clearly

$$\sum_{1 \leq i < j \leq n} |2x_i x_j| \cdot |a_{ij}^0 - a_{ij}^1| \geq d .$$

(3.33)

We want to apply Talagrand’s inequality and need to show that $A^0 \in \overline{B}$ for a suitable $t > 0$. We construct such a $t$ now. To this end, we create a vector $\vec{a}$ by
Chapter 3. Independent Set for Semi-Random Graphs

letting \( \alpha_{ij} = 2x_ix_j \) for \( 1 \leq i < j \leq n \). (Confer \( \vec{\alpha} \) in (3.28).) By construction of \( A(G) \), for all indices \( 1 \leq i < j \leq n \), we have

\[
|a_{ij}^0 - a_{ij}^1| \leq 1 + \frac{1 - p_e}{p_e} = 1/p_e \leq 1/\varepsilon
\]

since \( p_e \geq \varepsilon \) for all potential edges \( e \). Therefore, with (3.33) it follows using \( \alpha_{ij} = 2x_ix_j \) that

\[
\sum_{a_{ij}^0 \neq a_{ij}^1} |\alpha_{ij}| \geq d\varepsilon \ . \quad (3.34)
\]

It is easy to see that \( \sum_{1 \leq i < j \leq n} \alpha_{ij}^2 \leq 2 \cdot \left( \sum_{1 \leq i < j \leq n} \alpha_{ij}^2 \right)^2 = 2 \), and hence

\[
\left( \sum_{1 \leq i < j \leq n} \alpha_{ij}^2 \right)^{1/2} \leq \sqrt{2} \ . \quad (3.35)
\]

Now, using (3.35), we can conclude with (3.34) that

\[
\sum_{a_{ij}^0 \neq a_{ij}^1} |\alpha_{ij}| \geq \frac{d\varepsilon}{\sqrt{2}} \cdot \left( \sum_{1 \leq i < j \leq n} \alpha_{ij}^2 \right)^{1/2} \ . \quad (3.36)
\]

Now, let \( t = d\varepsilon / \sqrt{2} \), and consider (3.28) in Theorem (13). With (3.36), we can now conclude that \( \vec{\alpha} \) is a witness that \( A^0 \in \overline{B_t} \). Now, using \( \Pr[B] = 1/2 \) since \( m \) is the median of \( \lambda_1 \), applying Talagrand’s inequality (3.29) yields

\[
\Pr[\lambda_1(A) \geq m + d] \leq \Pr[\overline{B_t}] \leq \frac{1}{\Pr[B]} \cdot e^{-t^2/4} = 2e^{-\frac{(d\varepsilon)^2}{8}} \ . \quad (3.37)
\]

In an analogous way to above, one can show that

\[
\Pr[\lambda_1(A) \leq m - d] \leq 2e^{-\frac{(d\varepsilon)^2}{8}} \ . \quad (3.38)
\]

Then, combining (3.37) and (3.38) yields

\[
\Pr[|\lambda_1(A) - m| \geq d] \leq 4e^{-\frac{(d\varepsilon)^2}{8}} \ . \quad (3.39)
\]

Furthermore, using the sharp concentration result (3.39), one can show that

\[
|E[\lambda_1(A)] - m| = O(1/\varepsilon) \ . \quad (3.40)
\]
3.3. Approximating the Independence Number

(The proof of (3.40) is easy and does not yield much new insight into our actual problem. For a clear presentation of only the central ideas, we omit it here. We refer the reader to [KV02], where the proof is presented for $p = \varepsilon$.)

By choice of $\varepsilon = \Omega((\log n)^2/n)$, we have

$$\frac{1}{\varepsilon} = \frac{1}{(\log n) \cdot (n/\varepsilon)^{1/2}} = \frac{1}{\varepsilon^{1/2} \cdot n^{1/2} \log n} = O(1/(\log n)^2) = o(1).$$

Together with (3.40) we get

$$|E[\lambda_1(A)] - m| = O(1/\varepsilon) = o(1) \cdot (\log n) \cdot (n/\varepsilon)^{1/2}.$$

This, together with $E[\lambda_1(A)] \leq 2^7 \cdot (\log n) \cdot (n/\varepsilon)^{1/2}$ from Lemma 5, yields that

$$m \leq E[\lambda_1(A)] + o(1) \cdot (\log n) \cdot (n/\varepsilon)^{1/2} \leq (1 + o(1)) \cdot 2^7 \cdot (\log n) \cdot (n/\varepsilon)^{1/2}. \quad (3.41)$$

Now assume that $\lambda_1(A) \geq 2^8 \cdot (\log n) \cdot (n/\varepsilon)^{1/2}$ occurs. With (3.41) we get

$$|\lambda_1(A) - m| \geq 2^6 \cdot (\log n) \cdot (n/\varepsilon)^{1/2} =: d.$$

Using this value of $d$, with (3.39), we finish our proof of the tail bound on $\lambda_1(A)$ in Lemma 5 by observing

$$\Pr[\lambda_1(A) \geq 2^8 \cdot (\log n) \cdot (n/\varepsilon)^{1/2}]$$

$$\leq \Pr[[\lambda_1(A) - m| \geq d]$$

$$\leq 4 \cdot e^{-d^2/8}$$

$$= 4 \cdot \exp(-2^9 \cdot n \varepsilon \cdot (\log n)^2),$$

which finishes our proof. \qed

3.3 Approximating the Independence Number

In this section, we present our approximation algorithm $\text{AdversaryIndSet}(G, \varepsilon)$ for semi-random graphs $G$ drawn from the $G(G, \varepsilon)$ model. Before, recall our definition of the non-neighborhood of a set of vertices in a graph, Definition 5 in Section 2.2: For a graph $G = (V, E)$ and $S \subseteq V$, the non-neighborhood $\overline{N}(S)$ of $S$ contains all vertices $v \in V \setminus S$ for which there is no edge $\{v, w\} \in E$ with $w \in S$.

Algorithm $\text{AdversaryIndSet}$ gets an adversary graph $G$, a flip probability $\varepsilon$, and a random graph $G$ drawn from $G(G, \varepsilon)$ as inputs. It uses the value $gis(G, \varepsilon, \delta)$ defined in (1.12) as a threshold in its greedy step. The algorithm looks as follows.
The same holds if Step 3 outputs big($C$), with our bounds on $\alpha$. Since (8 log $n$)/$\varepsilon$, the non-neighborhood has size $|\overline{N}(S')|$ ≤ (2 log $n$)($n/\varepsilon$)$^{1/2}$, and hence

$$\alpha(G) \leq (8 \log n)/\varepsilon + (2 \log n)(n/\varepsilon)^{1/2} = O((\log n)(n/\varepsilon)^{1/2}) \ .$$

since $\varepsilon \geq n^{-1/2}$. For Step 4, this upper bound on $\alpha(G)$ is obvious if big($C$) is output.

We next prove Theorem 7, giving our analysis of AdversaryIndSet’s performance. We restate the theorem for convenience.

Fix a graph $G$ and a flip probability $n^{-1/2} \leq \varepsilon \leq 1/2$. Let $\mathcal{G}$ be drawn from $\mathcal{G}(G, \varepsilon)$. Then, AdversaryIndSet($\mathcal{G}$, $G$, $\varepsilon$) has polynomial expected running time. Furthermore, if $\varepsilon$ is sufficiently high, i.e., $\frac{\log(1/\varepsilon)}{\varepsilon} \leq \frac{n^2}{|E|}$, it has approximation guarantee $O((n\varepsilon)^{1/2})$. Otherwise, the approximation guarantee is $O\left(\frac{\log(1/\varepsilon)}{|E|} \cdot \frac{\log(1/\varepsilon)}{\varepsilon^{1/2}}\right)$.

**Proof of Theorem 7.** Our proof follows the same approach as the one for Theorem 2 in Section 2.2. We start with the approximation guarantee. As an intermediate goal, we show that in any case, we compute a solution with approximation ratio

$$O\left(\frac{(\log n)(n/\varepsilon)^{1/2}}{\text{gis}(G, \varepsilon, 1/2)}\right).$$

Using this, we can prove that we achieve the desired approximation guarantee.

The solution in Step 5 is optimal and hence even has approximation ratio 1. If any other step outputs its solution big($C$), it has size at least gis($G, \varepsilon, 1/2$), since otherwise Step 1 switches to exhaustive search and jumps to Step 5. On the other hand, the optimum is not too large if big($C$) is output: If Step 2 outputs it, with $\alpha(\mathcal{G}) \leq \lambda_1(A(\mathcal{G}))$ from Lemma 5, it follows that

$$\alpha(G) \leq \lambda_1(A(\mathcal{G})) < 2^8 \cdot (\log n) \cdot (n/\varepsilon)^{1/2} = O((\log n) \cdot (n/\varepsilon)^{1/2}) \ .$$

The same holds if Step 3 outputs big($C$): Then, for all sets $S' \subseteq V$ of size $(8 \log n)/\varepsilon$, the non-neighborhood has size $|\overline{N}(S')| \leq (2 \log n)(n/\varepsilon)^{1/2}$, and hence

$$\alpha(G) \leq (8 \log n)/\varepsilon + (2 \log n)(n/\varepsilon)^{1/2} = O((\log n)(n/\varepsilon)^{1/2}) \ .$$

We start with the approximation guarantee. As an intermediate goal, we show that in any case
we achieve an approximation ratio of

$$\frac{\alpha(G)}{|\text{big}(C)|} = O\left(\frac{(\log n)(n/\varepsilon)^{1/2}}{\text{gis}(G, \varepsilon, 1/2)}\right).$$

(3.42)

We complete our proof of the approximation guarantee. The theorem claims two approximation guarantees, depending on whether the flip probability $\varepsilon$ is sufficiently high or not. We consider the two cases. Recall the definition of

$$\text{gis}(G, \varepsilon, \delta) = \delta \cdot \min\left\{\frac{\ln n}{\varepsilon}, \frac{n^2 \ln n}{|E| \ln(1/\varepsilon)}\right\}$$

in (1.12). If $\varepsilon$ is small, i.e., $\frac{\ln(1/\varepsilon)}{\varepsilon} > \frac{n^2}{|E|}$, we have

$$\text{gis}(G, \varepsilon, 1/2) = \Omega\left(\frac{n^2 \log n}{|E| \log(1/\varepsilon)}\right),$$

since the right hand side term determines the minimum in (1.12). Therefore, (3.42) yields an approximation ratio of

$$O\left(\frac{(\log n)(n/\varepsilon)^{1/2}}{|E| \log(1/\varepsilon)}\right) = O\left(\frac{|E|}{n^{3/2}} \cdot \frac{\log(1/\varepsilon)}{\varepsilon^{1/2}}\right)$$

as claimed in the theorem. If the flip probability $\varepsilon$ is in contrast sufficiently high, i.e., $\frac{\ln(1/\varepsilon)}{\varepsilon} \leq \frac{n^2}{|E|}$, it follows that $\text{gis}(G, \varepsilon, 1/2) = \Omega((\log n)/\varepsilon)$. With (3.42), we get an approximation ratio of

$$O\left(\frac{(\log n)(n/\varepsilon)^{1/2}}{(\log n)/\varepsilon}\right) = O((\lambda_1)^{1/2}).$$

The Expected Running Time  We turn to the expected running time and show that the expected time spent in any step is polynomial. Let $T_i$ be the random variable for the time spent in Step $i$ and $F_i$ be its failure probability. For Steps 1–2, we immediately get even polynomial worst-case running time, since their efforts are the same as the ones of Steps 1–2 in ImprovedIndSet (when run on a graph). For the latter, we showed polynomial worst-case running time in the proof of Theorem 2.

We turn to Steps 3–5. Let $s' := (8 \log n)/\varepsilon$. Step 3’s effort is

$$O\left(\text{poly}(n) \cdot \binom{n}{s'}\right) = O(\text{poly}(n) \cdot n^{s'}) = O\left(\text{poly}(n) \cdot \exp\left(\frac{8(\ln n)^2}{\varepsilon \ln 2}\right)\right),$$

since it tests $\binom{n}{s'}$ sets, each of which in polynomial time. The step is only executed if $F_2$ occurs, i.e., Step 2 does not output big($C$). Then, $\lambda_1 \geq 2^k \cdot (\log n) \cdot (n/\varepsilon)^{1/2}$,
which happens with probability at most \(4 \cdot \exp(-2^9 \cdot n\varepsilon \cdot (\log n)^2)\) by Lemma 5. We conclude that the expected running time of Step 3 is

\[
E[T_3] = O\left(\text{poly}(n) \cdot \exp\left(\frac{8(\ln n)^2}{\varepsilon \ln 2} \cdot \exp(-2^9 \cdot n\varepsilon \cdot (\log n)^2)\right)\right)
\]

or

\[
E[T_3] = O\left(\text{poly}(n) \cdot \exp\left(\frac{8(\ln n)^2}{\varepsilon \ln 2} \cdot \frac{2^9 \cdot n\varepsilon \cdot (\ln n)^2}{(\ln 2)^2}\right)\right) .
\]

The exponent in (3.43) is at most zero if and only if \(\varepsilon \geq ((8 \ln 2)/2^9)^{1/2} \cdot n^{-1/2}\), which holds since \(\varepsilon \geq n^{-1/2}\). Thus, \(E[T_3] = O(\text{poly}(n))\), which is polynomial. Set \(n' := (2 \log n)(n/\varepsilon)^{1/2}\). Then,

\[
\Pr[F_3] = \Pr[\exists S' \subseteq V, |S'| = s' : |\overline{N}(S')| > n'] .
\]

