Search Processes
and their Average Case Analysis

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY
ZURICH

for the degree of
Dr. sc. techn.

presented by
NICOLA GALLI
Dipl. Informatik-Ing. ETH
born 28.5.1971
citizen of Besazio, Switzerland

accepted on the recommendation of
Prof. Dr. Klaus Simon, examiner
Prof. Dr. Emo Welzl, co-examiner

1998
Acknowledgments

I am grateful to my advisor Prof. Klaus Simon who gave me the opportunity to study a lot of interesting problems which mostly arised from stimulating discussions. I am also thankful to Prof. Emo Welzl for accepting to be my co-advisor and for his valuable comments.

Many thanks to Prof. Andrew D. Barbour, Johannes Blöhmer, Giovanni Cesari, Joachim Giesen, Thomas Raschle, Bernhard Seybold, Falk Tschirschnitz, and Bernhard von Stengel for fruitful discussions and helpful comments. I also thanks Thomas Bickel, Jakob Magum, Paul Trunz, and all other colleagues of the Institute for Theoretical Computer Science for the pleasant atmosphere during the last three years.

Finally, a special thanks to my parents and my brothers for their constant support.
Abstract

The last decade is characterized by a rapid growth of the amount of information stored worldwide. The role of search strategies is, therefore, a key aspect of the management of data. Many algorithms have been developed to obtain more and more efficient solutions where time and space complexity have been mostly considered to measure the efficiency.

The traditional type of complexity treatment is the so called worst case analysis. However, this approach is often rather pessimistic. The observed behavior is in many cases much better than the worst case. Therefore, it is meaningful to analyze the average performance. But this approach involves a new problem: A probabilistic model is needed. The major difficulty is to find a good compromise to get both a general and simple model.

In this thesis, we handle two basic problems from this point of view. First, we consider the search in graphs. Starting from an average case analysis of Breadth-First-Search, we propose a general model to investigate a vast class of graph algorithms. This model is also useful to analyze structural properties of random graphs. Then, we consider the search in ordered sets, namely the dictionary problem. We study two different solutions: Randomized search trees and Hashing. More precisely, we present a realization of randomized search trees which extends the traditional solution [6] in the weighted case and which points out the affinity with balanced search trees. Then, we propose a perfect hashing scheme based on the compression of sparse tables. The resulting perfect hash function is surprisingly simple and efficient (in terms of time and space complexity).
Riassunto

L’ultimo decennio è stato caratterizzato da una rapida crescita del volume di informazioni memorizzate in tutto il mondo. Il ruolo delle strategie per la ricerca dei dati è quindi divenuto un aspetto determinante per la loro gestione. Molti algoritmi sono stati sviluppati per ottenere soluzioni sempre più efficienti.

Tradizionalmente gli studi riguardanti l’efficienza di un algoritmo – misurata soprattutto in termini di tempo e di memoria impiegate – sono basati sull’analisi del caso peggiore (worst case analysis). Però i risultati si sono rilevati spesso pessimistici se confrontati con valori empirici. Di conseguenza si è pensato di analizzare il comportamento medio di un algoritmo (average case analysis). Purtroppo anche questo approccio non è privo di problemi: infatti richiede un modello probabilistico. La difficoltà maggiore consiste nel trovare un modello generale e semplice allo stesso tempo.

Contents

1 Introduction 1

2 Preliminaries 7
   2.1 Discrete Probability Theory 7
      2.1.1 Probability Space and Events 7
      2.1.2 Random Variables 8
      2.1.3 Bounds 10
   2.2 Graph Theory 10
      2.2.1 Undirected Graphs 11
      2.2.2 Directed Graphs 11
      2.2.3 Further Definitions 12
   2.3 Asymptotics 13

I Searching in Graphs 15

3 Breadth-First-Search 17
   3.1 Introduction 17
   3.2 Discrete Time Disintegration Process 19
   3.3 The Distribution of Isolated Vertices 21
   3.4 Exploring a Random Graph 27
   3.5 Conclusion 33
## 4 Graph Exploration

4.1 Introduction ........................................ 35
4.2 The Model .......................................... 37
4.3 Node Exploration ................................. 38
4.4 Searches with Local Information .............. 48
4.5 Shortest Paths and Minimum Spanning Tree .... 50
4.6 Non-homogeneous Random Graphs ............ 52
4.7 Extended Node Exploration .................... 57
4.8 Conclusion ........................................ 59

## II Searching in Sets

5 Randomized Search Trees ............................ 63
5.1 Introduction ......................................... 63
5.2 Weighted Random Search Trees ................. 67
5.3 Algorithmic Aspects ............................... 74
5.4 Conclusion .......................................... 77

6 Hashing ................................................. 79
6.1 Introduction ......................................... 79
6.2 Perfect Hashing and Table Compression .......... 81
6.3 Table Compression with Random Entries .......... 83
6.4 Tetris-Hashing ..................................... 90
6.5 Dynamic Tetris-Hashing .......................... 93
6.6 Experiments ......................................... 94
6.7 A Related Problem ................................ 100
6.8 Conclusion ......................................... 105

7 Conclusion ............................................. 107
Chapter 1

Introduction

In almost all non-trivial problems in computer science, there is need to store and access a large amount of data. A typical example are bank accounts (but also patient information in a hospital or airport traffic information, etc.). Actually, the problem is not only to store the static information (e.g. the balance of the account) but also to manage all transactions associated to them (like withdraw or deposit operations). Usually, transactions need to test integrity conditions. Furthermore, also the whole history of the transactions is often needed. This requires to frequently access the data. Therefore, search operations must be implemented in a particularly efficient way.

Usually, the information is associated to a key. Its function is to uniquely identify the items to simplify the management of the data (e.g. with account numbers and transaction numbers) or to build data structures (e.g. indices) to search the information more efficiently. In this way, the problem is reduced to the basic question of maintaining an ordered key set under insert, delete, and access (search) operations. This problem is known as the dictionary problem.

Recently, the growing of communication systems like internet and the delocalization of data renewed the interest in other search problems in which the data is not structured. The absence of structure modifies the problem significantly since no key is available anymore. For example, a simple search of information on the web by using standard search machines like Alta Vista, Magellan, or Yahoo is not very efficient. The problem is that all data stored world wide which seem to be relevant is
returned. Unfortunately, lots of useless information is mixed with the interesting one. As alternative, one can send a web-roboter on the net which, starting from a given site, follows (relevant) links until the suitable pages – or web sites – are found. The advantage is that the search is local and the user can better interact with the roboter, filtering the collected data.

The problem of discovering some information by following links (e.g. by surfing in the net) can be formalized as follows. We are given a set of locations (nodes) and a set of connections between them (edges). Then we are asked to visit (reach) the nodes starting from a given location by traversing the edges. This corresponds to an exploration of a network.

The importance of these questions motivated many computer scientists to look for efficient algorithms and data structures to solve them. The simple linear lists were rapidly replaced by (balanced) trees, several variants of heaps (e.g. Fibonacci Heaps [17]), or hashing. For an overview of these topics see for instance [21, 32]. For graph problems, the basic algorithms Breadth-First-Search and Depth-First-Search were extended to solve lots of problems like shortest paths, spanning trees [11], and flow problems. Further algorithms were stepwise refined and adapted to special types of graphs. We refer to [2, 13] for a survey.

But how can we know if a given algorithm is efficient? Two parameters are usually used to express the efficiency:

- the time complexity and
- the space complexity.

These measures should enable us to predict the behavior of an algorithm for a given input and to decide if it is reasonably efficient. To simplify the task, the input is partitioned into classes and the parameters, time or memory requirements, are determined for each class. Often, the classification is induced by the size of the input. Then a typical result is, for instance, that the running time is linear in the input size (namely, the time needed to handle the input is doubled whenever the input size is doubled).

There are two meaningful approaches to analyze the complexity of an algorithm. We can bound the time and the memory used to process any input of a given class. We say that the result holds in the worst case. This kind of analysis, however, can be very pessimistic. Frequently, it happens that only few instances of the input – which occur rarely in
practice – considerably increase the complexity of an algorithm. In this case, we judge an algorithm less efficient although it works well in practice. Therefore, it seems appropriate to consider the average performance of an algorithm, the *average case*, to bring the prediction closer to reality.

Average case analysis became popular in the sixties thanks to the pioneering work of Knuth [26, 27, 28]. Since then, a lot of results have been shown which pointed out the discrepancy between worst case and average case, see e.g. [15, 35]. The average case analysis matched much better the empirical observations. Furthermore, the algorithms designed to have a good average behavior were more elegant and simple than the ones thought to limit the costs in the worst case. A possible explanation is that the algorithms must be extended to handle few hard cases to improve the performance in the worst case. However, this optimism for average case analysis has to be redimensioned. In fact, a new intrinsic problem arises whenever an expectation is computed: A *probabilistic model* for the input is required.

Often, no assumption about the input is justifiable. In this case, the uniform distribution is preferred – however, it is not always the most appropriate. The choice of the input model is, indeed, one of the most critical problems in the analysis of algorithms. On the one hand, we can choose a general model. The price is a complex analysis. On the other hand, we can choose a simple model, loosing generality. Then, what is a good compromise? Perhaps, the following sentence of Box [40, p.4] suggests the answer:

"All models are wrong, but some are useful."

Any input model represents only a facet of the reality. Therefore, one can always find arguments to corroborate the choice or to criticize it. The point is that it is not sufficient to show a certain behavior of the algorithm. It is much more important to show why it is so. Thus, a useful model should satisfy these requirements:

1. it should be as representative as possible,
2. it should suggest what happens in other cases (models),
3. the problem should become intelligible.
To find such a model, a deep understanding of the problem is required. To achieve this, it is necessary to find an adequate description of the problem: Its core must be detected and the relevant parameters have to be determined. The goal is to find a representation which is as simple as possible, such that the corresponding analysis gives a convincing and intuitive explanation of the observed phenomena.

Furthermore, one should resist the temptation to approximate the model too early. In fact, small changes can dramatically modify its nature. Even though the results of the analysis matches the real behavior, the whole intuition of the causes can be completely lost. Therefore, it is important to start with an exact analysis and to make approximations as late as possible. From this perspective, it turns out to be fruitful to interpret the problems as stochastic processes, as illustrated for example in [8, 37, 43].

Of course, an exact analysis costs a lot of additional effort. Is it really necessary? What is this effort good for? There are at least two answers. The first one is that a better understanding of the problem (process) allows us to make better design decisions. Roughly, if we do not know the problem, then we cannot solve it. The second answer is that by analyzing the problem exactly it is possible to discover relationships with other problems — for which, maybe, good solutions are already known. Moreover, these relationships can also suggest generalizations of the problem which can be used for other purposes. An example is given by DEVROYE [9]. Starting from several distinct analyses of tree structures, he developed a general model to analyze them simultaneously. His model was applicable to many other tree structures. Results in this direction contribute to build a general, elegant theory started by KNUUTH [26] and enriched, among other, by FLAJOLET and SEDGEMICK [15].

This thesis must be situated in this context. We consider the two search problems discussed previously. First, we investigate the exploration of a graph. We start with the analysis of Breadth-First-Search in a random graph (Chapter 3). We study the number of edges traversed until all nodes are reached. The corresponding distribution is closely related with the probability that the graph is connected. In particular, the analysis proposed offers an algorithmic, intuitive approach to investigate structural properties of graphs. For instance, we estimate the size of the connected component containing a given node $s$ (extending some results of KARP [24]).
From the analysis of Breadth-First-Search we derive in Chapter 4 a general model which allows us to study other exploration strategies, the connectedness of generalized random graphs (namely non-homogeneous random graphs), and weighted problems like minimum spanning trees and the single source shortest paths problem. In particular, we give lower bounds for the number of edges traversed to solve the problems (which matches the upper bounds).

Then, we consider two solutions for the dictionary problem. In Chapter 5, we analyze (weighted) randomized search trees. The analysis points out the affinity with balanced search trees and QuickSort. Furthermore, this problem illustrates how an exact analysis can lead to the right design decisions for the algorithm. In fact, the derived realization extends the traditional solutions in the weighted case.

Finally, in Chapter 6 we propose a surprisingly simple and efficient perfect hash function based on the ambivalence between hashing and table compression. For a given (static) key set, the hash table is minimal. For dynamic key sets, the resulting perfect hash function can maintain the table nearly minimal. For this last problem, theoretical and experimental arguments are integrated.

Experiments — e.g. simulations and statistics — were traditionally used to illustrate how the theory matched empirical results which were interpreted as the real world. However, people have realized that experiments are not necessarily the real world since they are also subjected to the given model. Hence, experimental analysis is not only an instrument to verify a theory but also an alternative to the mathematical approach, especially when the theoretical arguments are too hard. Indeed, a good use of focused experiments can contribute substantially to the solution of a problem. Furthermore, interesting conjectures can arise from experimental results. This represents perhaps the most exciting aspect of experimental analysis from a theorists point of view.

This induced the study of new methodologies, as it is well illustrated by McGeoch [30]. The two approaches, theoretical and experimental, show a kind of complementarity. On the one hand, it is easier to obtain experimental results, but it is more difficult to interpret them correctly. In particular, if some measurements are conducted for several input sizes, e.g. for \( n = 1, \ldots, 1000 \), then it is not obvious to extrapolate the values for \( n > 1000 \). One can propose hypotheses, but they are no proofs. On the other hand, theoretical analysis often uses the \( O \)-notation, i.e. constant factors are ignored. Then it can happen that
these hidden constants are huge in reality. Hence, an algorithm could appear efficient (in terms of complexity) without being it in practice.

For the analysis of hashing, we exploit this complementarity. In particular, the mathematical part is based on a specific model (random keys) while the experiments suggest that the result holds in general. Moreover, some conjectures related to pattern matching are formulated at the end of the chapter.
Chapter 2

Preliminaries

2.1 Discrete Probability Theory

This is an introduction to discrete probability theory. The purpose is not to give a self-contained treatment of probability theory, but only to review the minimal knowledge and notation needed in the next chapters. For more details we refer to [3, 14, 34, 35].

2.1.1 Probability Space and Events

A discrete probability space consists of a finite or countably infinite set \( \Omega \), the sample space, and a probability measure. The elements of \( \Omega \) are called elementary events and a subset \( \mathcal{E} \subseteq \Omega \) is an event. The probability measure \( \Pr(.,:) \), also called distribution, is a mapping from the space \( \Omega \) to \([0,1] \) such that

1. \( \Pr(\mathcal{E}) \geq 0 \) for any event \( \mathcal{E} \subseteq \Omega \),
2. \( \Pr(\Omega) = 1 \), and
3. for every two events \( \mathcal{E}_1, \mathcal{E}_2 \subseteq \Omega \) with \( \mathcal{E}_1 \cap \mathcal{E}_2 = \emptyset \), it holds

\[
\Pr(\mathcal{E}_1 \cup \mathcal{E}_2) = \Pr(\mathcal{E}_1) + \Pr(\mathcal{E}_2).
\]

In general, the probability of the union of any two events satisfies

\[
\Pr(\mathcal{E}_1 \cup \mathcal{E}_2) = \Pr(\mathcal{E}_1) + \Pr(\mathcal{E}_2) - \Pr(\mathcal{E}_1 \cap \mathcal{E}_2)
\]
which implies
\[ \Pr(\mathcal{E}_1 \cup \mathcal{E}_2) \leq \Pr(\mathcal{E}_1) + \Pr(\mathcal{E}_2). \]

Two events are called \textit{independent} if
\[ \Pr(\mathcal{E}_1 \cap \mathcal{E}_2) = \Pr(\mathcal{E}_1) \cdot \Pr(\mathcal{E}_2). \]

The \textit{conditional probability}
\[ \Pr(\mathcal{E}_1 \mid \mathcal{E}_2) \]
of an event $\mathcal{E}_1$ given $\mathcal{E}_2$ is defined as
\[ \Pr(\mathcal{E}_1 \mid \mathcal{E}_2) = \frac{\Pr(\mathcal{E}_1 \cap \mathcal{E}_2)}{\Pr(\mathcal{E}_2)} \]
whenever $\Pr(\mathcal{E}_2) > 0$. For independent events it holds obviously
\[ \Pr(\mathcal{E}_1 \mid \mathcal{E}_2) = \Pr(\mathcal{E}_1). \]

\subsection*{2.1.2 Random Variables}

A \textit{random variable} $X$ is a mapping from $\Omega$ to a given (discrete) set $\mathcal{X}$. We will usually consider the case $\mathcal{X} = \mathbb{N}$. This mapping assigns a value $x \in \mathcal{X}$ to each elementary event and the probability distribution of $X$ corresponds to
\[ \Pr(X = x) = \sum_{\omega \in \Omega: X(\omega) = x} \Pr(\omega). \]

The conditional distribution of $X$ given an event $\mathcal{E}$, denoted by
\[ \Pr(X = x \mid \mathcal{E}), \]
is accordingly defined. If the conditioning event involves another random variable $Y$ defined on the same sample space, the conditional probability distribution of $X$ given $Y = y$ is
\[ \Pr(X = x \mid Y = y) = \frac{\Pr(X = x \land Y = y)}{\Pr(Y = y)}. \]

The \textit{expected value} of a discrete random variable $X$ is
\[ \mathbb{E}(X) = \sum_x x \cdot \Pr(X = x). \]
Let \( \varphi \) be a function from \( \mathcal{X} \) to \( \mathcal{X} \). Then the previous definition is easily extended to

\[
E(\varphi(X)) = \sum_x \varphi(x) \cdot \Pr(X = x).
\]

The variance of \( X \) is defined as

\[
\text{Var}(X) = E\left( (X - E(X))^2 \right).
\]

**Lemma 2.1** Let \( X \) and \( Y \) be random variables, and \( \alpha \) a real number. Then

\[
\begin{align*}
(2.1) \quad & E(X + Y) = E(X) + E(Y), \\
(2.2) \quad & E(\alpha \cdot X) = \alpha \cdot E(X),
\end{align*}
\]

and, if \( X \) and \( Y \) are independent,

\[
\begin{align*}
(2.3) \quad & \text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y), \\
(2.4) \quad & \text{Var}(\alpha \cdot X) = \alpha^2 \cdot \text{Var}(X).
\end{align*}
\]

It follows that

\[
\text{Var}(X) = E(X^2) - E(X)^2.
\]

By definition, the conditional expected value of \( X \) given an event \( \mathcal{E} \) is

\[
E(X \mid \mathcal{E}) = \sum_x x \cdot \Pr(X = x \mid \mathcal{E}).
\]

**Lemma 2.2** Let \( X \) be a random variable, \( \mathcal{E} \) an event, and \( \tilde{\mathcal{E}} = \Omega - \mathcal{E} \) its complementary event. Then

\[
E(X) = E(X \mid \mathcal{E}) \cdot \Pr(\mathcal{E}) + E(X \mid \tilde{\mathcal{E}}) \cdot \Pr(\tilde{\mathcal{E}}).
\]

Of course, this can be extended for any family of pairwise disjoint events with \( \bigcup_{i=1}^n \mathcal{E}_i = \Omega \), namely

\[
E(X) = \sum_{i=1}^n E(X \mid \mathcal{E}_i) \cdot \Pr(\mathcal{E}_i).
\]
2.1.3  Bounds

The expected value and the variance of a random variable can be used to bound some useful probabilities. Let $X$ be any positive-valued random variable and $t > 0$. Then

$$\Pr(X \geq t) \leq \frac{E(X)}{t}$$

$$\Pr(X - \mu \geq t) \leq \frac{\text{Var}(X)}{t^2 + \text{Var}(X)^2}$$

$$\Pr(|X - \mu| \geq t) \leq \frac{\text{Var}(X)}{t^2}$$

which are known as Markov, Cantelli, and Chebychev inequalities, respectively.

For the special case that $X$ takes on only values in $\mathbb{N}$, it follows

$$\Pr(X > 0) \leq E(X)$$

and

$$\Pr(X = 0) \leq \frac{\text{Var}(X)}{E(X)^2}.$$  

The last bound is usually called second moment method.

2.2  Graph Theory

The graph theoretical notation and the definitions are quite standard and can be found, extended, in [13, 23].

A graph $G = (V, E)$ consists of a vertex or node set $V$ and an edge set $E$. The nodes can be represented by points, and edges by connections between two points. If we consider the connections oriented, then the graph is said directed, otherwise it is undirected. The notation associated to a graph is similar in both cases. However, for directed graphs (also called digraphs) the definitions are often doubled because of the direction of the edges. Therefore, we will present the definition first for undirected graphs, and successively for digraphs.
2.2. Graph Theory

2.2.1 Undirected Graphs

In an undirected graph $G = (V, E)$, the set of edges $E$ consists of unordered pairs $\{v, w\}$ of nodes in $V$. In general, we assume $v \neq w$. The set

$$\Gamma(v) \overset{\text{def}}{=} \{w \in V \mid \{v, w\} \in E\}$$

contains the nodes adjacent to $v$, and its size, called degree of $v$, is denoted by $\gamma(v) = |\Gamma(v)|$. The definition can be extended to sets of vertices. Let $A \subseteq V$, then

$$\Gamma(A) \overset{\text{def}}{=} \bigcup_{v \in A} \Gamma(v).$$

An isolated node is a node $v$ with $\gamma(v) = 0$.

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two undirected graphs. Then $G_1$ is a subgraph of $G_2$ if $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$.

A path from node $v_0$ to node $v_k$ is a sequence of vertices $v_0, \ldots, v_k$ such that $\{v_i, v_{i+1}\} \in E$ for all $i = 0, 1, \ldots, k-1$. The path is simple if all its vertices are distinct. A circuit is a path with $k \geq 3$ which starts and ends in the same node, namely $v_0 = v_k$. A graph without circuits is a forest. A vertex $w$ is reachable from a vertex $v$ if there is a path from $v$ to $w$ containing at least two vertices. The distance from $v$ to $w$ is the number of edges in the shortest simple directed path between them.

An undirected graph is connected if for any pair $v, w$ of vertices there is a path from $v$ to $w$. The maximal connected subgraphs of $G$ are called connected components (or short components). A connected forest is called tree. A subgraph which is a tree and contains all nodes of the graph is called a spanning tree.

2.2.2 Directed Graphs

If the graph is directed, then an edge corresponds to an ordered pair of distinct vertices $(v, w) \in V \times V$. Then, the edge $(v, w)$ starts in $v$ and ends in $w$. We also say that $(v, w)$ is incident to $v$ and $w$. The set of nodes adjacent to a node $v$ are

$$\Gamma^+(v) \overset{\text{def}}{=} \{w \in V \mid (v, w) \in E\}$$

and

$$\Gamma^-(v) \overset{\text{def}}{=} \{u \in V \mid (u, v) \in E\}.$$
The \textit{out-degree} and the \textit{in-degree} of node \( v \) are defined as the corresponding sizes, namely

\[ \gamma^+(v) \overset{\text{def}}{=} |\Gamma^+(v)| \quad \text{and} \quad \gamma^-(w) \overset{\text{def}}{=} |\Gamma^-(w)|, \]

respectively. An \textit{isolated node} is a node \( v \) with

\[ \gamma^+(v) = 0 \quad \text{and} \quad \gamma^-(v) = 0, \]

Let \( G_1 = (V_1, E_1) \) and \( G_2 = (V_2, E_2) \) be two digraphs. Then \( G_1 \) is a subgraph of \( G_2 \) if \( V_1 \subseteq V_2 \) and \( E_1 \subseteq E_2 \), like in the undirected case. The \textit{symmetric hull} of a digraph \( G = (V, E) \) is the undirected graph, in which any pair \((v, w) \in E\) is considered as undirected.

A \textit{(directed) path} from node \( v_0 \) to node \( v_k \) is a sequence of vertices \( v_0, v_1, \ldots, v_k \) such that \((v_i, v_{i+1}) \in E\) for all \( i = 0, 1, \ldots, k - 1 \). The path is \textit{simple} if all its vertices are distinct. The concept of circuit is replaced by a circle, i.e. a path with \( k \geq 2 \) and \( v_0 = v_k \). A graph without cycles is \textit{acyclic}. A vertex \( w \) is \textit{reachable} from a vertex \( v \) if there is a directed path from \( v \) to \( w \) containing at least two vertices. The distance from \( v \) to \( w \) is the number of edges in the shortest simple directed path between them.

The concept of connectedness must be split into two different cases. We distinguish between \textit{strong} and \textit{weak connectedness}. A digraph is strongly connected if any node is reachable from any other node. It is weakly connected if the symmetric hull of the graph is connected. The \textit{weakly} and \textit{strongly} connected components are accordingly defined.

\subsection*{2.2.3 Further Definitions}

The following definitions are valid for both directed and undirected graphs. A \textit{weighted} (di)graph is a graph \( G = (V, E) \) within a weight function \( C : E \rightarrow \mathbb{R} \), which associates a real number to each edge in the graph. The weights can be interpreted as \textit{costs} or \textit{distances}, depending on the applications.

Problems on graphs are usually related to combinatorial questions. We will investigate basically probabilistic problems on graphs. Therefore, we will consider \textit{random graphs}, first introduced by \textsc{Erdős} and \textsc{Rényi} [12].
2.3. Asymptotics

The vertex set of a random graph \( G_{n,p} = (V, E) \) is given by

\[ V = \{1, \ldots, n\}. \]

The edges correspond to independent random experiments where the probability that an edge exists is \( p \in [0,1] \) (in the following we will abbreviate \( q = 1 - p \)). These experiments are executed for the set:

\[ E = \{\{i,j\} \mid i,j \in V, i \neq j\} \]
for an undirected graph and

\[ E = \{(i,j) \mid i,j \in V, i \neq j\} \]
for a directed graph.

This model of random graphs is also called \( G_{n,p} \)-model.

Often, parameters of random graphs analyzed in function of the probability \( p \) show a particular behavior, namely a threshold behavior. Let \( Q \) be a predicate on graphs. Then \( t = t(c,n) \) is called a threshold function for \( Q \) if there is a number \( c_0 \) such that

\[ c > c_0 \Rightarrow \lim_{n \to \infty} \Pr(G_{n,t(c,n)} \text{ satisfies } Q) = 1 \]

and

\[ c < c_0 \Rightarrow \lim_{n \to \infty} \Pr(G_{n,t(c,n)} \text{ satisfies } Q) = 0. \]

2.3 Asymptotics

Many results in average case analysis are computed in term of asymptotic estimate. We will use the following (standard) notation.

Let \( f, g \) be functions from \( \mathbb{N} \) to \( \mathbb{R}_+ \). Then

\[ f = O(g) \iff \exists n_0 > 0, c > 0 : f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0, \]

\[ f = \Omega(g) \iff \exists n_0 > 0, c > 0 : f(n) \geq c \cdot g(n) \text{ for all } n \geq n_0, \]

and

\[ f = \Theta(g) \iff f = O(g) \land f = \Omega(g). \]

Furthermore,

\[ f = o(g) \iff \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0. \]
and

\[ f(n) \approx x \]

is an abbreviation for

\[ \lim_\limits_{n \to \infty} f(n) = x. \]
Part I

Searching in Graphs
Leer - Vide - Empty
Chapter 3

Breadth-First-Search

3.1 Introduction

Traversing a graph is a fundamental problem in graph theory since it is the abstraction of a variety of processes. There are basically two reasons to traverse a graph. On the one hand, we traverse all edges (passing several times through some nodes) to discover properties of the graph like the presence of cycles, strongly connected components, etc. On the other hand, one can only be interested in visiting the nodes (e.g. to determine which nodes can be reached from a given starting point). Then the edges are simply considered as paths to reach the vertices and not all of them need to be considered.

Two possible solutions are widely used to visit all nodes of a graph efficiently. The first one is Breadth-First-Search, the second one Depth-First-Search [46]. Both algorithm work in linear time $O(n+m)$ where $n$ denotes the number of nodes and $m$ the number of edges of the graph. These solutions are very simple to implement and, therefore, used for lots of problems.

Actually, these search strategies are used for both kind of problems, namely to traverse all edges and to visit all nodes. However, in the latter case one can execute a Breadth-First-Search until all nodes are visited and stop before all edges are traversed. Then, the algorithm clearly works in the worst case in linear time $O(n+m)$, but the average case presents some interesting aspects related to the connectedness of a
random graph.

Consider the problem of determining whether a graph is connected or not. Since we do not have any oracle to know this in advance, the natural thing to do is to test it—for instance by executing BFS. Thus, there is an obvious relationship between the algorithmic and the combinatorial aspects. But the relationship between searches and connectedness is much deeper. The distribution of the number of edges traversed during the search is closely related with the probability that the graph is connected. Therefore, an accurate average case analysis of BFS becomes particularly interesting for analyzing the structural properties of a random graph.

But what is the best way to analyze Breadth-First-Search? A possibility is to interpret it as a stochastic process. This approach seems especially meaningful because other problems related with the connectedness of a graph can be analyzed similarly. An example is the number of isolated nodes in a random graph. In the following sections, we will focus the attention on these problems. We will show that both the isolated nodes and the BFS-process can be seen as disintegration processes (with anomalies). Other methods could be used, however the approach proposed illustrates clearly the relationships between the different problems and it allows us to treat them in the same way. This seems more elegant than using three different techniques—even though some of them could be shorter.

In Section 3.2 we consider the disintegration process to introduce the basic techniques. In Section 3.3 we analyze a first example of disintegration process with anomalies, i.e. the number of isolated nodes. Here, the analysis of the asymptotic behavior—and especially of the anomalies—is of particular interest. Finally, in Section 3.4 we study the average performance of Breadth-First-Search and we extend Karp's analysis [24] of the expected size of the connected components in a random graph. This section constitutes the main part of the chapter since it contains most of the new results. In fact, in the previous sections the techniques used are more interesting than the results themselves.

In the next chapter, the analysis will be extended to other exploration strategies. The interpretation as disintegration process will be a central aspect to understand an important class of graph algorithms.
3.2 Discrete Time Disintegration Process

The disintegration of radioactive atoms is one of the simplest stochastic processes that exists, and also one of the most important. The distribution of the number of existing radioactive atoms is given by a binomial distribution. This is true if we assume that the time is both continuous or discrete. The discrete case is particularly interesting because it represents the basic process of a lot of problems in graph theory.