If this event occurs, there are sets \(S', N' \subseteq V, |S'| = s'\) and \(|N'| = n'\), such that none of the \(s'n'\) potential edges between \(S'\) and \(N'\) exists. Each edge is absent with probability at most \(1 - \varepsilon\). Using a union bound over all sets \(S', N'\), we get with \(1 - x \leq e^{-x}\) for \(x \in \mathbb{R}\)

\[
\Pr[F_3] \leq \left(\frac{n}{s'}\right) \cdot \left(\frac{n}{n'}\right) \cdot (1 - \varepsilon)^n, \]

\[
\leq n' \cdot n'' \cdot \exp(-s'n') \]

\[
= \exp\left(\frac{8 \cdot (\ln n)^2}{\varepsilon \ln 2} + \frac{2 \cdot (\ln n)^2(n/\varepsilon)^{1/2}}{\ln 2} - \frac{16 \cdot (\ln n)^2(n/\varepsilon)^{1/2}}{(\ln 2)^2}\right) \]

\[
\leq \exp\left(\frac{8 \cdot (\ln n)^2}{\varepsilon \ln 2} - \frac{16 \cdot (\ln n)^2(n/\varepsilon)^{1/2}}{(\ln 2)^2}\right) .
\]

using

\[
\frac{8 \cdot (\ln n)^2}{\varepsilon \ln 2} \leq \frac{8 \cdot (\ln n)^2(n/\varepsilon)^{1/2}}{\ln 2},
\]

due to \(\varepsilon \geq n^{-1/2} \geq 1/n\) to derive (3.44). Since the number of tested sets \(S''\) in Step 4 is

\[
\left(\frac{n}{(8 \log n)(n/\varepsilon)^{1/2}}\right) \leq \exp\left(\frac{8 \ln 2}{(\ln n)^2(n/\varepsilon)^{1/2}}\right),
\]

we can infer with (3.45) that

\[
E[T_4] = O\left(\text{poly}(n) \cdot \exp\left(\frac{8 \ln 2}{(\ln n)^2(n/\varepsilon)^{1/2}}\right) \cdot \exp\left(-\frac{8 \ln 2}{(\ln n)^2(n/\varepsilon)^{1/2}}\right)\right) .
\]

or

\[
E[T_4] = O(\text{poly}(n)) .
\]
In a fixed tested set $S''$, there are
\[
\left(\frac{(8 \log n)(n/\epsilon)^{1/2}}{2}\right) \geq \frac{8^2(\log n)^2 n/\epsilon}{4} = \frac{16}{(\ln 2)^2} \cdot (\ln n)^2 n/\epsilon
\]
potential edges. Thus, $S''$ is independent with probability at most
\[
(1 - \epsilon)^{\frac{16}{(\ln 2)^2} \cdot (\ln n)^2 n/\epsilon} \leq \exp\left(-\epsilon \cdot \frac{16}{(\ln 2)^2} \cdot (\ln n)^2 n/\epsilon\right)
\]
\[
= \exp\left(-\frac{16}{(\ln 2)^2} \cdot (\ln n)^2 n\right).
\]

Recall that the number of tested sets in Step 4 is at most
\[
\exp\left(\frac{8}{\ln 2} \cdot (\ln n)^2 (n/\epsilon)^{1/2}\right) = \exp\left(O((\ln n)^2 n^{3/4})\right) = \exp\left(o((\ln n)^2 n)\right)
\]
since $\epsilon \geq n^{-1/2}$. With a union bound, we get
\[
\Pr[F_4] \leq \exp\left(o((\ln n)^2 n)\right) \cdot \exp\left(-\frac{16}{(\ln 2)^2} \cdot (\ln n)^2 n\right)
\]
\[
= \exp(-\Omega((\log n)^2 n)) .
\] (3.46)

Step 5 is only executed if $F_4$ occurs or if Step 1 fails, i.e., $|\text{big}(C)| < \text{gis}(G, \epsilon, 1/2)$. Lemma 4 shows that this happens with probability at most $e^{-n \ln n}$. Together with (3.46), we can conclude that Step 5 is executed with probability at most $\exp(-\Omega((\log n)^2 n)) + \exp(-n \ln n) = O(e^{-n \ln n})$. Since Step 5 tests $2^n$ sets, its effort is $O(\text{poly}(n) \cdot 2^n)$, and we get
\[
\mathbb{E}[T_5] = O(\text{poly}(n) \cdot 2^n \cdot e^{-n \ln n}) = O(\text{poly}(n))
\]
which completes our proof. 

3.4 The Expected Behavior of Greedy Independent Set

In this section, we prove Corollary 2, analyzing the expected behavior of GreedyColor for our semi-random graphs. The corollary states the following.
Fix a constant \( \delta \in (0, 1) \). Fix a graph \( G = (V, E) \) and a flip probability \( \varepsilon = \varepsilon(n) \) with \( n^{-(1-\delta)} \leq \varepsilon \leq 1/2 \). Let \( \mathcal{G} \) be drawn from \( \mathcal{G}(G, \varepsilon) \), and let \( I \) be the largest color class computed by \text{GreedyColor}(\mathcal{G}) \). Let \( \varepsilon \) be high if \( \frac{\ln(1/\varepsilon)}{\varepsilon} \leq \frac{n^2}{|E|} \) and small otherwise. Then, the expected approximation ratio \( \mathbb{E}[\text{ar}(I)] \) is

\[
\begin{align*}
\mathbb{E}[\text{ar}(I)] &= \begin{cases} 
O(1) & \text{if } \varepsilon \text{ is high and} \\
O\left(\frac{|E|}{n^2} \cdot \frac{\log(1/\varepsilon)}{\varepsilon}\right) & \text{if } \varepsilon \text{ is small.}
\end{cases}
\end{align*}
\]

Furthermore, with probability at least \( 1 - 2 \cdot e^{-n \ln n} \), the approximation ratio fulfills

\[
\text{ar}(I) = \begin{cases} 
O\left(\left(\frac{n \log n}{\varepsilon}\right)^{1/2}\right) & \text{if } \varepsilon \text{ is high and} \\
O\left(\frac{n}{n^2 \log n} \cdot \frac{\log(1/\varepsilon)}{\varepsilon^{1/2}}\right) & \text{if } \varepsilon \text{ is small.}
\end{cases}
\]

\textbf{Proof of Corollary 2.} We start with the statement about the expected approximation ratio. Let \( \mathcal{G} = (V, \mathcal{E}) \) be drawn from \( \mathcal{G}(G, \varepsilon) \). In Lemma 11, we analyzed the distribution of the independence number in a random \( H_d(n, p) \) graph, i.e., also for \( G(n, p) \). Clearly, the probability that \( \alpha(G) \) is greater than some threshold \( t \) is at most the probability that this happens assuming that the adversary graph \( G \) contains no edges. (Removing edges from \( G \) increases the probability of large independent sets, since non-edges of \( G \) are absent in \( \mathcal{G} \) with higher probability than edges of \( G \).) If \( G = (V, \emptyset) \), our random graph \( \mathcal{G} \) is simply \( G(n, \varepsilon) \) distributed. Therefore, Lemma 11 with \( d = 2 \) and \( p = \varepsilon \) immediately yields that with probability at least \( 1 - 1/n \),

\[
\alpha(\mathcal{G}) = O\left(\frac{\log n}{\varepsilon}\right),
\]

and with probability at least \( 1 - e^{-n \ln n} \),

\[
\alpha(\mathcal{G}) = O\left(\left(\frac{n \log n}{\varepsilon}\right)^{1/2}\right).
\]

(By our choice of \( \varepsilon \geq n^{-(1-\delta)} \), which is the same as in Lemma 4, Lemma 11 is applicable.) Lemma 4 states that with probability at least \( 1 - e^{-n \ln n} \), \text{GreedyColor}(\mathcal{G}) finds an independent set \( I \) of size

\[
|I| = \Omega\left(\min\left\{\frac{\log n}{\varepsilon}, \frac{n^2 \log n}{|E| \log(1/\varepsilon)}\right\}\right).
\]

Using (3.47) and (3.49) instead of (2.28) and (2.15), the same arguments as in the proof of Corollary 1 yield that the expected approximation ratio of the greedy
### 3.5. Small Flip Probabilities

For small values of \( \varepsilon \) (i.e., \( \frac{\ln(1/\varepsilon)}{\varepsilon} > \frac{n^2}{|E|} \)), (3.50) yields \( E[ar(I)] = O\left(\frac{|E| \cdot \log(1/\varepsilon)}{\varepsilon} \right) \) as claimed in the corollary. For high values of \( \varepsilon \), (3.50) yields \( E[ar(I)] = O(1) \) as claimed.

We turn to the tail bound on the approximation ratio of the greedy independent set. With (3.48) and (3.49), we conclude that with probability at least \( 1 - 2 \cdot e^{-n \ln n} \),

\[
E[\text{ar}(I)] = \frac{O\left(\frac{n \log n}{\varepsilon}\right)}{\Omega\left(\min\left\{\frac{\log n}{\varepsilon}, \frac{n^2 \log n}{|E| \log(1/\varepsilon)}\right\}\right)} \approx O\left(\frac{E}{\varepsilon}\right) = O(1). \tag{3.51}
\]

Depending on whether \( \varepsilon \) is high or small in the above sense, the denominator in (3.51) is \( \Omega\left(\frac{\log n}{\varepsilon}\right) \) or \( \Omega\left(\frac{n^2 \log n}{|E| \log(1/\varepsilon)}\right) \), respectively.

#### Fix a graph \( G = (V, E) \) and an arbitrary flip probability \( \varepsilon = \varepsilon(n), 0 < \varepsilon \leq 1/2 \). Let \( G \) be drawn from \( \mathcal{G}(G, \varepsilon) \). Let

\[
p^* := \frac{|E| \cdot (1 - \varepsilon) + \left(\binom{n}{2} - |E|\right) \cdot \varepsilon}{\binom{n}{2}}. \tag{3.52}
\]

Then, \( \text{SmallEdgePrIndSet}(G, p^*) \) has polynomial expected running time. If \( p^* \geq 1/n \), it has approximation guarantee \( O(np^*) \). If \( p^* < 1/n \), it has approximation guarantee \( O(1) \).
Proof of Theorem 8. Recall Theorem 6, which we proved in Section 2.5. The theorem analyzes the performance of SmallEdgePrIndSet\((H, p)\) for a random hypergraph \(H \sim H_d(n, p)\) and an arbitrary edge probability \(p\). In the graph case \(d = 2\), it states that the approximation guarantee is \(O(np)\) for \(p \geq 1/n\) and \(O(1)\) for \(p < 1/n\).

Theorem 8 analyzes the behavior of SmallEdgePrIndSet\((G, p^*)\) for a random graph \(G = (V, E)\) from \(G(n, p)\) and the value \(p^*\) from (3.52). (It is easy to see that always \(0 < p^* < 1\).) Since the approximation guarantee in Theorem 6 depends only on \(n\) and the value of the parameter \(p\) passed to SmallEdgePrIndSet, it immediately follows by our above discussion that for \(p^* \geq 1/n\), we have approximation guarantee \(O(np^*)\), and for \(p^* < 1/n\), we have approximation guarantee \(O(1)\), as claimed in our theorem. This finishes our analysis of the approximation guarantee of SmallEdgePrIndSet\((G, p^*)\).

We turn to the expected running time, and show that the part of Theorem 6’s proof which analyzes the expected running time of SmallEdgePrIndSet\((H, p)\) immediately yields a proof for the expected running time of our algorithm. To this end, we first consider \(E[|E|]\), the expected number of edges in our semi-random graph \(G = (V, E)\). Since edges in the adversarial graph \(G = (V, E)\) finally appear with probability \(1 - \epsilon\), and non-edges appear with probability \(\epsilon\), we have

\[
E[|E|] = |E| \cdot (1 - \epsilon) + \left(\binom{n}{2} - |E|\right) \cdot \epsilon.
\]

Consequently, using \(p^* = \frac{|E| \cdot (1 - \epsilon) + \left(\binom{n}{2} - |E|\right) \cdot \epsilon}{\binom{n}{2}}\), we can conclude that

\[
E[|E|] = \binom{n}{2} \cdot p^*.
\]

Now observe two facts with respect to the expected running time part of Theorem 6’s proof (for \(d = 2\), i.e., in the \(G(n, p)\) model).

1. The proof uses that the expected number of edges in a random graph from \(G(n, p)\) is \(B\left(\binom{n}{2}, p\right)\) binomially distributed to be able to apply the Chernov bound (2.33). But our Chernov bound works not only for binomially distributed random variables, but also for sums of independent random variables \(X_i \in \{0, 1\}\) where the individual \(X_i\) do not share the same probability to take on the value 1. Therefore, the number \(|E|\) of edges in our random graph \(G = (V, E)\) is suitable to apply (2.33) to it.

2. The proof of Theorem 6 uses that SmallEdgePrIndSet correctly computes the expected number of edges \(\mu = \binom{n}{2} \cdot p\) in \(G(n, p)\). Since we pass \(p^*\) to SmallEdgePrIndSet\((G, p^*)\), it computes \(\mu = \binom{n}{2} \cdot p^* = E[|E|]\) due to (3.54).
Using these two observations, it is easy to verify that the analysis of the expected running time in the proof of Theorem 6 immediately turns into an analysis of the expected running time of $\text{SmallEdgePrIndSet}(G, p^*)$. This yields that the expected running time of $\text{SmallEdgePrIndSet}(G, p^*)$ is polynomial.

\[\Box\]

### 3.6 Conclusions and Open Problems

We have analyzed the approximability of INDEPENDENT SET in our semi-random graph model $\mathcal{G}(G, \varepsilon)$. We modified the algorithm by Krivelevich and Vu [KV02] such that we can handle random graphs from $\mathcal{G}(G, \varepsilon)$ instead of $G(n, p)$.