We assume that at time \( t = 0 \) there are \( N \) radioactive atoms which can decay independently with a fixed probability \( p \) at each epoch or can remain radioactive with probability \( q = 1 - p \). Let \( X_t \) denote the number of radioactive atoms at time \( t \). Then, on condition that \( X_{t-1} = \ell + i \), there are \( \ell \) radioactive atoms at time \( t \) if and only if \( i \) atoms decay. The corresponding transition probability is

\[
Pr(X_t = \ell \mid X_{t-1} = \ell + i) = \binom{\ell+i}{i} \cdot p^i \cdot q^{\ell}.
\]

Since

\[
Pr(X_t = \ell) = \sum_{i \geq 0} Pr(X_t = \ell \mid X_{t-1} = \ell + i) \cdot Pr(X_{t-1} = \ell + i)
\]

we get for \( P_{t,\ell} \triangleq Pr(X_t = \ell) \) the recursion

\[
P_{t,\ell} = \sum_{i \geq 0} \binom{\ell+i}{i} \cdot p^i \cdot q^\ell \cdot P_{t-1,\ell+i}
\]

with

\[
P_{0,\ell} = \begin{cases} 1 & \text{for } \ell = N \\ 0 & \text{otherwise} \end{cases}
\]

Although expression (3.2) looks intractable, it implies a much easier recursion for the corresponding generating function.

Definition 3.1 Let \( a_0, a_1, a_2, \ldots \) be a sequence of real numbers. Then

\[
G(z) \triangleq \sum_{i \geq 0} a_i \cdot z^i
\]

is the corresponding generating function. Further, if \( (a_i)_{i \geq 0} \) is a probability distribution, i.e. \( a_i \in [0,1] \) for all \( i = 0,1, \ldots \), and \( \sum_{i \geq 0} a_i = 1 \), then \( G(z) \) is called probability generating function.
For recursion (3.2) we obtain

\[(3.3)\quad g_t(z) \overset{\text{def}}{=} \sum_{t \geq 0} P_{t,t} \cdot z^t\]

\[= \sum_{t \geq 0} \sum_{i \geq 0} \binom{t+i}{i} \cdot p^i \cdot q^{t-i} \cdot P_{t-1,t+i} \cdot z^t\]

\[a \overset{\text{def}}{=} t+i \quad \sum_{a \geq 0} \sum_{i \leq a} \binom{a}{i} \cdot p^i \cdot q^{a-i} \cdot P_{t-1,a} \cdot z^{a-i}\]

\[= \sum_{a \geq 0} P_{t-1,a} \sum_{i \leq a} \binom{a}{i} \cdot p^i \cdot (qz)^{a-i}\]

\[= (p + qz)^{a} = (1 + q(z-1))^{a}\]

or finally

\[(3.4)\quad g_t(z) = g_{t-1}(1 + q(z-1)).\]

Since

\[g_0(z) = z^N = (1 + q^0(z-1))^N,\]

by induction on \(t\) it follows

\[(3.5)\quad g_t(z) \overset{(3.4)}{=} g_{t-1}(1 + q(z-1)) \overset{1.A.}{=} (1 + q^t(z-1))^N\]

which can be expanded to

\[g_t(z) = ((1 - q^t) + q^t z)^N = \sum_{t=0}^{N} \binom{N}{t} q^t (1 - q^t)^{N-t} \cdot z^t.\]

By comparing the coefficients, we get the (expected) binomial distribution

\[(3.6)\quad P_{t,t} = \binom{N}{t} q^t (1 - q^t)^{N-t}.\]

From (3.5) we derive also

\[(3.7)\quad E(X_t) = \sum_{t \geq 0} t \cdot P_{t,t} = \frac{d}{dz} g_t(z) |_{z=1} = Nq^t,\]

and, similarly,

\[(3.8)\quad \text{Var}(X_t) = \frac{d^2}{dz^2} g_t(z) |_{z=1} + E(X_t) - E(X_t)^2 = Nq^t(1 - q^t).\]
3.3 The Distribution of Isolated Vertices

A first example of disintegration process (with anomalies) consists in the probability that in an undirected random graph $G_{n,p}$ there are $\ell$ isolated vertices. We denote

- the set of the isolated nodes in $G_{n,p}$ by $X_n$ and
- the size of $X_n$ by $x_n$

Now, let

$$P_{n,\ell} \overset{\text{def}}{=} \Pr(x_n = \ell),$$

then $P_{n,\ell}$ satisfies the following recursion.

**Lemma 3.2**

(3.9) $P_{n,\ell} = q^{n-1}(P_{n-1,\ell-1} - P_{n-1,\ell}) + \sum_{i \geq 0} \binom{\ell+i}{i} p^i \cdot q^{\ell} \cdot P_{n-1,\ell+i}$

where

$$P_{0,0} = 1 = P_{1,1}$$

and

$$P_{n,\ell} = 0 \quad \text{for } \ell \notin \{0,\ldots,n\}.$$  

**Proof** Equation (3.9) expresses exactly the transitions which may occur from $X_{n-1}$ to $X_n$, i.e. by adding the vertex $n$ to $G_{n-1,p}$. The first possibility is that $n$ is a new isolated vertex. Then none of the $n-1$ possible edges in $1 \leq i \leq n-1$ exists and by definition these are independent. Therefore, the transition probability from $P_{n-1,\ell-1}$ to $P_{n,\ell}$ is given by

$$\Pr(\gamma(n) = 0) = q^{n-1}.$$  

On the other hand, an edge $\{u,n\}$ may exist. Then we get different subcases depending on

(3.10) $|\Gamma(n) \cap X_{n-1}| \overset{\text{def}}{=} i.$

Now, $i = 0$ implies that $n$ is not adjacent to any vertex in $X_{n-1}$, which has probability

$$q^{x_{n-1}} = q^x = q^\ell.$$
on condition that there is an edge \( \{u, n\}, u \in \{1, \ldots, n - 1\} \), with corresponding probability

\[
(1 - q^{n-1-i}) \cdot q^i.
\]

It remains the case \( i \geq 1 \) which guarantees that \( n \) is not isolated. Further, we have

\[
X_n = X_{n-1} - (\Gamma(n) \cap X_{n-1})
\]

and for that reason

\[
\ell = |X_n| = |X_{n-1}| - |\Gamma(n) \cap X_{n-1}| = |X_{n-1}| - i
\]

which implies

\[
x_{n-1} = \ell + i.
\]

Since \( |\Gamma(n)| \) satisfies a binomial distribution we observe the transition probability from \( P_{n-1, \ell+i} \) to \( P_{n, \ell} \)

\[
\Pr(|\Gamma(n) \cap X_{n-1}| = i \mid x_{n-1} = \ell + i) = \left( \binom{\ell+i}{i} \right) \left( 1 - q \right)^{i} \left( q \right)^{\ell+i-i}.
\]

By summing up over all possible transitions we get (3.9).

Let \( g_n(z) \) be the generating function of \( x_n \), namely

\[
g_n(z) \stackrel{\text{def}}{=} \sum_{\ell=0}^{n} P_{n, \ell} \cdot z^\ell.
\]

Then, expression (3.9) leads to the recursion

\[
g_n(z) \stackrel{(3.9)}{=} q^{n-1} \sum_{\ell=0}^{n} \left( P_{n-1, \ell-1} - P_{n-1, \ell} \right) \cdot z^\ell
\]

\[
= (z-1)g_{n-1}(z)
\]

\[
+ \sum_{\ell=0}^{n} \sum_{i \geq 0} \binom{\ell+i}{i} p^i \cdot q^\ell \cdot P_{n-1, \ell+i} \cdot z^\ell
\]

\[
= (3.3) g_{n-1}(1+q(z-1))
\]

or finally

\[
(3.11) g_n(z) = q^{n-1}(z-1) \cdot g_{n-1}(z) + g_{n-1}(1+q(z-1)).
\]
The interpretation of $x_n$ as disintegration process becomes apparent. The isolated vertices correspond to the radioactive atoms and the number of nodes $n$ agrees with the discrete time. At each epoch any radioactive atom may decay with probability $p$ (if the new node is connected to it) or remains radioactive (isolated) with probability $q$. The anomaly appears when the new node added to the graph is isolated. This can be interpreted as the generation of a new radioactive atom.

**Theorem 3.3**

\[ g_n(z) = \sum_{\ell=0}^{n} \binom{n}{\ell} q^{n\ell-\frac{(\ell+1)\ell}{2}} (z-1)^\ell \]  

\[ \Pr(x_n = \ell) = \sum_{j=\ell}^{n} \binom{n}{j} \binom{j}{\ell} (-1)^{j-\ell} q^{nj-\frac{(j+1)j}{2}} \]  

\[ \frac{d}{dz} g_n(z) = nq^{n-1}g_{n-1}(z) \]  

\[ E(x_n) = nq^{n-1} \]  

\[ \text{Var}(x_n) = nq^{n-1}((1-q^{n-1}) + (n-1)pq^{n-2}) \]  

**Proof** First, we show (3.12) by induction on $n$. For $n = 0$ we find

\[ g_0(z) = \binom{0}{0} q^0(z-1)^0 = 1. \]

For $n \geq 1$ first note

\[ q^{n-1}(z-1)g_{n-1}(z) = \sum_{\ell=0}^{n-1} \binom{n-1}{\ell} q^{(n-1)(\ell+1)-\frac{(\ell+1)\ell}{2}} (z-1)^{\ell+1} \]

\[ = \sum_{\ell=1}^{n} \binom{n-1}{\ell-1} q^{(n-1)\ell-\frac{(\ell-1)\ell}{2}} (z-1)^\ell \]

and

\[ g_{n-1}(1+q(z-1)) = \sum_{\ell=0}^{n-1} \binom{n-1}{\ell} q^{(n-1)\ell-\frac{(\ell+1)\ell}{2}} q^{\ell}(z-1)^\ell. \]

Therefore,

\[ g_n(z) = q^{n-1}(z-1)g_{n-1}(z) + g_{n-1}(1+q(z-1)) \]
\[ \sum_{t=0}^{n} \binom{n-1}{t-1} + \binom{n-1}{t} \] 
\[ q^{n-\frac{(t+1)t}{2}} (z - 1)^t \]
which shows (3.12).

From
\[ (z - 1)^t = \sum_{j=0}^{t} \binom{t}{j} z^j (-1)^{t-j} \]
together with (3.12) we infer
\[ g_n(z) = \sum_{t=0}^{n} \binom{n}{t} q^{n-\frac{(t+1)t}{2}} \sum_{j=0}^{t} \binom{t}{j} z^j (-1)^{t-j} \]
\[ = \sum_{t=0}^{n} \sum_{j=t}^{n} \binom{n}{j} \binom{j}{t} (-1)^{j-t} q^{n-\frac{(j+1)j}{2}} z^t \]
which shows (3.13). Next, due to
\[ \binom{n}{t} = \frac{n}{t} \binom{n-1}{t-1} \]
we derive
\[ \frac{d}{dz} g_n(z) = \frac{d}{dz} \sum_{t=0}^{n} \binom{n}{t} q^{n-\frac{(t+1)t}{2}} (z - 1)^t \]
\[ = \sum_{t=1}^{n} \left[ \binom{n-1}{t-1} q^{n-1} q^{n-1} q^{(n-1)(t-1)} - \frac{t(t-1)}{2} \right] \frac{d}{dz} (z - 1)^t \]
\[ = nq^{n-1} \sum_{t=0}^{n-1} \binom{n-1}{t} q^{(n-1)(t-1)} \frac{d}{dz} (z - 1)^t \]
which is equivalent to
\[ \frac{d}{dz} g_n(z) = nq^{n-1} g_{n-1}(z). \]

Because of \( g_n(1) = 1 \) for all \( n \in \mathbb{N} \), Expression (3.15) follows. By iterating (3.14) we finally get (3.16).
3.3. The Distribution of Isolated Vertices

So far, we have investigated the problem exactly. Now, we will approximate \( g_n(z) \). Note that if

\[ g_n(z) \approx g_{n-1}(z), \]

then from (3.14) we infer

\[ \frac{d}{dz}g_n(z) \approx nq^{n-1}g_n(z). \]

The corresponding solution is given by

\[ g_n(z) \approx e^{nq^{n-1}(z-1)} \]

which is the generating function of a Poisson distribution with parameter \( nq^{n-1} \).

**Theorem 3.4** For all \( z \in [0, 1] \) it holds

\[ \left| g_n(z) - e^{nq^{n-1}(z-1)} \right| \leq \min\{q^{-1} \cdot (1 + \frac{p}{q} \cdot (n-1)), \frac{1}{n} + \frac{p}{q} \}. \]

**Proof** At first, we claim that it is sufficient to show

(3.17) \[ |g_n(z) - g_{n-1}(z)| \leq \varepsilon \]

in order to get

(3.18) \[ \left| g_n(z) - e^{nq^{n-1}(z-1)} \right| \leq \varepsilon \]

where \( \varepsilon \) stands for a constant with \( 0 < \varepsilon < 1 \). From

\[ g_n(x) - \varepsilon \leq g_{n-1}(x) \leq g_n(x) + \varepsilon \]

we infer

\[ nq^{n-1}(g_n(x) - \varepsilon) \leq nq^{n-1}g_{n-1}(x) \leq nq^{n-1}(g_n(x) + \varepsilon) \]

and, further, with (3.14)

\[ nq^{n-1}(g_n(x) - \varepsilon) \leq \frac{d}{dx}g_n(x) \leq nq^{n-1}(g_n(x) + \varepsilon) \]

which implies

\[ \frac{d}{dx}g_n(x) \leq nq^{n-1} \leq \frac{d}{dx}g_n(x) \]

\[ \frac{d}{dx}g_n(x) + \varepsilon \leq nq^{n-1} \leq \frac{d}{dx}g_n(x) - \varepsilon. \]
Now, by integrating over \([z, 1]\), we get

\[
\int_{z}^{1} \frac{d}{dx} g_n(x) \, dx \leq \int_{z}^{1} nq^{n-1} \, dx \leq \int_{z}^{1} \frac{d}{dx} g_n(x) \, dx.
\]

This is equivalent to

\[
\ln \left( \frac{1 + \varepsilon}{g_n(z) + \varepsilon} \right) \leq nq^{n-1}(1 - z) \leq \ln \left( \frac{1 - \varepsilon}{g_n(z) - \varepsilon} \right)
\]

or, by exponentiation,

\[
g_n(z) - \varepsilon \leq e^{nq^{n-1}(z-1)} \leq g_n(z) + \varepsilon
\]

which completes the proof of

\[(3.19) \quad |g_n(z) - g_n(z - 1)| \leq \varepsilon \quad \Rightarrow \quad |g_n(z) - e^{nq^{n-1}(z-1)}| \leq \varepsilon.\]

Next, the Taylor approximation of

\[g_n(z + q(z - 1)) = g_n(z + p(1 - z))\]

implies

\[g_n(z + p(1 - z)) = g_n(z) + p(1 - z) \cdot \frac{d}{dz} g_n(z) |_{z = \xi}
\]

for some \(\xi \in [z, z + p(1 - z)]\). By applying this to Equation (3.11) we obtain

\[(3.20) \quad |g_n(z) - g_n(z - 1)| \leq p(n - 1)q^{n-2} \cdot (1 - z) \cdot g_n(z - 1) + q^{n-1} (1 - z) \cdot g_n(z).\]

Now, \(g_k(z) \leq 1\), for \(0 \leq z \leq 1\), induces

\[(3.21) \quad |g_n(z) - g_n(z - 1)| \leq q^{n-1} \left( 1 + \frac{p}{q} (n - 1) \right).\]

On the other hand, we have

\[(1 - z) \cdot \frac{d}{dz} g_n(z) = \sum_{\ell \geq 1} \ell \cdot P_n, \ell \cdot z^{\ell - 1}(1 - z) \leq 1\]

\[\leq \frac{1}{\ell}\]
and for that reason
\[
\frac{n}{n} \cdot q^{n-1} (1 - z) \cdot g_{n-1}(z) \overset{(3.14)}{=} \frac{1}{n} (1 - z) \cdot \frac{d}{dz} g_n(z) \leq \frac{1}{n}
\]
which together with \( 1 - \xi \leq 1 - z \leq (1 - \xi)/q \) implies
\[
(3.22) \qquad |g_n(z) - g_{n-1}(z)| \leq \frac{p}{q} + \frac{1}{n}.
\]
The inequalities (3.21) and (3.22) complete the proof.

From Theorem 3.4 we infer that the number of isolated vertices \( x_n \) is asymptotically Poisson distributed (see [14, Chap. XI.6]) with parameter \( \lim_{n \to \infty} n q^{n-1} \), if it exists. If this is not the case, then the distribution degenerates. For example, if \( n q^{n-1} \to 0 \), then \( p_{n,t} \approx 1 \) for \( t = 0 \) and 0 otherwise. For further discussions see for instance [39].

### 3.4 Exploring a Random Graph

The exploration of an undirected random graph \( G_{n,p} \) in a Breadth-First-Search (BFS) manner is a further example of disintegration process with anomalies. Actually, this interpretation is particularly useful since it gives the most intuitive explanation of the nature of this process.

The algorithm in Figure 3.1 starts in a given vertex \( s \) and maintains the set \( R \) of visited vertices and a First-In-First-Out queue \( Q \) of visited but unexplored vertices, initially \( R = Q = \{s\} \). In every step we remove one vertex \( v \) from \( Q \) and add all the neighbors of \( v \) which are not in \( R \) to \( Q \) and \( R \). If \( Q \) becomes empty, the process stops.

We are interested in the number of unvisited nodes
\[
u_t \overset{\text{def}}{=} |V - R|
\]
after \( t \) iterations of the while-loop. When the algorithm has finished, the set \( V - R \) contains exactly the vertices which are not in the connected component \( C(s) \) of the start vertex \( s \). The unvisited nodes can be interpreted as radioactive atoms which decay at epoch \( t \) if they are connected to the vertex \( v \) by an edge. The anomaly of this disintegration appears if the queue is empty. In this case the process stops and the number of unreached nodes will be considered as undefined in the future.
BFS \((G = (V, E), s \in V)\):

1. \(Q \leftarrow \{s\}\);
2. \(R \leftarrow \{s\}\);
3. \text{while } Q \neq \{\} \text{ do}
4. \text{remove the first element } v \text{ of } Q;
5. \(A \leftarrow \{u \in V - R \mid vu \in E\}\);
6. \(Q \leftarrow Q \cup A\);
7. \(R \leftarrow R \cup A\);
8. \text{od.}

Figure 3.1: Exploration algorithm

epochs. Since \(n - (t - 1) - u_{t-1}\) corresponds to the number of nodes in the queue at time \(t - 1\), we have

\[
\Pr(u_t = \ell \mid u_{t-1} = n - (t - 1)) = 0 \text{ for any } \ell \in \mathbb{N},
\]

(because the algorithm has finished at time \(t - 1\) and \(u_t\) is undefined). On the other hand, we obtain for defined \(u_t\)

\[
(3.23) \quad \Pr(u_t = \ell \wedge |C(s)| \geq t) = P_{t, \ell}^{(n)} = \sum_{i \geq 0} \binom{t+i}{i} p^i q^\ell \cdot P_{t-1, t+i}^{(n)}
\]

where \(P_{0, n-1}^{(n)} = 1\) and \(P_{t, \ell}^{(n)} = 0\) for \(\ell < 0, t < 0,\) and \(t + \ell > n\). The generating function satisfies

\[
g_t^{(n)}(z) = \sum_{\ell \geq 0} P_{t, \ell}^{(n)} \cdot z^\ell
\]

\[
= \sum_{\ell \geq 0} \sum_{i \geq 0} \binom{t+i}{i} p^i q^\ell \cdot P_{t-1, t+i}^{(n)} \cdot z^\ell
\]

\[
= \sum_{\ell \geq 0} \sum_{i \geq 0} \binom{t+i}{i} p^i q^\ell \cdot P_{t-1, t+i}^{(n)} \cdot z^\ell
\]

\[
= g_{t-1}^{(n)}(1 + q(z - 1))
\]
\[ -\sum_{t\geq 0} \left( \begin{array}{c} n-(t-1) \\ t \end{array} \right) p^{n-(t-1)-t} (qz)^t \cdot P_{t-1,n-(t-1)}^{(n)} \] 

that is

\[ (3.24) g_t^{(n)}(z) = g_{t-1}^{(n)}(1 + q(z - 1)) - (1 + q(z - 1))^{n-(t-1)} P_{t-1,n-(t-1)}^{(n)} \]

where \( g_0^{(n)}(z) = z^{n-1} \). By (3.23), \( g_t^{(n)}(z) \) is not a probability generating function, in fact

\[ g_t^{(n)}(1) = \Pr(|C(s)| \geq t) \]

and, furthermore,

\[ g_t^{(n)}(0) = P_{t,0}^{(n)} \]

is the probability that after \( t \) iterations all nodes are visited. Clearly, for \( t = n \) we obtain

\[ (3.25) g_n^{(n)}(z) = \Pr(|C(s)| = n) = \varphi_n \]

where \( \varphi_n \) denotes the probability that a graph with \( n \) nodes is connected. With \( \varphi_n \) we are able to solve (3.24). First recall that the process stops at time \( k \) if \( u_k = n - k \). For that reason, we have

\[ (3.26) P_{k,n-k}^{(n)} = \binom{n-1}{k-1} q^{(n-k)k} \cdot \varphi_k \]

since \( k - 1 \) of the vertices in \( C(s) \) are chosen at random and none of them is joint by an edge to the rest of the graph. Then, by induction on \( t \) we infer from (3.24)

\[ (3.27) g_t^{(n)}(z) = g_0^{(n)}(1 + q^t(z - 1)) \]

\[ - \sum_{k=1}^{t-1} (1 + q^k(z - 1))^{n-(t-k)} \cdot P_{t-k,n-(t-k)}^{(n)} \]

\[ = (1 + q^t(z - 1))^{n-1} \]

\[ - \sum_{k=1}^{t-1} (1 + q^{t-k}(z - 1))^{n-k} \cdot \binom{n-1}{k-1} q^{(n-k)k} \cdot \varphi_k \]

which together with (3.25) implies

\[ (3.28) \varphi_n = g_n^{(n)}(1) = 1 - \sum_{k=1}^{n-1} \binom{n-1}{k-1} q^{(n-k)k} \cdot \varphi_k \]
and

\[ \varphi_n = g^{(n)}_n(0) = (1 - q^n)^{n-1} - \sum_{k=1}^{n-1} \binom{n-1}{k-1} (q^k - q^n)^{n-k} \varphi_k. \]

The identity (3.28) was first noted by Gilbert [19]. He also found

\[ \varphi_n = q^{\binom{n(n-1)}{2}} \sum_{r_1, \ldots, r_n} \frac{n! \cdot (-1)^{m-1}(m-1)!}{r_1! \cdots r_n!} \prod_{i=1}^{n} \left( \frac{q^{(i-1)}}{i!} \right)^{r_i}, \]

where the sum is extended over all \( r_i \geq 0 \) with \( r_1 + 2 \cdot r_2 + \cdots + n \cdot r_n = n \), namely over the partitions of \( n \), and \( m = r_1 + r_2 + \cdots + r_n \). Due to Mallows and Riordan [29] we can represent \( \varphi_n \) in terms of the multivariate Bell polynomials, which are defined as

\[ Y_n(x_1, \ldots, x_n) = \sum_k \frac{n!}{k_1! \cdot k_2! \cdots k_n!} \prod_{i=1}^{n} \left( \frac{x_i}{i!} \right)^{k_i}. \]

The sum is over all partitions \( k \) of \( n \), namely \( k_1 + 2 \cdot k_2 + \cdots + n \cdot k_n = n \), see [4] for more details. Then, using umbral notation (like in [29]) we define

\[ f^m \equiv f_m = (-1)^{m-1}(m-1)! . \]

It follows immediately that

\[ \varphi_n = q^{n(n-1)/2} \cdot Y_n \left( f \cdot \frac{1}{q^0}, \ldots, f \cdot \frac{1}{q^{n(n-1)/2}} \right). \]

Again, we are interested in the asymptotic behavior of the process. From (3.27) we infer

\[ (1 + q^t(z - 1))^{n-1} - \sum_{k=1}^{t-1} \binom{n-1}{k-1} q^{(n-k)k} \cdot \varphi_k \leq g^{(n)}_t(z) \]

and

\[ g^{(n)}_t(z) \leq (1 + q^t(z - 1))^{n-1} . \]

This means that the BFS-process behaves like a disintegration process if

\[ \Pr(|C(s)| < t) = 1 - g^{(n)}_t(1) \approx 0 , \]
3.4. Exploring a Random Graph

namely, if the probability that the anomaly has occurred is nearly 0. Our concern is to determine under which conditions this is true. First, due to Stirling's formula we have

\[
\binom{n-1}{k-1} \leq \frac{(n-1)^{k-1}}{(k-1)!} \leq \frac{(n-1)^{k-1}}{(k-1)^{k-1}} \cdot e^{k-1}
\]

which together with

\[
\varphi_k = g_k^{(k)}(0) \leq (1 - q^k)^k \leq (k \cdot p)^k
\]

leads to

\[
(3.34) 1 - g_t^{(n)}(1) = \sum_{k=1}^{t-1} (n-1)^{k-1} k \cdot \varphi_k
\]

\[
\leq q^{n-1} + q^{n-1} \sum_{k=2}^{t-1} \left( \frac{n-1}{k-1} \cdot e \cdot q^{n-1-k} k \cdot p \right)^{k-1}
\]

Next, for \( k \geq 2 \) we get

\[
(k/(k-1))^{k-1} \leq e
\]

and further

\[
k \leq \left( 1 - \frac{1 + \varepsilon + \ln(p \cdot n)}{p \cdot n} \right) \cdot n \ \Rightarrow (n-1) e q^{n-1-k} p \leq 1 - \varepsilon
\]

for any \( \varepsilon \in (0, 1) \). Therefore, we obtain

\[
1 - g_t^{(n)}(1) \leq q^{n-1} e \sum_{k=1}^{t-1} \left( (n-1) e q^{n-1-k} p \right)^{k-1} \leq q^{n-1} \frac{e}{\varepsilon}
\]

for

\[
t \leq \left( 1 - \frac{1 + \varepsilon + \ln(p \cdot n)}{p \cdot n} \right) \cdot n.
\]

On the other hand,

\[
q^{n-1} \leq \binom{n-1}{1-1} q^{(n-1) \cdot 1} \varphi_1 \leq 1 - g_t^{(n)}(1).
\]

We summarize these results in the following theorem.
Theorem 3.5 If

\begin{equation}
(3.35) \quad t \leq \left(1 - \frac{1 + \varepsilon + \ln(p \cdot n)}{p \cdot n}\right) \cdot n,
\end{equation}

then

\begin{equation}
(3.36) \quad q^{n-1} \leq 1 - g_t^{(n)}(1) \leq \frac{e}{\varepsilon} \cdot q^{n-1}.
\end{equation}

Remark Theorem 3.5 agrees with the known threshold behavior of connectivity in random graphs [3, 36]. Let $\alpha < 1 < \beta$. Then, for $p \leq \alpha/n$, the graph breaks up into many small components. For $\beta/n \leq p \leq \alpha \cdot \log n/n$ a giant component dominates the behavior of the exploration. In this case, the approximation of BFS by the disintegration process is only reasonable for a moderate $t$. Finally, for $p \geq \beta \cdot \log n/n$, the graph is almost surely connected.

Karp [24] exploited a similar idea to estimate the size of the connected components. Basically, he approximated the distribution of $u_t$ by a binomial distribution. However, his analysis is only meaningful for the case that $p \cdot n$ is constant. The following corollary extends its results for arbitrary $p$.

Corollary 3.6 Let $C(s)$ denote the connected component containing node $s$. Then

\[ n - \mathbb{E}(|C(s)|) = O\left(n \cdot e^{-p n} \cdot p n\right). \]

Proof We consider the number of unreached nodes $u_t$ at time

\begin{equation}
(3.37) \quad t = \left(1 - \frac{1 + \varepsilon + \ln(p \cdot n)}{p \cdot n}\right) \cdot n.
\end{equation}

In the following, we will abbreviate $c(s) = |C(s)|$. Then, we have

\[
\mathbb{E}(n - c(s) \mid c(s) \geq t) \cdot \Pr(c(s) \geq t) \leq \\
\leq \mathbb{E}(u_t \mid c(s) \geq t) \cdot \Pr(c(s) \geq t) \\
\leq \left. \frac{d}{dz} g_t^{(n)}(z) \right|_{z=1} \\
\leq (n - 1) \cdot q^t
\]

and, due to (3.35) and (3.36),

\[
\mathbb{E}(n - c(s) \mid c(s) \leq t) \cdot \Pr(c(s) \leq t) \leq (n - 1) \cdot O(q^{n-1}).
\]
3.5. Conclusion

Hence,

\[ n - \mathbb{E}(c(s)) \leq (n - 1) \cdot q^t + O((n - 1) \cdot q^{n-1}) = O(n \cdot q^t). \]

Finally, from (3.37) we infer

\[ n - \mathbb{E}(c(s)) = O(n \cdot q^t) = O(n \cdot e^{-p^t}) \overset{\text{3.37}}{=} O(n \cdot e^{-p^t} \cdot p n) \]

which completes the proof.

For \( p = \beta \cdot \log n / n \) we obtain

\[ n - \mathbb{E}(|C(s)|) \leq O(n^{1-\beta} \cdot \beta \ln(n)) \approx 0. \tag{3.38} \]

In words, the number of nodes not reachable from the starting point \( s \) tends to 0. Hence, the probability that the graph is not connected approaches 0, too.