We presented an approximation algorithm for INDEPENDENT SET that guarantees an approximation ratio of $O((n\varepsilon)^{1/2})$ in expected polynomial time. Furthermore, we proved that the worst-case efficient greedy algorithm has constant expected approximation ratio and has approximation ratio $O\left(\left(\frac{n\varepsilon}{\log n}\right)^{1/2}\right)$ with probability at least $1 - 2 \cdot e^{-n\ln n}$. (All these statements hold for $\varepsilon$ high enough.) As above, we have in some sense a trade-off between the guarantees on running time and solution quality.

With respect to the largest eigenvalue $\lambda_1(A)$ of our auxiliary matrix $A(G)$ for a random $\mathcal{G}(G, \varepsilon)$ graph $G$, we proved that its expected value fulfills $E[\lambda_1(A)] \leq 2^7 \cdot (\log n) \cdot (n/\varepsilon)^{1/2}$. Also, we obtained an upper bound on the probability that $\lambda_1(A)$ is at least two times this value.

**Open Problems** Our approximation algorithm $\text{AdversaryIndSet}$ works for flip probabilities $\varepsilon \geq n^{-1/2}$. As for our algorithms for $H_2(n, p)$, it would be interesting to achieve a smaller lower bound for $\varepsilon$. Also, it would be interesting to eliminate the need of knowing the adversary graph $G$ which was used to create the given semi-random graph $G$. Again, an algorithm automatically adapting to all possible adversary choices would be desirable.
Chapter 4

Shortest Common Superstring

Contents

4.1 Results in the Bernoulli and Perturbation Model ............ 103
   4.1.1 Upper Bounding the Optimal Compression .............. 104
   4.1.2 Computing a Shortest Superstring .................... 111
   4.1.3 The Probabilistic PTAS ............................... 114
4.2 Generalization to the Mixing Model ......................... 117
4.3 Conclusions and Open Problems ............................. 123

In this chapter, we prove our results for Shortest Common Superstring as stated in Section 1.3.2. As mentioned, our probabilistic approximation scheme ApproxSCS (Algorithm 14, cf. also Theorems 9 and 10) uses a subroutine called BoundCompression (Algorithm 11), which tries to certify that a certain upper bound on the optimal compression of the random input $S$ holds. This is done in order to prove that the greedy solution has a sufficiently good approximation ratio.

BoundCompression obtains an upper bound on $\text{opt}_{c}(S)$ by computing a maximum spanning tree in an auxiliary graph, which is defined in Section 4.1.1, and compares the obtained value with a certain threshold. By proving an upper bound on the failure probability of BoundCompression, i.e., by proving a tail bound on the value of the maximum spanning tree (cf. Lemmas 14 and 17 below), we can upper bound the failure probability of the greedy step in ApproxSCS and also get Lemmas 6 and 7 as corollaries, which give our tail bound on the optimal compression. We also prove Theorem 11, which shows that GreedySCS achieves with probability exponentially close to 1 almost optimal approximation ratio in the length measure.

For the sake of a clear presentation, we prove our results in the Bernoulli and perturbation model in Section 4.1 and generalize them to the mixing model in Section 4.2. Finally, in Section 4.3, we draw conclusions and discuss open problems.

4.1 Results in the Bernoulli and Perturbation Model

In Section 4.1.1, we give algorithm BoundCompression and analyze its failure probability in the Bernoulli and perturbation model. We also prove Lemma 6.
In Section 4.1.2, we present algorithm ShortestSupStr, which is used in ApproxSCS to compute a shortest superstring. Finally, we present our approximation scheme in Section 4.1.3.

### 4.1.1 Upper Bounding the Optimal Compression

For a set $S$ of strings and two strings $s, t \in S$ such that $s$ is a substring of $t$, it is easy to see that removing $s$ from the set $S$ does not affect the set of possible superstrings for $S$. Consequently, also the length of a shortest superstring for $S$ does not change. The following algorithm isolates, given an input $S$ for ShortestCommonSuperstring, a set $S_{rel}$ of “relevant” (regarding opt$(S)$) strings.

**Algorithm 10 Partition($S$) (Isolate relevant strings from $S$.)**

1: $S_{rel} := S$.
2: while there are two strings $s, t \in S_{rel}$ with $s$ a substring of $t$ do
   
   $S_{rel} := S_{rel} \setminus \{s\}$.

end while
3: Set $S_{sub} := S \setminus S_{rel}$ and output $(S_{rel}, S_{sub})$.

It is easily seen that finally every string in $S_{sub}$ is a substring of one in $S_{rel}$, and $S_{rel}$ is substring-free, which means that for no two strings $s, t \in S_{rel}$, $s$ is a substring of $t$. Furthermore, as argued above, $S$ and $S_{rel}$ have the same possible superstrings, and opt$_i(S) = opt_i(S_{rel})$. Obviously, Partition can be implemented with running time $O(poly(||S||))$.

For a set $S$ of strings, we now define the auxiliary graph which we use to upper bound the optimal compression for $S$.

**Definition 6.** For a set $S$ of strings, the string graph $G_s(S) = (S, E)$ is a complete (without self-loops), undirected, weighted graph with vertex set $S$. For an edge $e = \{s^i, s^j\} \in E$, the weight is

$$w(e) = \begin{cases} 
\min\{|s^i|, |s^j|\} & \text{if $s^i$ is a substring of $s^j$ or vice versa} \\
\max\{|ov(s^i, s^j)|, |ov(s^j, s^i)|\} & \text{otherwise} 
\end{cases}.$$ 

Furthermore, let mxt($S$) be the weight of a maximum spanning tree in $G_s(S)$.

**Lemma 13.** For every input $S$ for Shortest Common Superstring, opt$_c(S) \leq mxt(S)$.

Before we prove Lemma 13, given two strings $s$ and $t$ with $s$ being a substring of $t$, let the leftmost occurrence of $s$ in $t$ be the smallest starting index $1 \leq j \leq |t| - |s| + 1$ such that $s_i = t_{j-i}$, $i = 1, \ldots, |s|$. In other words, it is the leftmost position at which $s$ appears in $t$. 
In the proof, we use a known property of substring-free inputs for Shortest Common Superstring. Fix an input $S$ and let $t^*$ be a shortest superstring for $S$. Let $s^1, \ldots, s^n$ be $S$’s strings in order of increasing leftmost occurrence in $t^*$. It is well known (see e.g. Vazirani [Vaz01]) that if $S$ is substring-free, the optimal compression for $S$ fulfills

$$\text{opt}_c(S) = \sum_{i=1}^{n-1} |\text{ov}(s^i, s^{i+1})| .$$

(4.1)

Proof of Lemma 13. We construct a tree $T$ in the string graph $G_s(S)$ with weight $w(T) \geq \text{opt}_c(S)$. Set $(S_{\text{rel}}, S_{\text{sub}}) := \text{Partition}(S)$. Let $t^*$ be a shortest superstring for $S_{\text{rel}}$, and let $s^1_{\text{rel}}, \ldots, s^n_{\text{rel}}$ be the strings of $S_{\text{rel}}$ in order of increasing leftmost occurrence in $t^*$. Since $S_{\text{rel}}$ is substring-free, (4.1) yields

$$\text{opt}_c(S_{\text{rel}}) = \sum_{i=1}^{n' - 1} |\text{ov}(s^i_{\text{rel}}, s^{i+1}_{\text{rel}})| \leq \sum_{i=1}^{n' - 1} w(\{s^i_{\text{rel}}, s^{i+1}_{\text{rel}}\})$$

(4.2)

for the weights $w(\{s^i_{\text{rel}}, s^{i+1}_{\text{rel}}\})$ of the edges $\{s^i_{\text{rel}}, s^{i+1}_{\text{rel}}\} \in E$ in the string graph $G_s(S) = (S, E)$. Here, we used that by definition, $w(\{s^i_{\text{rel}}, s^{i+1}_{\text{rel}}\}) = \max\{|\text{ov}(s^i_{\text{rel}}, s^{i+1}_{\text{rel}})|, |\text{ov}(s^{i+1}_{\text{rel}}, s^i_{\text{rel}})|\}$. By choosing

$$P := \{\{s^i_{\text{rel}}, s^{i+1}_{\text{rel}}\}, \ i = 1, \ldots, n' - 1\},$$

we get a path $P$ in $G_s(S)$ which connects all strings in $S_{\text{rel}}$. As an example, consider Figure 4.1. The path $P$ through the four strings in $S_{\text{rel}}$ is shown by solid edges and has weight $6 + 7 + 7 = 20$.

Due to (4.2), the weight $w(P)$ of the path fulfills

$$w(P) \geq \text{opt}_c(S_{\text{rel}}) .$$

(4.3)

Figure 4.1: A String Graph With a Spanning Tree.
We construct our spanning tree $T$ by setting $T := P \cup A$ for a set $A$ of additional edges that connect the strings in $S_{\text{sub}}$ to $P$. The set $A$ looks as follows.

Let $s_1', \ldots, s_{n''}'$ be the strings in $S_{\text{sub}}$. We consider the computation of $(S_{\text{rel}}, S_{\text{sub}})$ by Partition$(S)$. For every string $s_i' \in S_{\text{sub}}$, there is a string $s_{\pi i}' \in S$ that caused Partition to remove $s_i'$ from $S_{\text{rel}}$ since it found $s_i'$ to be a substring of $s_{\pi i}'$. Let the set $A$ of additional edges be

$$A := \{ \{ s_i', s_{\pi i}' \}, \ i = 1, \ldots, n'' \} .$$

In Figure 4.1, the edges in $A$ are shown dashed. Since Partition never considers a string again once it is removed from $S_{\text{rel}}$, it is easy to see that adding the set $A$ of edges to $P$ does not create cycles. Therefore, in the end, $T = P \cup A$ contains $|S_{\text{rel}}| - 1 + |S_{\text{sub}}| = n - 1$ edges and no cycles and hence is a tree. Furthermore, notice that by definition of the edge weights in $G_s(S)$,

$$w(A) = \sum_{s_i' \in S_{\text{sub}}} w(\{ s_i', s_{\pi i}' \}) = \sum_{s_i' \in S_{\text{sub}}} |s_i'| = \| S_{\text{sub}} \| .$$

(4.4)

Since $\text{opt}_L(S_{\text{rel}}) = \text{opt}_L(S)$, we have $\text{opt}_c(S_{\text{rel}}) = \| S_{\text{rel}} \| - \text{opt}_L(S)$. Using this, together with (4.3) and (4.4), we infer

$$w(T) = w(P) + w(A) \geq \text{opt}_c(S_{\text{rel}}) + \| S_{\text{sub}} \| = \| S_{\text{rel}} \| - \text{opt}_L(S) + \| S_{\text{sub}} \| = \text{opt}_c(S) .$$

Together with $\text{mxt}(S) \geq w(T)$, the claim of the lemma follows.

We are ready to present algorithm BoundCompression$(S, c)$, which for a random input $S$ for Shortest Common Superstring and a rational number $c > 0$ either certifies by outputting “Success” that $\text{opt}_c(S) \leq cn \ln n$, or outputs “Fail”. The latter happens only with exponentially small probability in our models.

Algorithm 11 BoundCompression$(S, c)$ (Try to certify that $\text{opt}_c(S) \leq cn \ln n$.)

1: Compute $\text{mxt}(S)$.
2: if $\text{mxt}(S) \leq cn \ln n$ then
   Output “Success”.
else
   Output “Fail”.
end if

For the next lemma, recall that for a set $S$ of strings, $\Delta(S) = \max_{1 \leq i, j \leq n} |s_i' - s_j'| + 1$ measures the maximum length difference of two strings.
4.1. Results in the Bernoulli and Perturbation Model

**Lemma 14.** Fix a Bernoulli or perturbation model and \( \varepsilon > 0 \). For a random input \( S \) for **Shortest Common Superstring** and

\[
c \geq \frac{1 + (\ln \Delta(S)) / \ln n}{|\ln p_{\text{max}}|} + \varepsilon, \quad c = O(1),
\]

**BoundCompression** \((S, c)\) runs in time \( O(\text{poly}(||S||)) \). It outputs “Fail” with probability \( \exp(-\Omega(n \log n)) \). If it outputs “Success”, \( \text{opt}_c(S) \leq cn \ln n \).

**Proof.** The algorithm clearly has running time \( O(\text{poly}(||S||)) \). Lemma 13 yields that if “Success” is output, \( \text{opt}_c(S) \leq \text{mxt}(S) \leq cn \ln n \). It remains to analyze the failure probability, which is \( \Pr[\text{mxt}(S) > cn \ln n] \).

If \( \text{mxt}(S) > cn \ln n \), we have

\[
\text{mxt}(S) = (cn \ln n) + \delta =: f(\delta)
\]

for some \( \delta \in \{1, \ldots, ||S|| - cn \ln n\}. \) (It is easy to see that \( \delta \) cannot be larger, since always \( \text{mxt}(S) \leq ||S|| \).) In the following, we upper bound \( \Pr[\text{mxt}(S) = f(\delta)] \) for fixed \( \delta \).