3.5 Conclusion

The interpretation of the number of isolated nodes and of BFS as disintegration processes explains intuitively two curious characteristics of random graphs. On the one hand, it becomes apparent why the probability that an isolated node exists and the probability that the graph is connected have the same threshold behavior. The answer is that both processes are disintegrations with anomalies. On the other hand, for the analysis of BFS we did not use explicitly the fact that the graph is undirected. Then, the connectedness of directed random graphs can be analyzed in the same way which explains why also the probability that a directed graph is strongly connected has the same threshold. These aspects will be reconsidered and generalized in the next chapter.
Chapter 4

Graph Exploration

4.1 Introduction

The relationship between the behavior of Breadth-First-Search and the connectedness of a graph observed in the last chapter is quite intriguing. In particular, it induces to ask whether it is restricted to BFS or it is a characteristic common to other exploration strategies. To answer, we first have to give a general definition of exploration.

Consider a (directed) graph $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges. Furthermore, at the beginning a given subset $S \subseteq V$ of nodes are marked (in general, we will assume $|S| = 1$). The goal is to mark the maximal number of nodes in $V - S$ by using only the following operations:

1. We can select a node (according to an arbitrary strategy).
2. Once a node $v$ is selected, we can traverse an edge $e$ incident to $v$.
3. When an edge $e = (v, w)$ is traversed, we can mark the end-node $w$.

Note that a node $w \notin S$ cannot be marked if there exists no ingoing edge $e = (v, w) \in E$ where $v \in V$.

Remark When an edge is incident to a node, it means implicitly that it is outgoing from the node. If this is not the case, it will be mentioned
This definition is minimal. In practice, the operations are subject to further restrictions depending on the selection strategy. For instance, assume that the selected node must always be the same until all its incident edges are traversed. When no further untraversed incident edge exists, then a new marked node is selected. In this case, the exploration corresponds to Breadth-First-Search which suggests that the relationship with the connectedness can be extended to other traverse strategies.

But this definition of exploration has even more consequences. Other important algorithms can be described as node explorations. An example is Dijkstra's algorithm [11] to determine the minimum spanning tree (MST) or the single source shortest paths (SSSP) in a weighted graph (where the weights are non-negative). For these problems, upper bounds for the running time are commonly studied, but lower bounds are usually not. Mehlhorn and Priebe [33] analyzed the shortest paths problem in a complete weighted graph. They investigated the number of edges traversed when the weight function $C$ is provided in the form of an oracle that answers questions of the following kind:

- What is the weight $C(e)$ of a given edge $e$?
- Given a vertex $v \in V$ and an integer $k \in \{1, \ldots, n\}$, what is the end-point of the edge with source $v$ and with the $k$-th smaller weight?

They found that the expected number of edges traversed to solve the problem is $\Omega(n \log(n))$. This result holds for a class of weights (the so-called simple weights) in the end-point independent model (see Definition 4.1).

Although the bound of Mehlhorn and Priebe is valid for a particular class of graphs, it seems to be much more general. The interpretation as node exploration suggests the same. In fact, the equivalence between BFS and the connectedness implies that at least $\Omega(n \cdot \log(n))$ edges must be traversed to reach all nodes in a dense random graph (roughly, in a graph with more than $n \log(n)$ edges). If this result is valid also for any other search strategy, then the lower bound of Mehlhorn and Priebe for the SSSP problem can be extended to a larger class of graphs. In the following sections we will see that this is indeed the case: The average
number of edges traversed during a node exploration in dense graphs is \( \Omega(n \log(n)) \).

The situation changes significantly by extending the available operations. KARP [24] proposed an algorithm to determine the connected components in expected time \( O(n) \) (if the graph is not too sparse), and SCHNORR [41] found an algorithm to determine the transitive closure of an algorithm in linear expected time \( O(n + m^*) \), where \( m^* \) is the expected size of the output. To achieve these results, it is necessary to traverse the edges in the opposite direction, namely we determine the next edge to traverse by selecting the end-node \( w \) and by choosing an ingoing edge \( e = (v, w) \), as we will see in Section 4.7.

## 4.2 The Model

To analyze the exploration, we have to define the probabilistic model used. We consider the standard \( G_{n,p} \)-model of random graphs. We assume that the traverse operations are random in the following sense: When the node \( v \) is selected, the traversing edge is chosen with equal probability among the ones incident to node \( v \) which have not yet been traversed. For weighted graphs, we consider the end-point independent model.

**Definition 4.1** For every node \( v \) with out-degree \( k \), an arbitrary set of non-negative weights \( c_{v,1}, c_{v,2}, \ldots, c_{v,k} \) is fixed. These weights are assigned randomly to the \( k \) edges incident to \( v \). More precisely, let 
\[
e_1 = (v,v_1), e_2 = (v,v_2), \ldots, e_k = (v,v_k)
\]
be these edges, and let \( C(e) \) denote the weight of edge \( e \). Then
\[
\Pr(V_i : C(e_i) = c_i) = \Pr(V_i : C(e_i) = c_{\pi(i)})
\]
for any permutation \( \pi \) of \( \{1,2,\ldots,k\} \).

We will focus our attention on dense random graphs, namely

\[
p \geq (1 + \varepsilon) \cdot \frac{\log(n)}{n} \quad \text{and} \quad p = o(1)
\]

where \( \varepsilon > 0 \). The first condition guarantees that the graph is almost surely (strongly) connected. If this is not the case, the whole graph is traversed, and the running time is \( O(m + n) \). We do not handle the case \( p \neq o(1) \) because the search could be reduced to the exploration of the edges incident to a single node. This implies a running time of \( O(n) \).
4.3 Node Exploration

In this section we show that any node exploration requires to traverse $\Theta(n \cdot \log n)$ edges on average. The upper bound is quite simple to show (see Lemma 4.7). The proof of the lower bound is structured as follows. First, we show that the distribution of the number of unmarked nodes does not depend on the sequence in which the edges are considered. More formally, we label the nodes by the first time they have been selected. Let $\alpha_i$ denote the random variable corresponding to the number of traversed edges incident to the node with label $i$. Then the distribution of the number of unmarked nodes depends only on $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$. This means that for any fixed value of $\alpha$, say $\alpha = (a_1, a_2, \ldots, a_n)$, we can assume that we traverse all $a_i$ edges when the node with label $i$ is considered for the first time. This enable us to investigate the first two moments of the number of unmarked nodes when the $i$-th node is selected as a function of $\alpha$. Next, due to the second moment method, we find that if almost all nodes are marked, then the number of edges traversed is $\Omega(n \cdot \log(n))$ with high probability. Therefore, the number of edges traversed during the whole exploration, namely

$$M_n(\alpha) \overset{\text{def}}{=} \alpha_1 + \alpha_2 + \cdots + \alpha_n,$$

satisfies

$$M_n(\alpha) = \Omega(n \log n)$$

with high probability for any possible value of $\alpha = (a_1, \ldots, a_n)$. Note that the probability that $\alpha = \alpha$ is determined by the structure of the graph (i.e. the number of existing edges) and by the traverse strategy. Since the lower bound is independent of $\alpha$, we obtain

$$\mathbb{E}(M_n) = \Omega(n \log n)$$

which is also independent of the search strategy.

Note that by traversing an edge more than once we do not gain new useful information. Therefore, we will consider only strategies which traverse the edges at most once. This can be guaranteed by marking the traversed edges. Then, marked edges can be ignored for the rest of the exploration. Of course, we can also ignore the nodes with no incident unmarked edges.

Recall that when an (unmarked) edge $e$ is traversed, the starting node is known but not the end-node. Hence, we do not know a priori if
it is already marked or not. To start with, we determine the probability that the edge $e$ leads to an unmarked node.

**Lemma 4.2** Consider the exploration at an arbitrary point and let $U$ be the set of the currently unmarked nodes. Further, let $T(u)$, for $u \in V$, be the set of edges incident to $u$ which have been traversed in previous steps. If we select node $v$ and we choose an edge $e$ incident to $v$ but not in $T(v)$ (assuming that such an edge exists), then the probability that the corresponding end-point $w$ is unmarked is

$$\Pr(w \in U \mid |U| = \ell, |T(v)| = k, \gamma^+(v) > k) = \frac{\ell}{n - 1 - k}.$$  

**Proof** Since the edge $e$ is chosen (at random) among the edges incident to $v$ not traversed in previous steps, the end-point of $e$ is a node in $\Gamma(v) - N(v)$, where

$$N(v) \overset{\text{def}}{=} \{u \in V \mid (v, u) \in T(v)\}.$$  

Because we consider graphs in the $G_{n,p}$-model, we have

$$\forall u \in V - \{v\} - N(v) : \Pr(u \in \Gamma(v)) = p.$$  

That is, on condition that $\Gamma(v) - N(v) \neq \emptyset$, the end-point of $e$ can be any node in $V - \{v\} - N(v)$ with equal probability. The number of unmarked nodes among them is

$$|U \cap (V - N(v))| = |U \cap V| = \ell$$  

since $N(v) \cap U = \emptyset$. Therefore, the probability that the end-point of $e$ is unmarked is

$$\Pr(w \in U \mid |U| = \ell, |T(v)| = k, \gamma^+(v) > k) = \frac{\ell}{n - 1 - k}$$  

(which does not depend on $p$).

Roughly speaking, Lemma 4.2 implies that the end-node of a chosen edge $e$ can be interpreted as a random variable uniformly distributed on $V - (\{v\} \cup N(v))$. We will use this observation in the next lemma.

**Lemma 4.3** The distribution of the number of unmarked nodes does not depend on the sequence in which the edges are considered.
**Proof** Assume we have traversed the edges $e_1, e_2, \ldots, e_m$ and let $U_m$ be the set of unmarked nodes. Suppose $|U_m| = \ell$. Next, we continue the exploration by traversing the edges $e_{m+1}$ and $e_{m+2}$ which are incident to two nodes $v_1$ and $v_2$, respectively. Without loss of generality, $v_1$ and $v_2$ are distinct. Assume that $k_1$ ($k_2$) traversed edges are incident to node $v_1$ ($v_2$).

Then after $e_{m+1}$ and $e_{m+2}$ have been traversed, the number of unmarked nodes is $\ell - \delta$, where $\delta \in \{0, 1, 2\}$. The distribution of $\delta$ does not depend on the sequence in which we consider the two edges. In fact, if we consider first $e_{m+1}$ and then $e_{m+2}$, we obtain

$$\Pr(\delta = 0) = \frac{\ell}{n - 1 - k_1} \cdot (1 - \frac{\ell}{n - 1 - k_2}).$$

Next, we investigate the case $\delta = 1$. Let $w_1$ and $w_2$ denote the endpoint of $e_{m+1}$ and $e_{m+2}$, respectively. Further, let $U_{m+1}$ be the set of unmarked nodes after $e_{m+1}$ is traversed. Then, $\delta = 1$ if

- $w_1 \in U_m$ (which implies $U_{m+1} = U_m + \{w_1\}$), and $w_2 \notin U_{m+1}$ or
- $w_1 \notin U_m$ (thus $U_{m+1} = U_m$) and $w_2 \in U_m$.

From Lemma 4.2 we infer

$$\Pr(w_1 \in U_m) = \frac{\ell}{n - 1 - k_1}$$

and

$$\Pr(w_2 \notin U_{m+1} \mid w_1 \in U_m) = 1 - \frac{\ell - 1}{n - 1 - k_2}.$$ 

Hence,

$$\Pr(w_1 \in U_m \land w_2 \notin U_{m+1}) = \frac{\ell}{n - 1 - k_1} \cdot (1 - \frac{\ell - 1}{n - 1 - k_2}).$$

Similarly,

$$\Pr(w_1 \notin U_m) = 1 - \frac{\ell}{n - 1 - k_1}$$

and

$$\Pr(w_2 \in U_{m+1} \mid w_1 \notin U_m) = \frac{\ell}{n - 1 - k_2}.$$
That is,
\[
\Pr(w_1 \notin U_m \land w_2 \in U_{m+1}) = (1 - \frac{\ell}{n - 1 - k_1}) \cdot \frac{\ell}{n - 1 - k_2}.
\]
Therefore, we infer
\[
\Pr(\delta = 1) = \frac{\ell}{n - 1 - k_1} \cdot \left( 1 - \frac{\ell - 1}{n - 1 - k_2} \right) + \left( 1 - \frac{\ell}{n - 1 - k_1} \right) \cdot \frac{\ell}{n - 1 - k_2}.
\]
Finally,
\[
\Pr(\delta = 2) \overset{4.2}{=} \frac{\ell \cdot (\ell - 1)}{(n - 1 - k_1) \cdot (n - 1 - k_2)}.
\]
The distribution of \( \delta \) is exactly the same if we traverse first \( e_{m+2} \) and then \( e_{m+1} \).

This argument is valid for any \( m \geq 0 \). This means that any two edges \( e_i \) and \( e_{i+1} \) can be inverted in the sequence without modifying the final distribution of the number of unmarked nodes. We can repeat this inversion-step several times. Hence, we permute the sequence in which we traverse the edges. Thus, any traversing sequence produces the same final distribution of the number of unmarked nodes.

Due to Lemma 4.3, we may assume without loss of generality that we traverse all \( \alpha_i \) edges exactly when node \( i \) is considered for the first time. Then the labels associated to the nodes can be interpreted as time. Thus, we denote by \( u_t \) the number of unmarked nodes at time \( t \), namely after we have considered \( \alpha_1 + \alpha_2 + \cdots + \alpha_t \) edges.

Now, assume the vector \( \alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n) \) is fixed. Then \( u_t \) can be analyzed as a function of \( \alpha \).

**Lemma 4.4** For any value \( \alpha = (a_1, a_2, \ldots, a_n) \)

\[
(4.2) \quad \mathbb{E}(u_t \mid \alpha = \alpha) = (n - 1) \prod_{i=1}^{t} \left( 1 - \frac{a_i}{n - 1} \right),
\]

\[
(4.3) \quad \text{Var}(u_t \mid \alpha = \alpha) \leq \mathbb{E}(u_t \mid \alpha = \alpha).
\]

**Proof** First, assume \( j \) nodes are unmarked at time \( t - 1 \) and we explore \( \alpha_t \) edges at time \( t \). The corresponding conditional distribution of \( u_t \) is
given by

\begin{equation}
\Pr(u_t = i \mid u_{t-1} = j, \alpha = a) = \frac{\binom{j}{i} \cdot \binom{n-1-j}{a_t-(j-i)}}{\binom{n-1}{a_t}}.
\end{equation}

In fact, assume that \(j - i\) of the originally \(j\) unmarked nodes are marked at time \(t\). This means that \(j - i\) of the \(a_t\) traversed edges lead to these nodes. The other \(a_t - (j - i)\) lead therefore to some of the \(n-1-j\) nodes already marked at time \(t-1\). The possible combinations to choose the \(a_t\) edges are \(\binom{j}{j-i} \cdot \binom{n-1-j}{a_t-(j-i)}\). The total number of possibilities is \(\binom{n-1}{a_t}\) which shows (4.4). Next, note

\[
\sum_{i \geq 0} i \Pr(u_t = i \mid u_{t-1} = j, \alpha = a) = \sum_{i \geq 0} \frac{i \cdot \binom{j}{i} \cdot \binom{n-1-j}{a_t-(j-i)}}{\binom{n-1}{a_t}} = \sum_{j \geq 1} j \frac{(j-1)}{(i-1)} \frac{(n-1)-(j-1)}{a_t-(j-i)} \frac{(n-2)}{a_t}.
\]

Since

\[
\sum_{i \geq 1} \frac{(j-1)}{(i-1)} \frac{(n-1)-(j-1)}{a_t-(j-i)} \frac{(n-2)}{a_t} = 1,
\]

we get

\begin{equation}
\sum_{i \geq 0} i \cdot \Pr(u_t = i \mid u_{t-1} = j, \alpha = a) = j \cdot \left(1 - \frac{a_t}{n-1}\right).
\end{equation}

By definition,

\[
E(u_t \mid \alpha = a) = \sum_{i \geq 0} i \cdot \Pr(u_t = i \mid \alpha = a)
\]

and

\[
\Pr(u_t = i \mid \alpha = a) = \\
= \sum_{j \geq 0} \Pr(u_t = i \mid u_{t-1} = j, \alpha = a) \cdot \Pr(u_{t-1} = j \mid \alpha = a).
\]
Due to (4.5) we infer

\[
\sum_{i,j \geq 0} i \cdot \Pr(u_t = i \mid u_{t-1} = j, \alpha = a) \cdot \Pr(u_{t-1} = j \mid \alpha = a) = \\
\sum_{j \geq 0} j \cdot \left(1 - \frac{a_t}{n-1}\right) \cdot \Pr(u_{t-1} = j \mid \alpha = a)
\]

and finally

\[
E(u_t \mid \alpha = a) = \left(1 - \frac{a_t}{n-1}\right) \cdot E(u_{t-1} \mid \alpha = a).
\]

Since

\[
E(u_0 \mid \alpha = a) = n - 1,
\]

by induction we get (4.2).