If \( \text{mxt}(S) = f(\delta) \), there is a spanning tree \( T \subseteq E \) in the string graph \( G_s \) with edge weights \( w(e) \geq 0 \) for \( e \in T \) such that \( \sum_{e \in T} w(e) = f(\delta) \). By definition of \( G_s \), this implies the following: For every edge \( e = \{s^a, s^b\} \in T \), there is a **direction** \( (a, b) \in \{(i, j), (j, i)\} \) such that \( |\text{ov}(s^a, s^b)| = w(e) \) (e has **type** overlap) or \( s^a \) is a **substring** of \( s^b \) and \( |s^a| = w(e) \) (e has **type** substring). Furthermore, for a “substring” edge, there is a leftmost occurrence \( l_{a,b} \in \{1, \ldots, |s^b| - |s^a| + 1\} \) where \( s^a \) appears in \( s^b \). A spanning tree \( T \) together with its edge weights, directions, types, and leftmost occurrences is called a **spanning tree configuration**. An example of a string graph with a spanning tree configuration is shown in Fig. 4.2 (only the tree’s edges with their weights are shown). An arrow from \( s^a \) to \( s^b \) at an edge \( e = \{s^a, s^b\} \) means that \( e \)'s direction is \((i, j)\), and the type is indicated by ‘o’ (overlap) or ‘s’ (substring with leftmost occurrence \( l \)).

![Figure 4.2: A Spanning Tree Configuration C and the Resulting Equality Graph G_e(C).](image)

Below, we upper bound \( \Pr[\text{mxt}(S) = f(\delta)] \) using a union bound over all possible spanning tree configurations, i.e., we have to bound the number of configurations and the probability that a fixed configuration appears.
The number of possible configurations

We count the number of possible spanning tree configurations as follows:

- There are $n^{n-2}$ spanning trees in a graph on $n$ labeled vertices. Hence, we have $n^{n-2} \leq \exp(n \ln n)$ choices for the spanning tree.

- For natural numbers $x \geq y$, the number of ordered partitions of $x$ into $y$ nonnegative summands is $\binom{x+y-1}{y-1}$, and $\binom{y}{y} = \left(\frac{e}{y}\right)^y$. Hence, the number of ways to partition $f(\delta)$ into $n-1$ nonnegative edge weights is

  $$\binom{f(\delta) + n - 2}{n - 1} \leq \left(\frac{2f(\delta)}{n}\right)^n \leq \exp\left(\frac{f(\delta)}{n}\right) + O(n)$$

- The edge directions and types can be chosen in $4^{n-1} = \exp(O(n))$ ways.

- For the leftmost occurrence of a string $s'$ in $s'$, we have $|s'| - |s'| + 1 \leq \Delta(S)$ choices. For the leftmost occurrences of the at most $n-1$ “substring” edges, we have at most $(\Delta(S))^{n-1} \leq \exp(n \ln \Delta(S))$ choices.

Now, let $\hat{C}$ be the set of all spanning tree configurations. For brevity, let $k := (\ln \Delta(S))/\ln n$. Then, $n \ln \Delta(S) = kn \ln n$. Using the above number of ways to create a spanning tree configuration, we conclude

$$|\hat{C}| \leq \exp\left((n \ln n) + n \ln \left(\frac{f(\delta)}{n}\right) + (n \ln \Delta(S)) + O(n)\right)$$

$$= \exp\left((1 + k)(n \ln n) + n \ln \left(\frac{f(\delta)}{n}\right) + O(n)\right) \quad (4.5)$$

The probability of a fixed configuration

Now, we upper bound the probability that a fixed configuration $C \in \hat{C}$ appears, i.e., the string graph of $S$ contains the maximum spanning tree $T$ from $C$. We model the properties of $S$’s strings implied by $C$’s appearance using the equality graph $G_e(C) = (V, E)$, which looks as follows.

The vertex set is $V = \{s'_j | i = 1, \ldots, n, \ j = 1, \ldots, |s'_i|\}$. In other words, for every string $s'_i \in S$, the equality graph contains the vertices $s'_j, j = 1, \ldots, |s'_i|$, regardless of the actual assignment of elements from the alphabet $\Sigma$ to the letters $s'_j$. The vertices are seen as placeholders for the values of the $s'_i$. In consequence, $|V| = ||S||$.

To construct $G_e(C)$’s edge set $E$, we consider all edges $e = \{s', s''\} \in T$. Let $(a, b)$ be the direction of $e$. If $e$’s type is substring with leftmost occurrence $l_{a,b}$, we know that if $C$ appears, then $s^a$ appears as a substring at position $l_{a,b}$ in $s^b$, i.e., it holds that

$$s^a_1 \ldots s^a_{|s^a|} = s^b_{l_{a,b}} \ldots s^b_{l_{a,b} + |s^a| - 1}$$
4.1. Results in the Bernoulli and Perturbation Model

We add the $|s^a|$ edges

$$
\{s^a_1, s^b_1\}, \ldots, \{s^a_{|\varepsilon|}, s^b_{|\varepsilon|+|\varepsilon|-1}\}
$$

to $E$ to reflect this. Then, if $C$ appears, any two letters connected by an edge in $G_e$ have been assigned equal elements from $\Sigma$.

Now, assume that $e$ has type overlap. If $C$ appears, the suffix of $s^a$ of length $w(e)$ matches the prefix of $s^b$ of that length, or equivalently

$$
s^a_{|\varepsilon|-w(e)+1} \ldots s^a_{|\varepsilon|} = s^b_1 \ldots s^b_{w(e)} .
$$

Thus, we add the $w(e)$ edges

$$
\{s^a_{|\varepsilon|-w(e)+1}, s^b_1\}, \ldots, \{s^a_{|\varepsilon|}, s^b_{w(e)}\} .
$$
to $E$.

On the right hand side of Fig. 4.2, the equality graph for the configuration on the left hand side is shown. Let $C = \{C_1, \ldots, C_{|C|}\}$ contain the connected components of size at least two in $G_e$. For example, in Fig. 4.2, $|C| = 8$, and the component $C_1$ contains two letters. Let $L = \bigcup_{i=1}^{|C|} C_i$ be the set of all letters $s^a_j$ contained in the components in $C$. Since $G_e$’s edges are inserted according to the tree $T$, which contains no cycles, $G_e$ contains no cycles. Hence, all components $C_i$ are trees. The number of vertices in a tree is its edge number plus one. Since $G_e$ contains $\sum_{\varepsilon \in T} w(e) = f(\delta)$ edges, we get

$$
|L| = \sum_{i=1}^{|C|} |C_i| = f(\delta) + |C| . \quad (4.6)
$$

If $C$ appears, then for every class $C_i$ there is a $c_i \in \Sigma$ such that $c_i$ is assigned to all letters in $C_i$. In both models, the probability that this happens for a single component $C_i$ is at most $p_{\text{max}}^{(C_i)-1}$. Let $l_1, \ldots, l_{|C_i|}$ be the letters in $C_i$. For a fixed $c_i \in \Sigma$,

$$
\Pr\left[l_j = c_i, \ j = 1, \ldots, |C_i|\right] = \prod_{j=1}^{|C_i|} \Pr[l_j = c_i] \leq \Pr[l_1 = c_i] \cdot p_{\text{max}}^{(C_i)-1} .
$$

since $\Pr[l_j = c_i] \leq p_{\text{max}}$ for all $l_j$. Therefore, the probability that for some $c_i \in \Sigma$, $l_j = c_i$ for all $l_j$ is

$$
\sum_{c_i \in \Sigma} \Pr\left[l_j = c_i, \ j = 1, \ldots, |C_i|\right] \leq \sum_{c_i \in \Sigma} \Pr[l_1 = c_i] \cdot p_{\text{max}}^{(C_i)-1} = p_{\text{max}}^{(C_i)-1} .
$$
With \(|L| - |C| = f(\delta)|\) due to (4.6), we get
\begin{align*}
\Pr[C \text{ appears}] & \leq \prod_{i=1}^{\lfloor C \rfloor} p_{\text{max}}^{C_i - 1} \\
& = \exp \left( \sum_{i=1}^{\lfloor C \rfloor} (|C_i| - 1) \ln(p_{\text{max}}) \right) \\
& = \exp (\ln(p_{\text{max}})(|L| - |C|)) \\
& = \exp(\ln(p_{\text{max}})f(\delta)) . \quad (4.7)
\end{align*}

As discussed above, \(\Pr[\text{mxt}(S) = f(\delta)] \leq \Pr[\exists C \in \hat{C}: C \text{ appears}]\). With a union bound over all spanning tree configurations, whose number \(|\hat{C}|\) is bounded by (4.5), and using the upper bound (4.7) on the probability of a fixed configuration, we get
\begin{align*}
\Pr[\text{mxt}(S) = f(\delta)] & \leq \exp ((1 + k)(\ln n) + n \ln(f(\delta)/n) + \ln(p_{\text{max}})f(\delta) + O(n)) \quad (4.8) \\
& = \exp ((1 + k) - \ln p_{\text{max}}|c|)(\ln n) + O(n \ln \ln n + O(n)) . \quad (4.9)
\end{align*}

To get (4.9), consider the term
\begin{align*}
n \ln(f(\delta)/n) + \ln(p_{\text{max}})f(\delta) = n \ln \left( \frac{(cn \ln n) + \delta}{n} \right) + \ln(p_{\text{max}}) \cdot ((cn \ln n) + \delta)
\end{align*}
in (4.8). Standard analysis (i.e., determining the partial derivative with respect to \(\delta\)) shows that its value is at most the one for \(\delta = 0\). Using this and \(c = O(1)\) as assumed in the lemma, we conclude that is has value at most
\begin{align*}
n \ln(cn \ln n) + \ln(p_{\text{max}})cn \ln n = O(n \ln \ln n) - |\ln p_{\text{max}}|cn \ln n .
\end{align*}

Remember that \(k = (\ln \Delta(S))/\ln n\). Therefore, by choice of \(c\) in the lemma, \(c \geq (1 + k)/|\ln p_{\text{max}}| + \epsilon, \) or equivalently
\begin{align*}
|\ln p_{\text{max}}|c \geq 1 + k + \epsilon |\ln p_{\text{max}}| = 1 + k + \epsilon'
\end{align*}
for some fixed \(\epsilon' > 0\). Thus, \((1 + k) - |\ln p_{\text{max}}|c \leq -\epsilon', \) which together with (4.9) yields
\begin{align*}
\Pr[\text{mxt}(S) = f(\delta)] & \leq \exp(-\epsilon'(n \ln n) + O(n \ln \ln n) + O(n)) \\
& = \exp(-\Omega(n \log n)) . \quad (4.10)
\end{align*}

We can finish our proof with a union bound over all possible values of \(\delta \in \{1, \ldots, ||S|| - cn \ln n\}. \) Throughout Chapter 4, we assume that \(||S|| \leq 2^n\). Otherwise, we can solve SHORTEST COMMON SUPERSTRING optimally in time \(O(\text{poly}(||S||))\) with algorithm ShortestSupStr. Now since \(1 \leq \delta \leq ||S|| \leq 2^n\), we get
4.1. Results in the Bernoulli and Perturbation Model

\[ \Pr[\text{mxt}(S) > cn \ln n] \leq \sum_{\delta} \Pr[\text{mxt}(S) = f(\delta)] \]
\[ \leq 2^n \cdot \exp(-\Omega(n \log n)) \quad \text{(due to (4.10))} \]
\[ = \exp(O(n) - \Omega(n \log n)) \]
\[ = \exp(-\Omega(n \log n)) , \]

which finishes our proof. \( \square \)

We can now prove Lemma 6, i.e., the tail bound on the optimal compression in the Bernoulli and perturbation model. For convenience, we restate it here.

Fix \( \varepsilon > 0 \) and a Bernoulli model with \( H/|\ln p_{\text{max}}| < 1 + \varepsilon \). For a random input \( S \) with \( \Delta(S) = \text{polylog}(n) \), it holds that

\[ \Pr[\text{opt}_{c}(S) > (1 + \varepsilon) \cdot n \ln n / H] = \exp(-\Omega(n \log n)) . \]

Proof of Lemma 6. Set \( c := (1 + \varepsilon)/H \). If \( \text{opt}_{c}(S) > (1 + \varepsilon)(n \ln n)/H \), \text{BoundCompression}(S, c) fails, which has probability \( \exp(-\Omega(n \log n)) \) by Lemma 14. We show that the lemma is applicable, i.e., its assumptions are fulfilled.

Since \( \Delta(S) = \text{polylog}(n) \), we have \( \Delta(S) \leq (\ln n)^{k} \) for a fixed \( k \in \mathbb{N} \). Thus,

\[ (\ln \Delta(S))/\ln n = O(\ln \ln n)/\ln n = o(1) . \]

Since \( H/|\ln p_{\text{max}}| < 1 + \varepsilon \), we have

\[ 1/|\ln p_{\text{max}}| = (1 + \varepsilon)/H - \varepsilon' = c - \varepsilon' \]

for some \( \varepsilon' > 0 \). Now, observe that

\[ \frac{1 + (\ln \Delta(S))/\ln n}{|\ln p_{\text{max}}|} + \varepsilon'/2 = (1 + o(1))(c - \varepsilon') + \varepsilon'/2 \leq c . \]

Hence, Lemma 14’s assumptions are fulfilled. \( \square \)

4.1.2 Computing a Shortest Superstring

Let \( G = (V, E) \) be a complete (without self-loops), directed, weighted graph with edge weights \( w(u, v) \) for the edges \( (u, v) \), \( u \neq v \in V \). A Hamiltonian path is a sequence \( v_1, v_2, \ldots, v_{|V|} \) of pairwise different vertices \( v_i \in V \), and the weight of the path is \( \sum_{i=1}^{|V|-1} w(v_i, v_{i+1}) \).
We show that in such a graph, one can compute a maximum weight Hamiltonian path in \(|V|^2 2^{|V|}\) steps with dynamic programming. Algorithm ShortestSupStr below exploits this to compute a shortest superstring for an input \(S\) in time \(O(2^n \cdot \text{poly}(|S|))\).

Fix a graph \(G = (V, E)\) as above. For a subset \(V' \subseteq V\) of its vertices and a vertex \(v \in V'\), let \(H(V', v)\) be the maximum weight of a path visiting every vertex in \(V'\) exactly once, starting at vertex \(v\). Using dynamic programming, we compute the values \(H(V', v)\) for all subsets \(V' \subseteq V\) in some order of increasing cardinality, and all \(v \in V'\). Obviously, a maximum weight Hamiltonian path in \(G\) has weight \(\max_{v \in V'} H(V', v)\). Thus, having computed all the values \(H(V', v)\), we can compute the weight of a maximum Hamiltonian path. Since our final goal is to actually compute such a path and not only its weight, in addition to the values \(H(V', v)\), we store a path \(P(V', v)\) with weight \(H(V', v)\) for all these entries. Then, a single look-up yields our desired solution in the end.

It remains to give the Bellman equation for our problem. For the sets \(V' \subseteq V\) with \(|V'| = 1\), we have only one choice for a vertex \(v \in V'\), and clearly,

\[
H(V', v) = 0 ,
\]

since no edge is passed. For \(|V'| \geq 2\), we get the following recurrence:

\[
H(V', v) = \max_{u \in V' \setminus \{v\}} \{w(v, u) + H(V' \setminus \{v\}, u)\} , \tag{4.11}
\]

since a maximum weight path visiting each vertex of \(V'\) exactly once, starting at vertex \(v\), first reaches some vertex \(u \in V' \setminus \{v\}\) and then continues as a maximum weight path through the latter set. With respect to the number of steps needed to compute the values \(H(V', v)\), notice that we have to compute at most \(2^{|V'|}|V|\) values. For each value, we have to consider at most \(|V|\) already computed values due to (4.11). Hence, we need a total of at most \(|V|^2 2^{|V|}\) steps as claimed above. For the following algorithm MaxHamiltonianPath\((G, w)\), we have proved Lemma 15 below.

**Algorithm 12 MaxHamiltonianPath\((G, w)\)**

1. For all \(v \in V\): Set \(H(\{v\}, v) := 0\) and \(P(\{v\}, v) := v\).
2. for \(k = 1, \ldots, |V|\) do
   for all \(V' \subseteq V\), \(|V'| = k\), and \(v \in V'\) do
     Let \(u'\) be a vertex \(u \in V' \setminus \{v\}\) maximizing \(w(v, u) + H(V' \setminus \{v\}, u)\).
     Set \(H(V', v) := w(v, u') + H(V' \setminus \{v\}, u')\).
     Set \(P(V', v) := v, P(V' \setminus \{v\}, u')\).
   end for
end for
3. Let \(v^*\) be a vertex \(v \in V\) maximizing \(H(V, v)\) and output \(P(V, v^*)\).
4.1. Results in the Bernoulli and Perturbation Model

**Lemma 15.** For every complete, directed, weighted graph $G = (V, E)$ with edge weights $w(u, v)$, $u \neq v \in V$, $\text{MaxHamiltonianPath}(G, w)$ computes a maximum weight Hamiltonian path in time $O(|V|^2 2^{|V|})$.

We modify the definition of the string graph $G_s(S)$ from Sect. 4.1.1 to yield the directed string graph $\overrightarrow{G}_s(S)$.

**Definition 7.** For a set $S$ of strings, the directed string graph $\overrightarrow{G}_s(S) = (S, E)$ is a complete (without self-loops), directed, weighted graph with vertex set $S$. For an edge $e = (s', s') \in E$, the weight is $w(e) = |\text{ov}(s', s')|$.

**Algorithm 13 ShortestSupStr(S)** (Compute shortest superstring for $S$.)

1: Run $\text{Partition}(S)$ and let $(S_{rel}, S_{sub})$ be its output. Let $\{s'_1, \ldots, s'_r\}$ be the strings in $S_{rel}$.
2: Compute a maximum weight Hamiltonian path $s'^{h_1}_r, \ldots, s'^{h_r}_r$ in $\overrightarrow{G}_s(S_{rel})$ using algorithm $\text{MaxHamiltonianPath}$.
3: Output $s'^{h_1}_r \oplus s'^{h_2}_r \oplus \ldots \oplus s'^{h_r}_r$. $\triangleright$ Maximally overlap strings along the path.

As an example run of $\text{ShortestSupStr}$, consider Figure 4.3. It shows six strings, together with the resulting partition into $S_{rel}$ and $S_{sub}$ and the computed maximum weight Hamiltonian path on the four strings in $S_{rel}$. On the right hand side, the output $\text{optimalcompression}$ is shown, together with the alignment of consecutive strings along the path. It is easy to see that with respect to $S_{rel}$, the compression achieved by the output is the weight of the Hamiltonian path.

![Figure 4.3: A Solution Found by Algorithm ShortestSupStr.](image)

We are ready to prove Lemma 8, which states: For every input $S$ for $\text{ShortestCommonSuperstring}$, algorithm $\text{ShortestSupStr}(S)$ computes a shortest superstring in time $O(2^n \cdot \text{poly}(|S|))$. 
Proof of Lemma 8. Since $S$ and $S_{\text{rel}}$ share the same superstrings (see Sect. 4.1.1), to show that the output $s_1 \oplus s_2 \oplus \ldots \oplus s_n = t$ is a shortest superstring for $S$, it suffices to show this for $S_{\text{rel}}$. Clearly, $t$ is a superstring for $S_{\text{rel}}$. We show that it is a shortest one by showing that it has optimal compression for $S_{\text{rel}}$.

Let $t^*$ be a shortest superstring for $S_{\text{rel}}$ and $s_1^{i_1}, \ldots, s_n^{i_n}$ be $S_{\text{rel}}$'s strings in order of increasing leftmost occurrence in $t^*$. Since $s_1^{i_1}, \ldots, s_n^{i_n}$ is a maximum weight Hamiltonian path in $\overrightarrow{G}(S_{\text{rel}})$, with (4.1) we get

$$c(t) = \sum_{j=1}^{n'} |\text{ov}(s_j^{i_j}, s_{j+1}^{i_{j+1}})| \geq \sum_{j=1}^{n'} |\text{ov}(s_j^{i_j}, s_{j+1}^{i_{j+1}})| = \text{opt}_c(S_{\text{rel}}).$$

We turn to the running time. Partition runs in time $O(\text{poly}(||S||))$. With Step 2’s running time of $O(n^2 2^n)$ used for computing a maximum weight Hamiltonian path with dynamic programming, a running time of $O(\text{poly}(||S||) + n^2 2^n) = O(2^n \cdot \text{poly}(||S||))$ follows.

4.1.3 The Probabilistic PTAS

We are now in position to present our probabilistic approximation scheme ApproxSCS.

**Algorithm 14** ApproxSCS($S, c$) (Compute superstring $t$ with $\text{ar}_i(t) \leq c$.)

1: Run BoundCompression($S, c$).
2: if BoundCompression outputs “Success” then run GreedySCS($S$) and output the superstring it computes.
3: Run ShortestSupStr($S$) and output the superstring it computes.

Next, we prove Theorem 9, which we restate here for convenience.

Fix a Bernoulli or perturbation model, $\varepsilon \in (0, 1]$, and $k \in \mathbb{N}$. Set $c := 2(1 + k)/\ln P_{\text{max}}$. Then, for a random input $S$ with $||S|| \geq (2c/\varepsilon)(n \ln n)$ and $\Delta(S) \leq n^k$, ApproxSCS($S, c$) has factor $1 + \varepsilon$ for the length measure and polynomial expected running time.

Proof of Theorem 9. We start with the running time. BoundCompression($S, c$) runs in time $O(\text{poly}(||S||))$ by Lemma 14. Since the same holds for GreedySCS, Steps 1 and 2 have polynomial worst case running time. To prove a polynomial expected running time, it thus suffices to upper bound Step 3’s expected running time, which is the product of the time spent if it is executed and its execution probability. The former is $O(2^n \cdot \text{poly}(||S||))$ by Lemma 8. Now, we upper bound the latter.
4.1. Results in the Bernoulli and Perturbation Model

Since $\Delta(S) \leq n^k$, we have $k \geq (\ln \Delta(S))/\ln n$. Thus, by choice of $c = 2(1 + k)/|\ln p_{\text{max}}|$ in the theorem, we have $c \geq (1 + (\ln \Delta(S))/\ln n)/|\ln p_{\text{max}}| + \epsilon'$ for some $\epsilon' > 0$, and therefore, Lemma 14 is applicable. The lemma yields that BoundCompression($S$, $c$) fails in Step 1 with probability $\exp(-\Omega(n \log n))$. Since Step 3 is only executed if this happens, its execution probability is $\exp(-\Omega(n \log n))$, and thus its expected running time is

$$O(2^n \cdot \text{poly}(\|S\|) \cdot \exp(-\Omega(n \log n))) = O(\text{poly}(\|S\|)),$$

where we used $2^n \cdot \exp(-\Omega(n \log n)) = o(1)$.

We turn to the factor for the length measure. If Step 3 outputs a solution, it has optimal approximation ratio 1. Now assume that Step 2 outputs $t := \text{GreedySCS}(S)$. This happens only if BoundCompression($S$, $c$) succeeds, i.e., in this case, we have

$$\text{opt}_c(S) \leq cn \ln n$$

(4.12)
due to Lemma 14. Since $\|S\| \geq (2c/\epsilon)(n \ln n)$ is assumed in the theorem, we have

$$c \leq \epsilon\|S\|/(2n \ln n).$$

(4.13)

Now, plugging (4.13) into (4.12), we get

$$\text{opt}_c(S) \leq (\epsilon/2)\|S\|.$$  

(4.14)

It follows that

$$\text{opt}_l(S) = \|S\| - \text{opt}_c(S) \geq (1 - \epsilon/2)\|S\|.$$  

(4.15)

Clearly, $|t| \leq \|S\|$ (since GreedySCS in the worst case simply concatenates all strings). Now, (4.15) together with $\epsilon \leq 1$ yields a length approximation ratio of

$$\text{arl}(t) = \frac{|t|}{\text{opt}_l(S)} \leq \frac{\|S\|}{(1 - \epsilon/2)\|S\|} = \frac{1}{1 - \epsilon/2} \leq 1 + \epsilon,$$

which finishes our proof. \hfill \Box

We can now prove Theorem 11, which claims the following.

*Fix a Bernoulli or perturbation model with parameters as in Theorem 9 or fix a mixing model with parameters as in Theorem 10. Then, for a random input $S$, it holds that $\Pr[\text{arl}(\text{GreedySCS}(S)) \leq 1 + \epsilon] \geq 1 - \exp(-\Omega(n \log n))$.  


Proof of Theorem 11. We prove the claim for the Bernoulli and perturbation model. In the proof of Theorem 9, we have shown that under its assumptions, ApproxSCS guarantees approximation ratio \(1 + \varepsilon\). Hence, if Step 2 in the algorithm outputs the greedy solution, it achieves this approximation ratio. We conclude that if GreedySCS fails to produce approximation ratio at most \(1 + \varepsilon\), algorithm ApproxSCS performs Step 3 (computing an optimal solution). In the proof of Theorem 9, we have shown that the latter happens with probability \(\exp(-\Omega(n \log n))\), which directly yields our desired result.

We turn to the mixing model. In Section 4.2 below, we prove Theorem 10, which is a version of Theorem 9 for the mixing model. In the proof, we show that as before, ApproxSCS executes Step 3 with probability \(\exp(-\Omega(n \log n))\). The rest of our proof works as before. \(\square\)

We can also prove Corollary 3, which claims the following.

\*Fix an alphabet \(\Sigma\) and \(k \in \mathbb{N}\). Let \(\tilde{S}\) be the set of all inputs \(S\) for Shortest Common Superstring with \(||S|| \geq \left(4(1 + k)/\ln|\Sigma|\right)(n \ln n)\) and \(\Delta(S) \leq n^k\). Then, the fraction of inputs from \(\tilde{S}\) for which GreedySCS fails to produce length approximation ratio 2 is at most \(\exp(-\Omega(n \log n))\).\*

\*Proof.\* Consider a Bernoulli model with the uniform distribution over \(\Sigma\). Fix a sequence of \(n\) string lengths \(l_1, \ldots, l_n\) obeying the two bounds in the corollary. Let \(\tilde{S}'\) be the set of all inputs \(S' = \{s^1, \ldots, s^n\}\) with \(|s^i| = l_i\) for \(i = 1, \ldots, n\). We show that the statement in the corollary holds for \(\tilde{S}'\). Since it holds for all individual choices of \(l_1, \ldots, l_n\), it immediately follows that it holds for the set \(\tilde{S}\) of all inputs fulfilling the two bounds in any way.

Observe that the assumptions made in our corollary are exactly the ones from Theorem 9 for the uniform distribution over \(\Sigma\), i.e., \(p_{\text{max}} = 1/|\Sigma|\), and \(\varepsilon = 1\). In the proof of the theorem, we showed that in case the greedy solution is output, it has approximation ratio \(1 + \varepsilon = 2\) by choice of \(\varepsilon = 1\). Furthermore, we showed that the probability that the greedy solution is not output is \(\exp(-\Omega(n \log n))\). Hence, the probability that GreedySCS fails to produce length approximation ratio 2 for a set \(S'\) of \(n\) random strings \(s^i\) with lengths \(|s^i| = l_i\) is \(\exp(-\Omega(n \log n))\). By choice of the uniform distribution, every set of \(n\) strings with these lengths appears with the same probability, and hence, the \(\exp(-\Omega(n \log n))\) upper bound on the failure probability directly transfers into our claim about the fraction of inputs from \(\tilde{S}'\). \(\square\)
4.2 Generalization to the Mixing Model

In this section, we generalize our results from Section 4.1 to the mixing model. To be more precise, we prove Lemma 7 (the tail bound on $\text{opt}_c(S)$) and Theorem 10 (the analysis of $\text{ApproxSCS}$) in the mixing model. Both Lemma 7 and Theorem 10 have corresponding versions in the Bernoulli model, namely Lemma 6 and Theorem 9. Our new mixing model versions and the old Bernoulli model versions are almost identical, except one detail: If we replace $|\ln p_{\text{max}}|$ by $c_{\text{mix}}$ in Lemma 6, we get Lemma 7, and if we do the same with Theorem 9, we get Theorem 10.