Similarly to (4.5) we have

\[
\sum_{i \geq 0} i (i - 1) \Pr(u_t = i \mid u_{t-1} = j, \alpha = a) = \\
\left(1 - \frac{a_t}{n-1}\right) \left(1 - \frac{a_t}{n-2}\right) \sum_{j \geq 0} j (j - 1) \cdot \Pr(u_{t-1} = j \mid \alpha = a).
\]

Hence,

\[
\sum_{i \geq 0} i (i - 1) \cdot \Pr(u_t = i \mid \alpha = a) = \\
\left(1 - \frac{a_t}{n-1}\right) \left(1 - \frac{a_t}{n-2}\right) E(u_{t-1}^2 \mid \alpha = a) - E(u_{t-1} \mid \alpha = a)
\]

By definition

\[
E(u_t^2 \mid \alpha = a) = \sum_{i \geq 0} i \cdot (i - 1) \cdot \Pr(u_t = i \mid \alpha = a) + \\
\sum_{i \geq 0} i \cdot \Pr(u_t = i \mid \alpha = a).
\]

\[
= \left(1 - \frac{a_t}{n-1}\right) E(u_{t-1} \mid \alpha = a)
\]
Thus, we obtain
\[
E(u_t^2 | \alpha = a) = \left(1 - \frac{a_t}{n-1}\right) \left(1 - \frac{a_t}{n-2}\right) \times \\
\times \left(E(u_{t-1}^2 | \alpha = a) - E(u_{t-1} | \alpha = a)\right) \\
+ \left(1 - \frac{a_t}{n-1}\right) \cdot E(u_{t-1} | \alpha = a) \\
= \left(1 - \frac{a_t}{n-1}\right) \left(1 - \frac{a_t}{n-2}\right) \cdot E(u_{t-1}^2 | \alpha = a) \\
+ \left(1 - \frac{a_t}{n-1}\right) \frac{a_t}{n-2} \cdot E(u_{t-1} | \alpha = a).
\]

By induction we will show
\[
E(u_t^2 | \alpha = a) = E(u_t | \alpha = a) \cdot \left(1 + (n-2) \prod_{i=1}^{t} \left(1 - \frac{a_i}{n-2}\right)\right).
\]

First, for \( t = 0 \) we have
\[
E(u_0^2 | \alpha = a) = (n-1)^2.
\]

Next, assume it holds for \( t - 1 \). Then
\[
E(u_t^2 | \alpha = a) = \left(1 - \frac{a_t}{n-1}\right) \left(1 - \frac{a_t}{n-2}\right) E(u_{t-1} | \alpha = a) \times \\
\times \left(1 + (n-2) \prod_{i=1}^{t-1} \left(1 - \frac{a_i}{n-2}\right)\right) \\
+ \left(1 - \frac{a_t}{n-1}\right) \frac{a_t}{n-2} E(u_{t-1} | \alpha = a) \\
\overset{(4.6)}{=} E(u_t | \alpha = a) \left(1 - \frac{a_t}{n-2}\right) + \frac{a_t}{n-2} \\
+ E(u_t | \alpha = a) \cdot (n-2) \prod_{i=1}^{t} \left(1 - \frac{a_i}{n-2}\right) \\
= E(u_t | \alpha = a) \cdot \left(1 + (n-2) \prod_{i=1}^{t} \left(1 - \frac{a_i}{n-2}\right)\right).
\]

Since \( 1 - a_i/(n-2) \leq 1 - a_i/(n-1) \), we infer
\[
E(u_t^2 | \alpha = a) \leq E(u_t | \alpha = a) \left(1 + (n-1) \prod_{i=1}^{t} \left(1 - \frac{a_i}{n-1}\right)\right) \\
\overset{(4.2)}{=} E(u_t | \alpha = a) + E(u_t | \alpha = a)^2.
\]
Therefore,
\[ \text{Var}(u_t | \alpha = a) \leq \text{E}(u_t | \alpha = a). \]

In Lemma 4.4 we derived the first two moments of the number of unmarked nodes for fixed values of \( \alpha_i \). Now we are interested in the number of edges traversed until all nodes are marked. We will show that if the traversed edges are \( o(n \log(n)) \), then there exist unmarked nodes with high probability. Hence, if all nodes are marked, the number of edges traversed is \( \Omega(n \log(n)) \) with high probability.

**Lemma 4.5** Let \( a = (a_1, \ldots, a_n) \), where \( a_i = o(n) \) for every \( i \geq 1 \). Then
\[ M_n(a) = o(n \log n) \Rightarrow \text{Pr}(u_n = 0) = O(n^{-1+o(1)}). \]

**Proof** Consider
\[ M_n(a) = (n-1) \sum_{i=1}^{n} \frac{a_i}{n-1}. \]

By assumption, \( a_i = o(n) \). Thus
\[ M_n(a) = (n-1) \sum_{i=1}^{n} - \log \left(1 - \frac{a_i}{n-1}\right) (1 + o(1)) \]
because \( -\log(1 - x) = x + o(x) \). Due to (4.2),
\[ \sum_{i=1}^{n} - \log \left(1 - \frac{a_i}{n-1}\right) = - \log \prod_{i=1}^{n} \left(1 - \frac{a_i}{n-1}\right) = \log(n-1) - \log \text{E}(u_n | \alpha = a). \]

Assume \( M_n(a) = o((n-1) \log(n-1)) \). Then
\[ \log(n-1) - \log \text{E}(u_n | \alpha = a) = \frac{M_n(a)}{n-1} (1 + o(1)) = o(\log(n-1)). \]

Hence
\[ \text{E}(u_n | \alpha = a) = (n-1)^{1-o(1)}. \]

Since
\[ \text{Var}(u_n | \alpha = a) \leq \text{E}(u_n | \alpha = a) \]
it follows (due to the second moment method, see [3])

$$\Pr(u_n = 0 \mid \alpha = a) \leq \frac{\text{Var}(u_n \mid \alpha = a)}{\text{E}(u_n \mid \alpha = a)^2} \leq \frac{1}{\text{E}(u_n \mid \alpha = a)}$$

or finally

$$\Pr(u_n = 0 \mid M_n = o(n \log n)) = O\left(\frac{1}{n^{1-o(1)}}\right).$$

The previous arguments were independent of the density of the graph. We argued that if we traverse $o(n \cdot \log(n))$ edges, then some nodes remain unmarked. Now, we consider dense graphs, namely with more than $n \log(n)$ edges (roughly speaking). In this case, almost all nodes are eventually marked and the edges traversed are indeed $\Omega(n \log(n))$ with high probability.

**Lemma 4.6** Let $G_{n,p}$ be a random digraph with $p \geq (1 + \varepsilon) \cdot \log(n)/n$, $\varepsilon > 0$, and $p = o(1)$. Next, let $M_n$ denote the number of edges traversed during the exploration. Then

$$\text{E}(M_n) = \Omega(n \cdot \log(n)),$$

independently of the exploration strategy.

**Proof** Assume we explore the graph according to some strategy. The strategy used determines the vector $\alpha$ of traversed edges or, more precisely, its distribution. Due to $p \geq (1 + \varepsilon) \cdot \log(n)/n$, the graph is almost surely (strongly) connected. Therefore, almost all nodes are eventually marked. Further, from $p = o(1)$ it follows $\alpha_i/n = o(1)$ for all $i$ (with high probability). In fact, the probability that a vertex exists with out-degree greater than $\gamma$ is bounded by

$$\Pr(\exists v : \gamma^+(v) \geq \gamma) \leq n \cdot \left(\frac{n-1}{\gamma}\right) \cdot p^\gamma \leq n \cdot \left(\frac{n^\gamma}{\gamma!} \cdot p^\gamma\right)$$

Using Stirling's formula for to approximate $\gamma!$, we get

$$\Pr(\exists v : \gamma^+(v) \geq \gamma) \leq n \cdot \frac{\gamma}{e} \cdot \left(\frac{n^\gamma}{\gamma} \cdot e \cdot p\right)^\gamma$$

If $\gamma = \Omega(n)$, then

$$\Pr(\exists v : \gamma^+(v) \geq \gamma) = \Theta(n^2) \cdot (O(p))^n = \Theta(n^2) \cdot (o(1))^n \approx 0.$$
That is, the maximal degree of a node is $o(n)$ with high probability. This implies that the conditions of Lemma 4.5 are satisfied.

Now, assume

$$M_n(a) = o(n \cdot \log(n)).$$

From Lemma 4.5 we infer

$$\Pr(u_n = 0) = O(n^{-1+o(1)}).$$

But as mentioned before, $u_n = 0$ with high probability, which is a contradiction. Therefore,

$$M_n(a) = \Omega(n \cdot \log(n))$$

with high probability. Hence,

$$\mathbb{E}(M_n) \geq \sum_a M_n(a) \cdot \Pr(\alpha = a)$$

$$\geq \Omega(n \cdot \log(n)) \cdot (1 - o(1)) \sum_a \Pr(\alpha = a)$$

which completes the proof.

So far, we have investigated the lower bound. It remains to determine the upper one.

**Lemma 4.7** Let $G_{n,p}$ be a random digraph with $p \geq (1 + \varepsilon) \log(n)/n$, $\varepsilon > 0$. Let $M_n$ be the number of edges traversed during a node exploration. Then

$$\mathbb{E}(M_n) = O(n \log(n)).$$

**Proof** Since $p \geq (1 + \varepsilon) \cdot \log n/n$, $\varepsilon > 0$, the graph is almost surely strongly connected [36]. In this case, all nodes are eventually marked. Consider the search at an arbitrary point. Let $U$ be the set of the currently unmarked nodes, and assume $|U| = \ell$. We select a node $v$ and we traverse an incident edge $e = (v, w)$, not yet traversed. Suppose that $k$ edges incident to $v$ have been traversed in previous steps. From Lemma 4.2 we infer

$$\Pr(w \in U) = \frac{\ell}{n - 1 - k} \geq \frac{\ell}{n - 1}. $$
Therefore, the expected number of edges traversed until a new node is marked is bounded by \((n-1)/\ell\). By summing up over all \(\ell\) we obtain

\[
E(M_n) \leq \sum_{\ell=1}^{n-1} \frac{n-1}{\ell} = O(n \log(n)).
\]

(4.7)

If the graph is not connected, at most all the existing edges are traversed. But the probability that this happens is small. Let \(p \overset{\text{def}}{=} \beta_n \cdot \log(n)/n\) where \(\beta_n \geq (1 + \varepsilon)\). Then the average number of existing edges is \(O(\beta_n \cdot n \cdot \log(n))\), and

\[
E(M_n) = O(n \cdot \log(n)) \cdot (1 - O(n^{1-\beta_n} \cdot \beta_n \log(n)))
\]

\[+
O(\beta_n \cdot n \log(n) \cdot n^{1-\beta_n} \cdot \beta_n \log(n))
\]

\[= O(n \cdot \log(n))
\]

since the graph is not connected with probability

\(O(n^{1-\beta_n} \cdot \beta_n \log(n))\),

see (3.38).

Lemma 4.6 and Lemma 4.7 give finally the claimed expected running time for a node exploration.

**Theorem 4.8** Let \(G_{n,p}\) be a random digraph with

\[p \geq (1 + \varepsilon) \cdot \frac{\log(n)}{n},\]

\(\varepsilon > 0\), and \(p = o(1)\). Next, let \(M_n\) denote the number of edges traversed during the exploration. Then

\[E(M_n) = \Theta(n \cdot \log(n))\]

independently of the exploration strategy.

4.4 **Searches with Local Information**

During a node exploration, we selected the nodes according to an arbitrary strategy. However, some problems do not allow this flexibility.
For instance, assume the only information about the graph is the number of vertices $n$, a node $s$ (which is marked), and the edges incident to $s$. Every time we traverse an edge and we mark a node $w$, we gain new informations about the graph, namely the node $w$ self and its incident edges. Therefore, the unique way to explore the graph is to traverse edges incident to marked nodes. We call explorations of this kind *searches with local information*.

The main difference with a node exploration is that only vertices in a *strong* connected component are eventually marked, i.e. all nodes reachable from $s$. During a simple node exploration, all nodes are eventually marked, except the ones with no ingoing edges. However, since for

$$ p \geq (1 + \varepsilon) \cdot \frac{\log(n)}{n} $$

the graph is almost surely strongly connected – and therefore all vertices have ingoing edges with high probability – the analysis of a node exploration remains valid for the searches with local information, too.

**Theorem 4.9** Let $G_{n,p}$ be a random digraph with

$$ p \geq (1 + \varepsilon) \cdot \log(n)/n, $$

$\varepsilon > 0$, and $p = o(1)$. We explore the graph using local information. Then we need to explore

$$ E(M_n) = \Theta(n \cdot \log(n)) $$

edges on average, independently of the search strategy.

**Remark** Theorem 4.9 implies that the expected running time of both Breadth-First-Search and Depth-First-Search is $\Theta(n \cdot \log(n))$. It is not self-evident that the two algorithms have similar behavior. Consider a nearly complete random graphs, namely when $p \approx 1$. Then, the expected running time of Depth-First-Search remains $\Theta(n \log(n))$, in contrast to the one of Breadth-First-Search which becomes $O(n)$. This particular behavior will play an important role also for the analysis of the single source shortest paths problem, as we will see in the next section.
4.5 Shortest Paths and Minimum Spanning Tree

A concrete problem that can be interpreted as an exploration with local information is the single source shortest paths problem. Given a directed graph in which all edges have a non-negative weight, we want to compute the shortest paths between a source (i.e. a fixed node in the graph) and all the other nodes. We consider a random graph $G_{n,p}$ with

$$p \geq (1 + \varepsilon) \cdot \log(n)/n,$$

where $\varepsilon > 0$ and $p = o(1)$. Recall that the weights are end-point independent. Furthermore, only the following operations are available:

1. we can select a node $v$,
2. we can traverse the edge incident to the selected node $v$ not yet traversed with minimal weight,
3. we can mark the end-node of a traversed edge.