For the definition of $c_{\text{mix}}$, consider again (1.26) in Section 1.3.2, which claims that in the mixing model, $p_{\text{max}}^l = \exp(-\Omega(l))$, where $p_{\text{max}}^l$ is the maximum probability of a string of length $l$. Using this, we defined $c_{\text{mix}}$ by

$$c_{\text{mix}} = \sup \left\{ c > 0 \mid \exists l_0 \in \mathbb{N}: \forall l \geq l_0: |\ln p_{\text{max}}^l|/l \geq c \right\},$$

and concluded that always $c_{\text{mix}} > 0$. Since we deferred the proof of (1.26) to this section, we prove it now before we can turn to generalizing our results to the mixing model.

**Lemma 16.** Fix a mixing model. Then, $p_{\text{max}}^l = \exp(-\Omega(l))$.

**Proof.** Fix a mixing model. Given $l \in \mathbb{N}$, we fix $x^* \in \Sigma^l$ with $p_{\text{max}}^l = p_{\text{max}}^l$. Furthermore, for the function $\alpha: \mathbb{N} \to [0, 1)$ with $\lim_{g \to \infty} \alpha(g) = 0$ limiting the dependencies in the model (cf. (1.16)), we choose a constant $g_0 \in \mathbb{N}$ with $\alpha(g_0) < (1 - p_{\text{max}})/p_{\text{max}}$. Let $r \in \Sigma^l$ be a random string from our model, and set $m := [(l - 1)/g_0]$. Clearly, $m = \Omega(l)$.

For brevity, we define two events. For $k \in \{0, \ldots, m\}$, we let $E_k$ be the event “$r_{1+kg_0} = x^*_{1+kg_0}$”. Furthermore, we let $\tilde{E}_k := \bigcap_{k=0}^{k-1} E_k$. Then,

$$p_{\text{max}}^l = \Pr[r = x] \leq \Pr \left[ \bigcap_{k=0}^{m} E_k \right].$$

Observe that for every $k$, the letters concerning the events $E_k$ and $\tilde{E}_k$ have a distance of at least $g_0$ in the random string $r$. By choice of $g_0$, $(1 + \alpha(g_0))p_{\text{max}} < 1$. With (1.16) (limiting the dependencies in our model), it follows that

$$\Pr[E_k \mid \tilde{E}_k] = \frac{\Pr[E_k \cap \tilde{E}_k]}{\Pr[\tilde{E}_k]} \leq (1 + \alpha(g_0)) \Pr[E_k] \leq (1 + \alpha(g_0))p_{\text{max}} =: c < 1. \tag{4.16}$$

We can finish our proof by observing that

$$p_{\text{max}}^l \leq \Pr \left[ \bigcap_{k=0}^{m} E_k \right] = \prod_{k=0}^{m} \Pr[E_k \mid \tilde{E}_k] \leq c^{m+1} \leq c^{m \ln c} = e^{-\Omega(l)},$$

where $c := (1 + \alpha(g_0))p_{\text{max}}$. 

which finishes our proof. □

For convenience, we restate Lemma 7 and Theorem 10 before proving them. Lemma 7 states the following:

Fix \( \varepsilon > 0 \) and a mixing model with \( H/c_{\text{mix}} < 1 + \varepsilon \). For a random input \( S \) with \( \Delta(S) = \text{polylog}(n) \), it holds that

\[
\Pr \left[ \text{opt}_c(S) > \frac{(1 + \varepsilon) \cdot n \ln n}{H} \right] = \exp(-\Omega(n \log n)).
\]

Theorem 10 claims the following:

Fix a mixing model, \( \varepsilon \in (0, 1] \), and \( k \in \mathbb{N} \). Set \( c := 2(1+k)/c_{\text{mix}} \). Then, for a random input \( S \) with \( ||S|| \geq (2c/\varepsilon)(n \ln n) \) and \( \Delta(S) \leq n^k \), \( \text{ApproxSCS}(S, c) \) has factor \( 1 + \varepsilon \) in the length measure and polynomial expected running time.

Proof of Lemma 7 and Theorem 10

Our proof of Lemma 7 and Theorem 10 works as follows.

Remember that Lemma 14 gives an upper bound on the failure probability of algorithm \( \text{BoundCompression} \) in the Bernoulli model. Using Lemma 14 as a central tool, we were able to prove Lemma 6 and Theorem 9, giving our tail bound on \( \text{opt}_c(S) \) and the analysis of \( \text{ApproxSCS} \) in the Bernoulli model, respectively.

With Lemma 17 below, we give a mixing model version of Lemma 14. Just as Lemma 7 and Theorem 10, our new mixing model versions of Lemma 6 and Theorem 9, are obtained by replacing \( |\ln p_{\text{max}}| \) in the old versions by \( c_{\text{mix}} \). Lemma 17 is simply Lemma 14, our old analysis of \( \text{BoundCompression} \)’s failure probability in the Bernoulli model, with \( |\ln p_{\text{max}}| \) replaced by \( c_{\text{mix}} \). It is easy to verify that using Lemma 17 instead of Lemma 14, our proofs of Lemma 6 and Theorem 9 immediately turn into proofs for Lemma 7 and Theorem 10. Hence, to generalize our results from the Bernoulli to the mixing model, it only remains to prove Lemma 17:

Lemma 17. Fix a mixing model and \( \varepsilon > 0 \). For a random input \( S \) for Shortest Common Superstring and

\[
c \geq \frac{1 + (\ln \Delta(S))/\ln n}{c_{\text{mix}}} + \varepsilon, \quad c = O(1),
\]

\( \text{BoundCompression}(S, c) \) runs in time \( O(\text{poly}(||S||)) \). It outputs “Fail” with probability \( \exp(-\Omega(n \log n)) \). If it outputs “Success”, \( \text{opt}_c(S) \leq cn \ln n \).
4.2. Generalization to the Mixing Model

Proof. We adapt the part of Lemma 14’s proof that upper bounds the probability that a fixed spanning tree configuration $C$ appears. Obviously, the number of possible configurations is unaffected by the random input model.

Fix a configuration $C$ and consider the equality graph $G_e(C)$. We assign numbers $\nu(s'_j)$ to its vertices, i.e., the letters $s'_j$ of $S$’s strings. The goal of numbering the strings is to divide each string into several substrings, called blocks below, according to the numbering. Then, we analyze the probability distribution of the blocks, assuming independence of blocks. Finally, the analysis is completed by considering the dependencies between the blocks of the strings.

We number the letters of the strings by traversing the spanning tree $T$ in the string graph $G_e(S)$. The numbers $\nu(s'_j)$ are assigned as follows. We start at $s^1$ and set $\nu(s'_j) := j$, $j = 1, \ldots, |s^1|$. Then, while unnumbered strings $s'$ exist, we choose such an $s'$ with an edge $e = \{s', s''\} \in T$ such that $s''$ is already numbered, and assign numbers to the letters in $s'$. Since in this process, the already numbered strings always induce a subtree of $T$, it is clear that such an unnumbered $s'$ always exists, and for a chosen $s'$, the edge $e$ is unique, i.e., no numbering conflicts arise. The numbers are assigned to the letters in $s'$ in the following way: For the letters $s'_j$ connected to a letter $s''_j$ by an edge in $G_e$, we set $\nu(s'_j) := \nu(s''_j)$. Then, we assign numbers to the not yet numbered letters of $s'$ such that $\nu(s'_{j+1}) = \nu(s'_j) + 1$ for $j = 1, \ldots, |s'| - 1$. In other words, consecutive letters get consecutive numbers.

The numbering for the equality graph in Fig. 4.2 is shown in Fig. 4.4.

![Equality Graph](image-url)

Figure 4.4: An Equality Graph $G_e$ and the Resulting Block Equality Graph $\overline{G}_e$.

For a block length $l \in \mathbb{N}$ (specified later), we divide the strings $s'$ into blocks of length at most $l$. For a string $s'$, all letters $s'_j$ with the same value $\lfloor \nu(s'_j)/l \rfloor$ form a block. For example, in Fig. 4.4, the block length $l = 2$ is used, and $s^1$ contains three blocks $s^1_1s^1_2$, $s^1_3s^1_4$, and $s^1_5$. As in the proof of Lemma 14, let $C$ contain the connected components $C_i$ of size at least two in the equality graph $G_e(C)$, and let $L := \bigcup_{i=1}^{|C|} C_i$ contain all letters in these components. In Fig. 4.4, letters in $L$ are marked grey. For each string $s'$, let the block string $\hat{s}' = \hat{s}'_1 \cdots \hat{s}'_{|s'|}$ be the sequence of the blocks $\hat{s}'_j$ in $s'$ from left to right which have full length $l$ and contain only letters from $L$. In Fig. 4.4, these blocks are marked with boxes, e.g., $\hat{s}'_4 = \hat{s}'_{14}\hat{s}'_{26}$. Since each block is a length $l$ substring of $s'$, $\hat{s}'_j$ is a string over the alphabet $\Sigma'$. Notice that the blocks $\hat{s}'_j$...
are chosen by the \( l \)-th order probability distribution \( P(l): \Sigma^l \to [0, 1] \) of our mixing model.

As mentioned above, in the following we first neglect the dependencies between the blocks in a string. Thus, we assume that the strings are created by a Bernoulli model over the alphabet \( \Sigma^l \) in which each letter is chosen according to \( P(l) \). Hence, each letter from \( \Sigma^l \) corresponds to a block in a string. In this model, assuming independence of blocks, we estimate the probability of the configuration \( C \). Later, we consider the dependencies and correct our calculations accordingly to finish our proof.

**The probability of \( C \) assuming independence of blocks** Consider blocks \( \hat{s}_i, \hat{s}'_j \) in two block strings \( \hat{s}, \hat{s}' \) such that for \( k = 1, \ldots, l \), the two letters in \( \hat{s}_i \) and \( \hat{s}'_j \) at position \( k \) are connected by an edge in the equality graph \( G_e \). We call such blocks *matching*. For example, in Fig. 4.4, \( \hat{s}_1 \) and \( \hat{s}'_1 \) have this property. If \( C \) appears, \( \hat{s}_i = \hat{s}'_j \), i.e., the two strings of length \( l \) given by the letters of the two blocks are equal. To capture this, we create a *block equality graph* \( \hat{G}_e = (\hat{V}, \hat{E}) \) for the block strings \( \hat{s}' \). We let \( \hat{V} \) be the set of all blocks, and \( \{\hat{s}_i, \hat{s}'_j\} \in \hat{E} \) if \( \hat{s}_i \) and \( \hat{s}'_j \) are matching. In Fig. 4.4, the block equality graph for the equality graph on the left is shown. In the proof of Lemma 14, we argued that in case a configuration \( C \) appears, the letters contained in a connected component of the equality graph \( G_e \) are assigned equal values from \( \Sigma \). Here, by construction of the block equality graph \( \hat{G}_e \), if the configuration \( C \) appears, all blocks contained in a connected component of \( \hat{G}_e \) have been assigned the same string from \( \Sigma^l \). As mentioned, we interpret this as a letter from a Bernoulli model over alphabet \( \Sigma^l \). It is clear that we can apply our arguments from the proof of Lemma 14 regarding the equality graph to the block equality graph.

We lower bound the number \( |\hat{E}| \) of edges in \( \hat{G}_e \). Consider an edge \( e = \{s', s''\} \) in the spanning tree \( T \) of \( C \). There are indices \( a, a' \) such that \( G_e \) contains the \( w(e) \) edges

\[
\{s'_{a+k}, s''_{a'+k}\}, \; k = 0, \ldots, w(e) - 1.
\]

Let \( L' \) be the set of all letters in these edges. For \( l - 1 \leq k \leq w(e) - l \), regardless of the positions of the blocks in \( s' \), \( s'_{a+k} \) lies in a block of full length \( l \) in \( L' \subseteq L \). Thus, there are at least \((w(e) - 2(l - 1))/l\) blocks \( \hat{s}_j \) for which all letters are contained in \( L \). In our numbering \( \nu \), two letters connected by an edge in \( G_e \) have the same number. Thus, due to the edges \( \{s'_{a+k}, s''_{a'+k}\} \), for every above block \( \hat{s}_j \), the letters in \( s' \) connected to \( \hat{s}_j \) by these edges form a block \( \hat{s}_j' \) in \( L \), and thus \( \hat{s}_j \) and \( \hat{s}'_j \) are matching. It follows that for every edge \( e \in T \), \( \hat{G}_e \) contains at least \((w(e) - 2(l - 1))/l\) edges.
In the following, we set the block length \( l := \ln \ln n \). With
\[
    w(T) = f(\delta) = (cn \ln n) + \delta
\]
we get
\[
    |\widehat{E}| \geq \sum_{e \in \mathcal{T}} \frac{w(e) - 2(l - 1)}{l} = \frac{w(T)}{l} - O(n) = (1 - o(1)) \cdot \frac{f(\delta)}{l}.
\]
Here, we used that
\[
    \frac{f(\delta)}{l} = \Omega(n \log n) = \omega(n)
\]
by choice of \( l = \ln \ln n \).

As argued, \( \widehat{G}_e \) is an equality graph as before, assuming alphabet \( \Sigma' \). For clarity, we denote probabilities assuming independence of blocks by \( \Pr_i \) and ones with dependencies with \( \Pr_d \). With \( |\widehat{E}| = (1 - o(1))f(\delta)/l \), (4.7) yields
\[
    \Pr_i[C \text{ appears}] \leq \exp \left( \ln(p_{\text{max}}') \cdot (1 - o(1)) \cdot \frac{f(\delta)}{l} \right)
    \leq \exp \left( -(1 - \varepsilon'/2) \cdot c_{\text{mix}}f(\delta) \right)
\]
for every fixed \( \varepsilon' > 0 \) by definition of \( c_{\text{mix}} \), cf. (1.27). (The reason for using \( \varepsilon'/2 \) instead of \( \varepsilon' \) will become apparent later.) We defer choosing a concrete value for \( \varepsilon' \) for a moment.

**The probability of \( C \) including block dependencies** We now consider block dependencies and correct (4.17) accordingly. In \( G_v \), for the set \( L \) of all letters contained in \( C \)'s components, \( |L| = f(\delta) + |C| \leq 2f(\delta) \) since \( |C| \leq f(\delta) \). (For every of the \( |C| \) components, there is at least one edge, and there are \( f(\delta) \) edges in total.) It follows that the total number of blocks is
\[
    \sum_{i=1}^n |\hat{s}_i| \leq 2f(\delta)/l = o(1)f(\delta),
\]
since \( l = \omega(1) \). Let an assignment be a function \( \hat{a} \) from the set of all blocks \( \hat{s}_j \) in our strings to \( \Sigma' \). Thus, \( \hat{a} \) assigns to each block \( \hat{s}_j \) a string \( \hat{a}_j \) of length \( l \). For a block string \( \hat{s} \), we let \( \mathcal{E}_j \) be the event that \( \hat{s}_j = \hat{a}_j \), i.e., the \( j \)-th block in \( \hat{s} \) is chosen according to \( \hat{a} \). Furthermore, let
\[
    \overline{\mathcal{E}}_j := \bigcap_{j'=1}^{j-1} \mathcal{E}_{j'}.
\]
Then, the $n$ block strings are chosen according to $\hat{a}$ if and only if for $i = 1, \ldots, n$, the event $\hat{E}_j^{[p]+1}$ occurs.

Remember that the function $\alpha$ limiting the dependencies in the mixing model has range $[0, 1)$. Hence, $\alpha(1) < 1$. Analogously to (4.16), for events $E_j^i$ and $\hat{E}_j^i$, since the letters regarding the two events have distance one, we get

$$\Pr_d \left[ E_j^i \mid \hat{E}_j^i \right] \leq (1 + \alpha(1)) \cdot \Pr_d[E_j^i] < 2 \cdot \Pr[E_j^i].$$

Here, we used $\Pr_d[E_j^i] = \Pr[E_j^i]$, since dependencies between blocks do not affect the distribution of an individual block. Accordingly,

$$\Pr_d \left[ \hat{E}_j^{[p]+1} \right] = \prod_{j=1}^{[p]} \Pr_d \left[ E_j^i \mid \hat{E}_j^i \right]$$

$$= \prod_{j=1}^{[p]} 2 \cdot \Pr[E_j^i]$$

$$= 2^{[p]} \cdot \Pr \left[ \hat{E}_j^{[p]+1} \right], \quad (4.19)$$

i.e., the dependencies increase the probability that $\hat{s}_i^j$ is chosen according to $\hat{a}$ by a factor of $2^{[p]}$ compared to neglecting dependencies. Let $A$ be the subset of all possible assignments such that $C$ appears if and only if for some $\hat{a} \in A$, we have $\hat{s}_i^j = \hat{a}_i^j$ for all blocks. Using (4.19) and (4.18), we get for a fixed assignment $\hat{a}$

$$\Pr_d \left[ \hat{E}_j^{[p]+1}, \ i = 1, \ldots, n \right] = \prod_{i=1}^{n} \Pr_d \left[ \hat{E}_j^{[p]+1} \right]$$

$$< \prod_{i=1}^{n} 2^{[p]} \cdot \Pr \left[ \hat{E}_j^{[p]+1} \right]$$

$$= 2^{\sum_{i=1}^{n} [p]} \prod_{i=1}^{n} \Pr \left[ \hat{E}_j^{[p]+1} \right]$$

$$= 2^{\alpha(1)f(\delta)} \cdot \Pr \left[ \hat{E}_j^{[p]+1}, \ i = 1, \ldots, n \right]. \quad (4.20)$$

Since by choice of $A$, with or without dependencies, we have

$$\Pr[C \text{ appears}] = \sum_{\hat{a} \in A} \Pr \left[ \hat{E}_j^{[p]+1}, \ i = 1, \ldots, n \right],$$

(4.20) together with (4.17) yields

$$\Pr_d[C \text{ appears}] = 2^{\alpha(1)f(\delta)} \cdot \Pr[C \text{ appears}]$$

$$\leq 2^{\alpha(1)f(\delta)} \cdot \exp(-(1 - \varepsilon')/2) \cdot c_{\text{mix}} f(\delta))$$

$$\leq \exp(-(1 - \varepsilon') \cdot c_{\text{mix}} f(\delta)) \quad (4.21)$$
Using (4.21), we finish our considerations of the dependencies by concluding that in the proof of Lemma 14, we can replace (4.7) by

\[ \Pr(a|C \text{ appears}) \leq \exp(-(1 - \epsilon')c_{\text{mix}}f(\delta)) \]

Thus, in the calculations following (4.7), \(|\ln p_{\text{max}}|\) is replaced by \((1 - \epsilon')c_{\text{mix}}\), and (4.9) becomes

\[
\Pr[\text{mxt}(S) = f(\delta)] \\
\leq \exp(((1 + k) - (1 - \epsilon'c_{\text{mix}}c)(n \ln n) + O(n \ln \ln n) + O(n)) .
\]  

(4.22)

It is easy to see that due to \(c \geq (1 + k)/c_{\text{mix}} + \epsilon\) in the lemma, an \(\epsilon' > 0\) exists with \(c \geq (1 + k)/(1 - \epsilon'c_{\text{mix}}) + \epsilon'\), yielding

\[
(1 - \epsilon'c_{\text{mix}}c \geq 1 + k + \epsilon'', \epsilon'' > 0 .
\]

We fix \(\epsilon'\) in that way. Now, in (4.22),

\[
(1 + k) - (1 - \epsilon'c_{\text{mix}}c \leq -\epsilon'' ,
\]

yielding \(\Pr[\text{mxt}(S) = f(\delta)] = \exp(-\Omega(n \log n))\) as before. The rest of Lemma 14’s proof works unmodified. □

### 4.3 Conclusions and Open Problems

We presented approximation schemes for **Shortest Common Superstring** in the Bernoulli, mixing, and a perturbation model, which guarantee length approximation ratio \(1 + \epsilon\) for arbitrarily small \(\epsilon > 0\), and have polynomial expected running time for random inputs from the considered models. We also showed that the probability that the greedy superstring has an approximation ratio worse than \(1 + \epsilon\) is \(e^{-\Omega(n \log n)}\) in these models. Using this, with respect to the famous greedy superstring conjecture, which claims that GreedySCS has approximation guarantee 2 in the length measure, we were able to show that only for a \(e^{-\Omega(n \log n)}\) fraction of all inputs, GreedySCS can fail to produce approximation ratio 2. (This is a bit simplified since actually we can only prove this for inputs \(S\) with \(\|S\| = \Omega(n \log n)\) and \(\Delta(S) = \text{poly}(n)\).)

In both the Bernoulli and mixing model, we obtained a tail bound on the optimal compression of a random input \(S\). We showed that the probability that \(\text{opt}(S)\) exceeds \((1 + \epsilon) \cdot \frac{n \ln n}{H}\) is \(e^{-\Omega(n \log n)}\), where \(H\) denotes the entropy of the model. This upper bound is tight up to a constant factor in the exponent.

Finally, we presented an algorithm which computes a shortest superstring for an input \(S\) in time \(O(2^n \cdot \text{poly}(\|S\|))\).
Open Problems  Our approximation scheme for Shortest Common Superstring only works in the length measure. It would be interesting to devise a similar algorithm for the compression measure. This would, however, require new ideas: In the analysis of our approximation scheme \texttt{ApproxSCS} in the length measure, we estimated the length of the greedy solution \( t \) by \( |t| \leq ||S|| \), i.e., we assumed that in the worst case, \texttt{GreedySCS} simply concatenates the strings in the input and achieves a compression of 0. We still achieve factor \( 1 + \epsilon \) in the length measure since the greedy superstring is only output if the optimal compression is small enough and since the total length of the strings in the input is assumed to be large enough. (Confer the proof of Theorem 9.)

Now consider the compression measure. Using our spanning tree technique, we could in fact efficiently compute an upper bound on the optimal compression which is at most \( (1 + \epsilon') \cdot (n \ln n)/H \) with probability exponentially close to 1. But obviously, to achieve factor \( 1 + \epsilon \) for compression, this upper bound only helps if we can efficiently compute a superstring which has compression at least \( (1 - \epsilon'') \cdot (n \ln n)/H \) with high probability. One could e.g. try to prove this for the greedy superstring. Our results, however, do not contain an analysis of the greedy compression, and as mentioned, one might need new ideas to perform such an analysis.
In this appendix, we present some mathematical definitions and known facts used throughout the above chapters. Also, proofs of some properties are given.

A.1 Basic Inequalities

In this section, we discuss some inequalities we used in this thesis.

**Jensen’s Inequality**  In Chapters 2 and 3, we used Jensen’s inequality for convex functions, which we present here.

**Definition 8.** A real-valued function \( f(x) \) defined over some interval \( I \) is convex if for all \( x_1, x_2 \in I \) and every \( \lambda \in [0, 1] \),

\[
f(\lambda \cdot x_1 + (1 - \lambda) \cdot x_2) \leq \lambda \cdot f(x_1) + (1 - \lambda) \cdot f(x_2).
\]

In the above chapters, we used the following well-known fact.

**Fact 1.** Let \( f(x) \) be a real-valued, twice differentiable function over some interval \( I \). Then, \( f(x) \) is convex over \( I \) if and only if the second derivative \( f''(x) \geq 0 \) for all \( x \in I \).

We can now state Jensen’s inequality for convex functions.

**Theorem 14** (Jensen’s inequality). Let \( f(x) \) be a real-valued, convex function over some interval \( I \). Let \( x_1, \ldots, x_n \in I \), and let \( \lambda_1, \ldots, \lambda_n > 0 \) with \( \sum_{i=1}^{n} \lambda_i = 1 \). Then,

\[
f\left(\sum_{i=1}^{n} \lambda_i \cdot x_i\right) \leq \sum_{i=1}^{n} \lambda_i \cdot f(x_i).
\]  \hspace{1cm} (A.1)

In Chapter 3, we used Jensen’s inequality for the expectation of a random variable. We can easily prove a version for finite probability spaces using Theorem 14. Clearly, all probability spaces used in Chapter 3 are finite.
Theorem 15 (Jensen’s inequality for random variables). Let \( \Omega \) be a finite probability space, \( f(x) \) be a real-valued function which is convex over some interval \( I \), and \( X : \Omega \rightarrow I \) be a random variable. Then,

\[
f(E[X]) \leq E[f(X)].
\]

**Proof.** Let \( \omega_1, \ldots, \omega_n \in \Omega \) be the elementary events in our probability space. By definition of \( X \), all values \( X(\omega_i) \) lie in the interval \( I \) over which the function \( f \) is convex. Furthermore, \( \sum_{i=1}^{n} \Pr[\omega_i] = 1 \) (and w.l.o.g. \( \Pr[\omega_i] > 0 \ \forall i \)). Therefore, we can apply Theorem 14 and get

\[
f(E[X]) = f \left( \sum_{i=1}^{n} \Pr[\omega_i] \cdot X(\omega_i) \right) \\
\leq \sum_{i=1}^{n} \Pr[\omega_i] \cdot f(X(\omega_i)) \\
= E[f(X)]. \]

\( \square \)

**Estimations for \( 1 - x \)** Another inequality we used regularly in Chapters 2 and 3 is the well-known upper bound \( 1 - x \leq e^{-x} \) for all \( x \in \mathbb{R} \). We prove the lower bound

\[
1 - x \geq e^{-2x}, \ 0 \leq x \leq 3/4,
\]

which is also used in the two chapters. Let \( f(x) = 1 - x \) and \( g(x) = e^{-2x} \). Clearly, \( f(0) = g(0) = 1 \). Furthermore, numerical evaluation (cf. also Figure A.1) shows that \( f(3/4) > g(3/4) \). Thus, since \( f(x) \) is linear, to show that \( f(x) \geq g(x) \) for \( 0 \leq x \leq 3/4 \), it suffices to show that \( g(x) \) is a convex function, i.e., the second derivative \( g''(x) \geq 0 \) for \( 0 \leq x \leq 3/4 \). We have \( g''(x) = -2 \cdot e^{-2x} \) and \( g''(x) = 4 \cdot e^{-2x} \geq 0 \), completing our proof.

Our results for approximating Independent Set and Coloring presented in Section 1.2.1 limit the possible edge probabilities \( p \) to \( p \leq 3/4 \). We claimed that we can extend our results to the upper bound \( p \leq 1 - \varepsilon \) for arbitrarily small \( \varepsilon > 0 \), since in our proofs, we need \( p \leq 3/4 \) only in the estimation \( 1 - x \geq e^{-2x} \). From the properties of the exponential function, it is easy to see that with the above method, we can show that for every \( \varepsilon > 0 \), if \( c > 0 \) is chosen large enough, we have \( 1 - x \geq e^{-cx} \) for \( 0 \leq x \leq 1 - \varepsilon \). For example, Figure A.1 shows that for \( \varepsilon = 0.1, c = 3 \) suffices. This proves our claim on extending the range of \( p \) in Section 1.2.1.