This model corresponds to the one proposed in [33]. But in this case no restriction for the values of the weights is required (except that they are non-negative). Moreover, we can interpret dense random graphs as complete graphs where all but $p_i \cdot n$ edges incident to node $i$ have weight infinity, which extends the class of weights given in [33].

We want to determine a lower bound for the number of edges we have to traverse until the shortest path is computed for every node in the graph. We consider the following strategy (basically, Dijkstra's algorithm [11]). Assume the paths are computed for the first $n - \ell$ nearest nodes, which are stored in the set $N$. To find the $(n - \ell + 1)$-th nearest node, we consider the edge $e = (v, w) \in N \times V$ not yet traversed (namely in $E'$), which minimizes $\text{dist}(s, v) + C(e)$, compare Figure 4.1. Then

$$\text{dist}(s, w) \leftarrow \text{dist}(s, v) + C(e)$$

and $w$ is added to $N$. In this way, we select a node $v$ according to some strategy, or distribution (depending on $\text{dist}(s, v)$), but we traverse an incident edge $e$ chosen randomly among the ones not yet traversed (because of the end-point independent model). Therefore, this strategy corresponds to a search with local information and it requires to traverse $\Theta(n \cdot \log(n))$ edges on average.
SP \((G = (V, E, C), s \in V)\):

1. \(N \leftarrow \{s\}; \ d[s] \leftarrow 0;\)
2. \(\text{forall } v \in V - \{s\} \ \text{do } d[v] \leftarrow \infty \ \text{od};\)
3. \(E' \leftarrow \{(s, u) | u \in \Gamma^+(s)\};\)
4. \(\text{while } N \neq V \text{ and } E' \neq \emptyset \text{ do}\)
5. \(\text{select an edge } e = (v, w) \in E' \text{ which minimizes } d[v] + C(e);\)
6. \(\text{if } w \notin N \text{ then}\)
7. \(N \leftarrow N \cup \{w\};\)
8. \(d[w] \leftarrow d[v] + C(e);\)
9. \(E' \leftarrow E' \cup \{(w, u) | u \in \Gamma^+(w)\}\)
10. \(\text{fi};\)
11. \(E' \leftarrow E' - \{e\}\)
12. \(\text{od}.\)

Figure 4.1: Shortest Paths Algorithm

Moreover, the edges considered by this strategy must be traversed to obtain the correct result. In fact, we can ignore some of these edges only if we are sure that the corresponding end-nodes are already marked. But the unique way to know this is to traverse them. This means that to solve the single source shortest paths problem we need to traverse \(\Omega(n \cdot \log(n))\) edges, on average. Furthermore, there are algorithms which find a correct solution and run in expected (amortized) time \(O(n \cdot \log(n))\), including the operations to determine the edge to explore, see for instance [33].

Theorem 4.10 The single source shortest paths problem in a random digraph \(G_{n,p}\), where \(p = o(1)\) and \(p \geq (1+\epsilon) \cdot \log(n)/n\) with \(\epsilon > 0\), can be solved in the end-point independent model in expected time \(\Theta(n \cdot \log(n))\) if only the outgoing edges can be traversed.

In the shortest paths problem, the goal was to find the shortest paths from a source to every other node. A similar problem consists in determining the minimum spanning tree. More precisely, given a starting node we look for the paths to all reachable nodes such that the sum of all weights of the edges in the tree is minimized. We apply the same strategy as for the shortest paths, but we select the edge to traverse according to another test: We do not choose the edge \(e = (v, w)\) which minimize \(dist(s, v) + C(e)\) but simply the edge with minimal weight.
Therefore, this problem corresponds to an exploration with local information, too, and it requires the same running time.

**Theorem 4.11** The minimum spanning tree in a random digraph $G_{n,p}$, where $p = o(1)$ and $p \geq (1 + \varepsilon) \cdot \log(n)/n$ with $\varepsilon > 0$, can be determined in the end-point independent model in expected time $\Theta(n \cdot \log(n))$ if only the outgoing edges can be traversed.

Unfortunately, this analysis does not cover the general case, for instance when the graph is nearly complete (i.e. $p$ tending to 1). In fact, an exploration with local information can degenerate to the exploration of the edges incident to the starting node (e.g. by using BFS). In this case, the assumption about the weights plays an important role, as pointed out by MEHLHORN and PRIEDE [33]. Further, if the graph is sparse (i.e. $p \ll \log(n)/n$), there are algorithms which solve the problem in time $o(n \log(n))$. For example, THORUP [48] proposed a method to compute the shortest paths in an undirected graph with integer weights in linear time (in the number of nodes and edges). The analysis for general (sparse) graphs remains an open problem.

### 4.6 Non-homogeneous Random Graphs

The analysis of search algorithms like Breadth-First-Search is particularly useful to investigate structural properties of a random graph. For instance, KARP [24] determined in this way the expected size of the connected components in a graph $G_{n,p}$ for $p = c/n$, where $c$ is a constant greater than 1. This size corresponds to the number of nodes marked at the end of a search that uses local information.

Furthermore, this approach allows to determine how many edges must exist such that the graph is almost surely connected. For a homogeneous random graph $G_{n,p}$, it is well known that the threshold is given by $p = \log(n)/n$. In other words, if the graph contains less than $(1 - \varepsilon) \cdot n \log(n)$ edges (on average), then it is almost surely not connected. This condition can be extended to non-homogeneous random graphs.

**Definition 4.12** The vertex set of a non-homogeneous random digraph $G_{n,p} = (V, E)$ is given by $V = \{1, \ldots, n\}$ and the vector $p = (p_1, \ldots, p_n)$ determines the existence probability of the edges out-going edges incident
to node 1, 2, ..., n, respectively. More precisely, the edges \( e = (i, j) \), for each \( j \in V - \{i\} \), exist with probability \( p_i \), independently of the other edges.

Then, a necessary condition for the connectedness is shown in the following theorem.

**Theorem 4.13** Let \( G_{np} \) be a non-homogeneous random graph with

\[
p_i \cdot n = o(n),
\]

for all \( i \in V \). If

\[
E(|E|) = n \cdot (p_1 + p_2 + \cdots + p_n) \leq (1 - \varepsilon) \cdot n \log(n),
\]

where \( \varepsilon \in (0, 1) \), then the probability that the graph is (strongly) connected is \( O(n^{-\varepsilon + o(1)}) \).

**Proof** We consider the number of nodes with no ingoing edges. To determine this number, we construct the random graph by adding the edges incident to node 1, 2, ..., n according to \( p \). Then, let \( u_t \) denote the number of nodes with no ingoing edges when vertex \( t \) is handled, namely

\[
u_t \overset{\text{def}}{=} n - |\Gamma^+(1) \cup \cdots \cup \Gamma^+(t)|.
\]

Then, by similar argument as in Lemma 4.4 and Lemma 4.5, we get

\[
E(u_n) = n \cdot \prod_{i=1}^{n} (1 - p_i)
\]

and

\[
\log(n) - \log(E(u_n)) = \sum_{i=1}^{n} p_i \cdot (1 + o(1)) = \frac{|E|}{n} \cdot (1 + o(1)).
\]

If \( |E| \leq (1 - \varepsilon) \cdot n \log(n) \), then

\[
E(u_n) \geq n^{\varepsilon - o(1)}
\]

which together with

\[
\text{Var}(u_n) \leq E(u_n)
\]
(compare Lemma 4.4) implies

\[ \Pr(G_{n,p} \text{ is (strongly) connected}) \leq \Pr(u_n = 0). \]

Finally,

\[ \Pr(u_n = 0) \leq \frac{\text{Var}(u_n)}{E(u_n)^2} \leq n^{-\varepsilon + o(1)} \]

completes the proof.

In contrast to the homogeneous case, the condition

\[ E(|E|) \geq (1 + \varepsilon) \cdot n \log(n) \]

is far from being sufficient to guarantee connectedness. Furthermore, in non-homogeneous random graphs strong and weak connectedness have a different behavior. In the homogeneous case, if \( p \geq (1 + \varepsilon) \cdot \log(n)/n \), then all nodes are reachable from a given starting node \( s \) with high probability, cf. (3.38). The same result holds if we invert the direction of the edges, i.e. node \( s \) is reachable from any node in the graph with high probability. These two conditions imply the strong connectedness.

In the non-homogeneous case, we cannot use this argument. The reachability of all nodes from \( s \) implies only that the graph is weakly connected.

**Lemma 4.14** Let \( s \in V \) be a given node. Furthermore, let \( \rho_i, i \in V \), be probabilities satisfying

\[ \frac{\omega_n}{n} \leq \rho_i = o(1), \quad i = 1, 2, \ldots, n \]

where \( \omega_n \to \infty \) as \( n \to \infty \) and

\[ \sum_{i \geq 1} \rho_i \geq (1 + \varepsilon) \cdot \log(n) \]

for some \( \varepsilon > 0 \). If the values of \( p_i, i = 1, \ldots, n, \) are set to

\[ p_i = \rho_{\pi(i)} \]

according to a random permutation \( \pi \) over \( V \), then all nodes are almost surely reachable from \( s \).
Proof We build the random graph in two steps. In the first one, we generate a homogeneous random graph $G_1 = (V, E_1)$ with vertex set $V = \{1, \ldots, n\}$ and $E_1$ generated according to $p = \omega_n/n$. In the second step, we build a non-homogeneous random graph $G_2 = (V, E_2)$ according to $p'$, where

$$
\left(1 - \frac{\omega_n}{n}\right) \cdot \left(1 - p'_i\right) = 1 - p_i.
$$

Note that the vertex set is the same. The union of the two graphs

$$
G = (V, E_1 \cup E_2)
$$

gives a non-homogeneous random graph $G_{n,p}$ since

$$
\Pr((i,j) \notin G) = \Pr((i,j) \notin E_1 \cup E_2) = \left(1 - \frac{\omega_n}{n}\right) \cdot \left(1 - p'_i\right).
$$

Hence,

$$
\Pr((i,j) \notin G) = 1 - p_i.
$$

Now, assume without loss of generality that $\omega_n = o(\log(n))$. Let $U_1$ be the set of nodes in $G_1$ unreached from the given starting vertex $s$. Then, for $u_1 = |U_1|$ we get

$$
\mathbb{E}(u_1) = O\left(n \cdot \omega_n \cdot e^{-\omega_n}\right)
$$

as shown in Corollary 3.6. Markov inequality implies that

$$
(4.10)\quad u_1 \leq n \cdot \omega_n^2 \cdot e^{-\omega_n} \overset{\text{def}}{=} n_0
$$

with high probability and $n$ enough large.

In the second step, we generate a graph with $|E_2|$ edges according to $p'$. Since $\omega_n = o(\log(n))$, the expected number of edges in the first graph $G_1$ are

$$
\mathbb{E}(|E_1|) = o(n \log(n)).
$$

Hence,

$$
(4.11)\quad \mathbb{E}(|E_2|) = (1 + \varepsilon) \cdot n \log(n) - o(n \log(n))
$$

$$
\geq (1 + \varepsilon') \cdot n \log(n),
$$

where $0 < \varepsilon' < \varepsilon$. Furthermore, every edge in $G_2$ starts from a node in $U_1$ with probability $u_1/n$ because we assigned the probabilities $p_i$
(and therefore $p_i'$) according to a random permutation $\pi$. Therefore, the number $M_0$ of edges in $G_2$ starting from nodes in $U_1$ (for $n$ large) satisfies
\[
\mathbb{E}(M_0 \mid u_1 \leq n_0) \leq |E_2| \cdot \omega_n^2 \cdot e^{-\omega_n} \overset{\text{def}}{=} m_0.
\]
Due to Markov inequality,
\[
\Pr(M_0 \geq m_0 \cdot e^{\omega_n/2} \mid u_1 \leq n_0) = O\left(e^{-\omega_n/2}\right) \approx 0
\]
which implies
\[
M_0 = o(|E_2|)
\]
with high probability. Therefore, from (4.10) and (4.12) we infer that the event
\[
\mathcal{E} \overset{\text{def}}{=} \left(u_1 \leq n_0 \wedge M_0 = o(|E_2|)\right)
\]
holds with high probability. Next, the expected number of edges added in $G_2$ starting from vertices in $V - U_1$ is given by
\[
\mathbb{E}(|E_2| - M_0) \geq \mathbb{E}(|E_2| - M_0 \mid \mathcal{E}) \cdot \Pr(\mathcal{E}) \overset{(4.11)}{\geq} (1 + \varepsilon') \cdot n \cdot \log(n) \cdot (1 - o(1))
\]
for some $\varepsilon'' > 0$. Theorem 4.13 implies that almost all nodes will be incident to at least one of these edges. Thus, almost all nodes will be eventually reached – either directly through paths in $E_1$ or through an additional edge $e \in E_2$ starting from nodes in $V - U_1$.

**Corollary 4.15** Let $G_{n,p}$ be a non-homogeneous random graph with
\[
\frac{\omega_n}{n} \leq p_i = o(1), \quad i = 1, 2, \ldots, n,
\]
where $\omega_n \to \infty$ as $n \to \infty$, and
\[
\sum_{i \geq 1} p_i \geq (1 + \varepsilon) \cdot \log(n)
\]
for some $\varepsilon > 0$. Then, the graph is almost surely weakly connected.

**Remark** It is not clear if $\mathbb{E}(|E|) \geq n \log(n)$ implies weakly connectedness. However, it is surely not a sufficient condition for strong connectedness. For further discussions we refer to [45].
4.7 Extended Node Exploration

So far, we considered a simple node exploration. We have seen that it requires to traverse $\Theta(n \log(n))$ edges, on average. We will see that for an extended node exploration the expected running time can be reduced to $O(n)$. Recall that an extended node exploration is a node exploration in which also ingoing edges can be traversed. We consider a simple example to show the basic argument (namely, we investigate the reachability from a given node). We will apply the same method to compute the connected components of an undirected random graph in expected time $O(n)$. These two problems have been considered in [24] and [25], respectively. We will give a much simpler proof which, additionally, points out the similarities of the problems. For other applications we refer to [24, 41].

Theorem 4.16 Let $G_{n,p}$ be a random digraph with $p \in [0,1]$, and $s \in V$. Then it is possible to determine the nodes reachable from $s$ in expected time $O(n)$.

Proof This problem can be solved in two phases. First, a BFS-like search is started. At any step, an unvisited marked node is selected and all its incident edges are traversed. If all marked nodes are already visited, we continue the search by selecting an unmarked node which is marked with a new flag. The nodes treated successively are marked with the last introduced flag. The first phase stops when at least $n/2$ nodes are marked. The first phase — and the algorithm self — could stop already when all marked nodes have been visited. We continue the search to demonstrate the similarity with the undirected case.