**A.2 Eigenvalues**

In this section, we present some definitions and known facts regarding eigenvalues of matrices, which are used in Chapters 2 and 3. We restrict our discussion to
real, symmetric matrices, since the matrices used in the two chapters are also real and symmetric by construction. In Section A.2.1, we prove a relation between the independence number of the random graphs from Chapter 3 and the largest eigenvalue of our auxiliary matrix. In Section A.2.2, we prove a relation between the trace of even powers of symmetric matrices and their eigenvalues, which was also used in Chapter 3.

**Definition 9.** Let $A \in \mathbb{R}^{n\times n}$ be a symmetric matrix. A scalar $\lambda \in \mathbb{R}$ is called an eigenvalue for $A$ if there is a vector $\vec{x} \in \mathbb{R}^n$, $\vec{x} \neq \vec{0}$, such that

$$A\vec{x} = \lambda \vec{x},$$

and in this case, $\vec{x}$ is called an eigenvector for $\lambda$.

It is known that a symmetric matrix $A \in \mathbb{R}^{n\times n}$ has $n$ real eigenvalues (including so called algebraic multiplicities, which we do not discuss here). We sort the eigenvalues of $A$ in non-increasing order and as usual denote them by $\lambda_1(A) \geq \ldots \geq \lambda_n(A)$. Thus, $\lambda_1(A)$ is the largest eigenvalue of $A$, which is the object of investigation in Chapters 2 and 3.
A.2.1 The Independence Number and Largest Eigenvalue of Random Graphs

The so-called Rayleigh quotients of a symmetric matrix are related to its eigenvalues, and can be used to estimate the latter.

**Definition 10.** For a symmetric matrix \( A \in \mathbb{R}^{n \times n} \) and a vector \( \vec{x} = (x_1, \ldots, x_n)^T \in \mathbb{R}^n \) with \( \vec{x} \neq \vec{0} \), the Rayleigh quotient \( R_A(\vec{x}) \) is defined as

\[
R_A(\vec{x}) = \frac{\vec{x}^T A \vec{x}}{\vec{x}^T \vec{x}}.
\]

For later use, notice that by definition,

\[
\vec{x}^T \vec{x} = \sum_{i=1}^{n} x_i^2 \quad (A.2)
\]

and

\[
\vec{x}^T A \vec{x} = \sum_{i,j=1}^{n} x_i a_{ij} x_j \quad (A.3)
\]

It is known that, given a symmetric matrix \( A \in \mathbb{R}^{n \times n} \), for all vectors \( \vec{x}, \vec{0} \) (not only eigenvectors), it holds that

\[
\lambda_1(A) \geq R_A(\vec{x}) \geq \lambda_n(A),
\]

i.e., the Rayleigh quotient of every vector \( \vec{x} \neq \vec{0} \) lies between the smallest and the largest eigenvalue of \( A \). The Rayleigh-Ritz theorem strengthens this statement:

**Theorem 16** (Rayleigh-Ritz). Let \( A \in \mathbb{R}^{n \times n} \) be a symmetric matrix. Then,

\[
\lambda_1(A) = \max_{\vec{x} \neq 0} R_A(\vec{x}) \quad \text{and} \quad \lambda_n(A) = \min_{\vec{x} \neq 0} R_A(\vec{x}).
\]

Lemma 5 claims that for every graph \( G \) drawn from the \( G(G, \varepsilon) \) model, the largest eigenvalue of the auxiliary matrix \( A(G, G, \varepsilon) \) (cf. (1.13)) is an upper bound on the independence number of \( G \). We use Theorem 16 to prove this. Let \( G = (V, E) \) be a random graph drawn from \( G(G, \varepsilon) \). We prove that

\[
\alpha(G) \leq \lambda_1(A(G, G, \varepsilon)),
\]

regardless of the choice of the adversary graph \( G \), the flip probability \( \varepsilon > 0 \), and the outcome of the random graph \( G \). Fix a maximum independent set \( I^* \subseteq V \) in \( G \), and let \( \vec{x}^* = (x_1^*, \ldots, x_n^*)^T \in \{0, 1\}^n \) with \( x_i^* = 1 \) if and only if \( i \in I^* \) be the characteristic vector of \( I^* \). Our auxiliary matrix \( A = A(G, G, \varepsilon) = (a_{ij}) \) is real and symmetric by construction. We determine the Rayleigh quotient \( R_A(\vec{x}^*) \).
A.2. Eigenvalues

Consider indices $1 \leq i, j \leq n$. If $i, j \in I^*$, it holds that $x_i^* x_j^* = 1$ by construction of $\vec{x}^*$, and $a_{ij} = 1$ by construction of $A$ since $I^*$ is independent. Therefore, for $i, j \in I^*$, we have $x_i^* a_{ij} x_j^* = 1$. If at least one of $i, j$ lies outside $I^*$, $x_i^* x_j^* = 0$ and consequently, $x_i^* a_{ij} x_j^* = 0$. Therefore, the Rayleigh quotient $R_A(\vec{x}^*)$, we can infer

$$R_A(\vec{x}^*) = \frac{(\vec{x}^*)^T A \vec{x}^*}{(\vec{x}^*)^T \vec{x}^*} = \frac{\sum_{i,j=1}^{n} x_i^* a_{ij} x_j^*}{\sum_{i=1}^{n} (x_i^*)^2} = \frac{\sum_{i,j \in I^*} 1}{|I^*|} = \frac{|I^*|^2}{|I^*|} = |I^*|.$$

(A.4)

In deriving (A.4), we used (A.2) and (A.3) from above. Since $A$ is real and symmetric, we can apply Theorem 16 and get

$$\lambda_1(A) = \max_{\vec{x} \neq \vec{0}} R_A(\vec{x}) \geq R_A(\vec{x}^*) = |I^*| = \alpha(\mathcal{G}),$$

which completes the proof.

A.2.2 The Trace of Even Powers of Symmetric Matrices

In Section 3.2, we used a relation between the largest eigenvalue of an even power of a real, symmetric matrix, and its trace. We give some definitions and prove the relation. We start by defining diagonalizability and similarity of matrices.

**Definition 11.** Two matrices $A, B \in \mathbb{R}^{n \times n}$ are similar if a regular matrix $Q \in \mathbb{R}^{n \times n}$ exists such that $B = Q^{-1} A Q$. A matrix $A \in \mathbb{R}^{n \times n}$ is diagonalizable if a diagonal (i.e., all off-diagonal entries are zero) matrix $D \in \mathbb{R}^{n \times n}$ exists such that $A$ and $D$ are similar.

It is known that similar matrices have the same eigenvalues, including algebraic multiplicities. Let $A, D \in \mathbb{R}^{n \times n}$ be similar matrices with $D$ diagonal. Since the eigenvalues (with algebraic multiplicities) of $D$ are exactly its diagonal entries $d_{ii}$, $i = 1, \ldots, n$, by similarity of $A$ and $D$, the $d_{ii}$ are the $n$ eigenvalues of $A$. We exploit this below.

Now, let $A \in \mathbb{R}^{n \times n}$ be a symmetric matrix and $k \in \mathbb{N}$ be even. In the following, we show that all eigenvalues of the $k$-th power $A^k$ of $A$ are nonnegative. It is known that real, symmetric matrices are diagonalizable. Let $D$ be a diagonal matrix similar to $A$ and let $Q$ be the regular matrix with $D = Q^{-1} A Q$. Clearly,

$$A^k = (QDQ^{-1})^k = QD^k Q^{-1}.$$ 

Therefore, $A^k$ and $D^k$ are similar by definition and have the same eigenvalues. Furthermore, $D^k$ is also diagonal, and hence, the eigenvalues of $A^k$ are the diagonal entries of $D^k$. Let $d_{ij}^k$ denote the entries of $D^k$. Observe that for $i = 1, \ldots, n$, we have $d_{ii}^k = (d_{ii})^k$, i.e., the diagonal entries of $D^k$ are the diagonal entries of $D$ raised
to the power of $k$. Since $k$ is even, all diagonal entries $d_{ii}^k$ of $D^k$ are nonnegative, and since these values are exactly the eigenvalues of $A^k$ by the above discussion, we have proved the following lemma:

**Lemma 18.** Let $A \in \mathbb{R}^{n \times n}$ be a symmetric matrix and $k \in \mathbb{N}$ be even. Then,

$$
\lambda_1(A^k) \geq \ldots \geq \lambda_n(A^k) \geq 0 .
$$

As mentioned in Chapter 3, the trace of a matrix $A \in \mathbb{R}^{n \times n}$ is defined as

$$
\text{tr}(A) = \sum_{i=1}^{n} a_{ii} ,
$$

i.e., the sum of the diagonal entries of $A$. It is known that for a real matrix $A \in \mathbb{R}^{n \times n}$, it holds that

$$
\text{tr}(A) = \sum_{i=1}^{n} \lambda_i(A) , \quad (A.5)
$$

i.e., the trace of $A$ is the sum of its eigenvalues. Now, let $A \in \mathbb{R}^{n \times n}$ be symmetric and $k \in \mathbb{N}$ be even. Lemma 18 yields that all eigenvalues of $A^k$ are nonnegative. Together with (A.5), it follows that $\text{tr}(A^k) = \sum_{i=1}^{n} \lambda_i(A^k) \geq \lambda_1(A^k)$. We have proved the following lemma:

**Lemma 19.** Let $A \in \mathbb{R}^{n \times n}$ be a symmetric matrix and $k \in \mathbb{N}$ be even. Then,

$$
\text{tr}(A^k) \geq \lambda_1(A^k) .
$$
Bibliography


<table>
<thead>
<tr>
<th></th>
<th>Algorithm</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GreedyColor($H$)</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>Meta-IndSet($H, p$)</td>
<td>23</td>
</tr>
<tr>
<td>3</td>
<td>GreedySCS($S$)</td>
<td>34</td>
</tr>
<tr>
<td>4</td>
<td>ApproxIndSet($H, p, \varepsilon$)</td>
<td>53</td>
</tr>
<tr>
<td>5</td>
<td>ImprovedIndSet($H, p$)</td>
<td>57</td>
</tr>
<tr>
<td>6</td>
<td>ApproxColoring($H, p, \varepsilon$)</td>
<td>61</td>
</tr>
<tr>
<td>7</td>
<td>ImprovedColoring($H, p, \varepsilon$)</td>
<td>62</td>
</tr>
<tr>
<td>8</td>
<td>SmallEdgePrIndSet($H, p$)</td>
<td>70</td>
</tr>
<tr>
<td>9</td>
<td>AdversaryIndSet($G, G, \varepsilon$)</td>
<td>94</td>
</tr>
<tr>
<td>10</td>
<td>Partition($S$)</td>
<td>104</td>
</tr>
<tr>
<td>11</td>
<td>BoundCompression($S, c$)</td>
<td>106</td>
</tr>
<tr>
<td>12</td>
<td>MaxHamiltonianPath($G, w$)</td>
<td>112</td>
</tr>
<tr>
<td>13</td>
<td>ShortestSupStr($S$)</td>
<td>113</td>
</tr>
<tr>
<td>14</td>
<td>ApproxSCS($S, c$)</td>
<td>114</td>
</tr>
</tbody>
</table>
Index

adversary, 4, 6, 27, 40
alphabet, 32
analysis
  average-case, 3
  probabilistic, 4, 8
  smoothed, 4, 6
  worst-case, 3
approximation
  algorithm, 16
  guarantee, 16
  ratio, 16
  scheme, 16
Bernoulli model, 4, 36
  entropy, 37
chromatic number, 18, 32, 61
coloring, 18
  feasible, 18
  greedy, 21, 49
  partial, 50, 80
COLORING problem, 3, 18, 49
compression, 33
  optimal, 33, 35, 37, 41, 103
degree
  potential, 28
  probability, 20
efficient
  average-case, 3
  worst-case, 1
effort, 10
eigenvalue
  largest, 23, 26, 84
entropy, 37
equality graph, 108, 119

factor, see approximation guarantee
flip probability, 28

G(n, p) model, 4, 5, 8, 20, 27
graph, 18
  random, 4, 26
  semi-random, 4, 27

Hamiltonian path, 111

H_d(n, p) model, 20, 49
hypergraph, 18
  random, 4, 20, 49
  uniform, 18

independence number, 18, 53, 83
independent set, 18, 31
  greedy, 22, 28, 49, 80
  maximum, 18
INDEPENDENT SET problem, 3, 18, 49, 79
input, 16
  model, 3, 8
  random, 3, 5
  real-world, 2, 5
  semi-random, 4, 8
  worst-case, 2

mixing
  model, 4, 37
  entropy, 38
  property, 37

non-neighborhood, 56

optimization problem, 16
overlap, 33

performance
  average-case, 5
Index

- typical, 2
- worst-case, 3
- perturbation, 4, 6
- model, 27, 39
- prefix, 33
- projection graph, 55
- PTAS, 16, 34
  - probabilistic, 35, 43, 103

running time
  - expected, 3, 10
  - worst-case, 1

**Shortest Common Superstring problem**, 3, 32, 103
- spanning tree, 35, 103, 104
  - configuration, 107, 119
- spectral technique, 23, 25
- stochastic process, 36
- string, 32
  - graph, 104
    - directed, 113
  - length, 32
  - random, 36
  - semi-random, 39
  - substring, 33
  - superstring, 33
  - greedy, 34, 38, 43
  - shortest, 33, 43, 112
- suffix, 33

trade-off, 12, 25, 28, 76, 101