We have to determine how many steps are executed. Consider the number of unmarked nodes at step $t$. Let $v_t$ be the selected node. By definition, every unmarked node is adjacent to $v_t$ with probability $p$. Then

$$\mathbb{E}(u_t \mid u_{t-1} = u) = u \cdot q$$

where $u_t$ denotes the number of unmarked nodes at step $t$. Hence

$$\mathbb{E}(u_t) = \mathbb{E}(u_{t-1}) \cdot q = \mathbb{E}(u_0) \cdot q^t = (n-1) \cdot q^t.$$

Similarly,

$$\text{Var}(u_t) = (n-1) \cdot q^t \cdot (1-q^t).$$
Let
\[ \tau = \left\lfloor \frac{\log(4)}{-\log(q)} \right\rfloor. \]

Then, from Cantelli's inequality we get
\[
\Pr(u_{\tau} \geq \frac{n-1}{2}) \leq \frac{\text{Var}(u_{\tau})}{\text{Var}(u_{\tau}) + \left(\frac{n-1}{2} - \mathbb{E}(u_{\tau})\right)^2} = \mathcal{O}\left(\frac{n}{n + \frac{(n-1)^2}{4}}\right) = \mathcal{O}(n^{-1}).
\]

Therefore, in the first phase less than \( \tau = O(1/p) \) steps are executed with probability \( 1 - O(n^{-1}) \), otherwise the number of steps is bounded by \( n \). Since at any step we traverse \( p \cdot n \) edges, on average, the expected number of edges traversed in the first phase is
\[
\mathbb{E}(M_n^{(1)}) \leq \tau \cdot p \cdot n \cdot (1 - O(n^{-1})) + n \cdot p \cdot n \cdot O(n^{-1}) = O(n).
\]

In the second phase, we have to handle the second half of the nodes. To do this, an unmarked node is selected and a subsearch is started. Then, we execute a search with local information (e.g. a DFS) in which, this time, the ingoing edges are considered in place of the outgoing ones. Furthermore, the nodes are not marked immediately after the corresponding edge is traversed, but at the end of the subsearch. A subsearch stops either when a path between the starting node and a marked node \( w \) is found, or if all incident edges are exhausted, namely if a whole connected component has been explored. In the first case, all nodes on the path are marked with the same flag as \( w \). In the second case, with a new flag. This step is applied to other unmarked nodes until all vertices have been considered.

The nodes marked in the first phase are more than \( n/2 \). Every time an edge is chosen, it leads to marked nodes at least with probability
\[
\varrho \overset{\text{def}}{=} \frac{n/2 - \tau}{n - 1} \geq 1 - \frac{1}{2} \frac{1}{p \cdot n}
\]
since the edge cannot be incident to any of the \( \tau \) nodes treated in the first phase. If \( p \cdot n \to \infty \), then \( \varrho \approx 1/2 \), otherwise the expected number of existing edges is \( O(n) \), which is a bound for the running time. Therefore, we may assume w.l.o.g. that \( \varrho \) is a constant (near to 1/2). Hence, the number of nodes handled during every subsearch are few, namely
\[
\Pr(\# \text{ handled nodes} = k) \leq \varrho^k.
\]
4.8. Conclusion

If $k$ nodes are handled, then at most $k^2 + 1$ edges are traversed. Therefore, the expected number of edges traversed is bounded by

$$\sum_{k \geq 0} (k^2 + 1) \cdot \Pr(\text{\# handled nodes} = k) = O(1).$$

Hence, the whole cost (in edges) of the second phase is

$$E(M_n^{(2)}) = O(n)$$

because we execute at most $n/2$ subsearches. This implies

$$E(M_n) = E(M_n^{(1)}) + E(M_n^{(2)}) = O(n)$$

which completes the proof. \hfill \Box

The same argument can be used to determine the connected components in an undirected random graph. In this case, the nodes with the same flag form a connected component.

**Theorem 4.17** Let $G_{n,p}$ be a random undirected graph. Then it is possible to determine the connected components in expected time $O(n)$. \hfill \Box

4.8 Conclusion

The analysis of a node exploration pointed out how other important algorithms, interpreted as special cases of explorations, have similar behaviors in dense random graphs. In particular, a better understanding of the processes revealed the trade-off between reached nodes and running time, which can be exploited for testing the connectedness (or computing the components) more efficiently. In contrast to the MST, this does not seem to apply for the shortest paths problem. It is very hard to imagine that some edges traversed during the algorithm of Dijkstra do not need to be considered. This induces to think that the lower bound of $\Omega(n \log n)$ edges traversed to determine the SSSP in dense random graphs holds in general (in the end-point independent model).
Part II

Searching in Sets
Leer - Vide - Empty
Chapter 5

Randomized Search Trees

5.1 Introduction

Maintaining a set of distinct keys \( S \) with the operations access, insert and delete is probably the most important problem in computer science. Various techniques have therefore been proposed and analyzed. A natural approach consists of a hierarchical division into subproblems, according to the divide and conquer paradigm. We consider the realization as a search tree – which is a binary tree such that the keys \( S \) are associated with the internal nodes (vertices) satisfying the constraint that the key in every vertex is greater than any key in its left subtree and less than any key in its right subtree.

Then the search for a key \( x \) in such a tree \( T \) can be realized easily by a recursive procedure. The termination of the search is unsuccessful if \( T \) is empty, successful if the key associated to the root (say \( r \)) is identical to \( x \). Otherwise we look in the left subtree if \( x < r \) or in the right one if \( x > r \). Similarly we can formulate an insert rule as follows: Search (unsuccessfully) for \( x \) and insert a new node with key \( x \) in place of the leaf (external node) where the search has finished.

To construct a search tree \( T \) for the set \( S = \{x_1, \ldots, x_n\} \) we add the keys one by one to an initially empty tree using the procedure described above. Let \( \varphi \) be a permutation of \( \{1, 2, \ldots, n\} \), then we observe that
for the input sequence
\[(5.1)\quad x_{\varphi(1)}, \ldots, x_{\varphi(n)}\]
there exists a unique determined search tree \(T = T(\varphi)\), the so called search tree induced by \(\varphi\).

It is well known that if every permutation of \(\{1, \ldots, n\}\) is equally likely in (5.1), then the average depth of an internal node is about \(1.386 \cdot \log n\), and this is nearly optimal [31]. A search tree produced on this assumption is a random search tree.

But in concrete applications the input is often not in random order. ARAGON and SEIDEL pointed out in [5] that the original input sequence can be replaced by a randomly generated one. In other words, the following was shown:

For every key \(x_i\) we choose a random number \(p(x_i)\) uniformly from the interval \([0, 1]\), the priority of \(x_i\). Then the numbers \(p(x_1), \ldots, p(x_n)\) are distinct with probability 1. On this assumption, there is a unique permutation \(\pi\) of \(\{1, \ldots, n\}\) such that
\[(5.2)\quad p(x_{\pi(1)}) > p(x_{\pi(2)}) > \cdots > p(x_{\pi(n)}).
\]
By construction, \(\pi\) is a random permutation. Then it is possible to build a search tree \(T\) induced by
\[(5.3)\quad x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}.
\]
Such a tree is called a randomized search tree for \(S\). Note that \(T\) is heap-ordered with respect to the priorities, i.e. the priority of any node is greater than the priority of any of its children.

This strategy leads to a more complicated update operation. In general a new key \(z\) changes the input sequence (5.3) to
\[x_{\pi(1)}, \ldots, x_{\pi(i)}, z, x_{\pi(i+1)}, \ldots, x_{\pi(n)}\]
where \(i\) is given by
\[p(x_{\pi(i)}) > p(z) > p(x_{\pi(i+1)}).
\]
This means that \(z\) has to be inserted as the root of a subtree and no longer as leaf, see Figure 5.1.

The deletion of a key \(u\) is defined as an inverse insertion. First we find \(u\), then we rotate it downward until it has no children and finally
5.1. Introduction

insert(z, T)
(1) search for z in the tree T;
(2) add z in the appropriate leaf position;
(3) \( y \leftarrow \text{parent of } z \);
(4) while \( p(z) > p(y) \) do
(5) rotate z to the root of the subtree of y 
      (see Figure 5.3)
(6) \( y \leftarrow \text{parent of } z \);
(7) od.

Figure 5.1: Insertion of an element

we remove it. We rotate to the left or to the right with respect to the
heap condition of the priorities in the resulting tree.

One reason for the popularity of randomized search trees is their
nearly optimal behavior for weighted keys, especially if the weights are
not known in advance. A weight \( w(x_i) \in \mathbb{N} \) is associated to every key
\( x_i \) in \( S \). Let \( W \) be the sum of the weights of all the keys in \( S \). In
[5] the value \( w(x_i) \) signifies the number of accesses of the key \( x_i \). The
operation of accessing key \( x \) for weighted data is modified as described
in Figure 5.2.

access(z, T)
(1) choose a new random number \( u \) from \([0, 1]\);
(2) search for \( z \) in the tree \( T \);
(3) if \( u > p(z) \) then
(4) \( p(z) \leftarrow u \);
(5) rotate \( z \) upward until \( T \) is in heap order
      with respect to the priorities
(6) fi.

Figure 5.2: Access operation

ARAGON and SEIDEL showed:

The expected time necessary to access a key \( x_i \in S \) with weight \( w(x_i) \)
is

\[ O\left(1 + \log \left( \frac{W}{w(x_i)} \right) \right). \]

If we interpret \( w(x_i)/W \) as the probability to access the key \( x_i \) then (5.4) is known to be optimal within a constant factor, see [31].

Although randomized search trees work well in practice, some points should be improved. The first one refers to the generation of the priorities \( p(x_i) \). In order to guarantee that the input sequence is uniquely determined by (5.2), it is necessary that all generated priorities are distinct. This is not easy to realize, since especially for large weighted applications – \( p(x_i) \) approaches 1. Aragon and Seidel [6] propose some possibilities to handle this disadvantage (e.g. by using hash functions and pseudo-random numbers).

Next, in [6] the weights must be integer since they are interpreted as the number of (virtual) copies of the key. This restriction can be natural for some applications, but it is not in general.

Finally, the priority \( p(x_i) \) of every key \( x_i \) has to be stored in the data structure. Hence, \( p(x_i) \) is a random number that contains no further informations about the search tree. For instance, in BB[\( \alpha \)]-trees we store with every node the number of its successors. This can be useful for other purposes. Therefore, the question arises whether the priorities are really necessary for this data structure. This problem has been discussed in [6] and has been extensively treated in [38] for the unweighted case.

The solution is closely related with randomized Quicksort which is probably the best known randomized algorithm in the literature. Randomized Quicksort uses a random pivot element to split the problem into recursively solved subproblems. For the trees, the pivot element corresponds to the root of the search tree and the subproblems are equivalent to the subtrees. Therefore, one first chooses the root of the search tree at random and then proceeds recursively.

This perspective leads to an alternative algorithmic idea where the position of a new key in the input sequence is determined by a top-down decision process. In the next sections, we will extend this idea to weighted randomized search trees. We will store with every key the sum of all weights of the corresponding subtree instead of its priority. In this way, the weight of a key can be increased by an arbitrary value in just one update operation. Thus, the weights can be arbitrary positive real numbers. Furthermore, the problem connected to the randomness of the
priorities is avoided and, additionally, the analysis becomes particularly simple and elegant.

5.2 Weighted Random Search Trees

Suppose without loss of generality that
\[ S = \{x_1, \ldots, x_n\} = \{1, \ldots, n\}. \]

For every key \( x_i = i \) we have a positive weight \( w_i \). Let \( T \) be a search tree for \( S \) and \( T' \) any subtree of \( T \), then \( W(T') \) denotes the sum of all weights of the keys in \( T' \).

On these assumptions, we define a weighted random search tree \( T_n \) as follows.

**Definition 5.1** : A weighted random search tree \( T_n \) is a search tree for \( S \) produced with the rules:

1. The probability that the key \( i \in S \) becomes the root of \( T_n \) is given by
\[
\frac{w_i}{W(T_n)} = \frac{w_i}{w_1 + \cdots + w_n}.
\]

2. Every subtree of \( T_n \) is constructed in the same way.
The most important parameter of \( T_n \) is the depth \( d_n^i \) of the key \( i \). Let \( r(T_n) \) be the root of \( T_n \), then \( d_n^i \) represents the length of the path from \( i \) to \( r(T_n) \).

First we consider \( d_n^i \). Let

\[
D_{n,\ell} = \Pr( d_n^i = \ell ), \quad \ell = 0, \ldots, n - 1,
\]

then we get

\[
D_{1,0} = \frac{w_1}{w_1} = 1.
\]

For \( n \geq 2 \) we observe

\[
D_{n,\ell} = \sum_{k=1}^{n} \Pr( d_n^i = \ell | r(T_n) = k ) \cdot \Pr( r(T_n) = k )
\]

\[
= \frac{w_n}{w_1 + \cdots + w_n} \cdot D_{n-1,\ell-1} + \frac{w_1 + \cdots + w_{n-1}}{w_1 + \cdots + w_n} \cdot \sum_{k=1}^{n-1} D_{k-1,\ell-1} \cdot \frac{w_k}{w_1 + \cdots + w_{n-1}} = D_{n-1,\ell},
\]

which shows that \( d_n^i \) is a time dependent birth process with discrete time \( n \). For such processes the expected value can be easily determined.

**Lemma 5.2** Let \( W_i = w_1 + \cdots + w_i \). Then

\[
\mathbb{E}( d_n^i ) = \frac{w_2}{W_2} + \cdots + \frac{w_n}{W_n}.
\]

**Proof** First, we have

\[
\mathbb{E}( d_n^i ) = \sum_{\ell=0}^{n-1} \ell \cdot D_{n,\ell}
\]

\[
= \sum_{\ell=0}^{n-1} \ell \cdot \left( \frac{w_n}{W_n} \cdot D_{n-1,\ell-1} + (1 - \frac{w_n}{W_n}) \cdot D_{n-1,\ell} \right)
\]
\[
\begin{align*}
E(d_{n-1}^1) &= \frac{w_n}{W_n} \sum_{\ell=1}^{n-1} (\ell - 1) \cdot D_{n-1,\ell-1} \\
&\quad + (1 - \frac{w_n}{W_n}) \sum_{\ell=0}^{n-2} \ell \cdot D_{n-1,\ell} \\
&= E(d_{n-1}^1) + \frac{w_n}{W_n}
\end{align*}
\]

which together with

\[
E(d_1^1) = 0
\]

implies by induction on \( n \)

\[
E(d_n^1) = \frac{w_2}{W_2} + \cdots + \frac{w_n}{W_n}
\]

Now, we use

\[
\frac{w_i}{W_i} \leq \int_{w_i-w_i}^{W_i} \frac{1}{x} \, dx
\]

to simplify (5.5). In fact, from

\[
E(d_n^1) = \sum_{i=2}^{n} \frac{w_i}{W_i} \leq \sum_{i=2}^{n} \int_{W_i-w_i}^{W_i} \frac{1}{x} \, dx = \int_{W_1}^{W_n} \frac{1}{x} \, dx
\]

we get

\[
E(d_n^1) = \ln \left( \frac{W_n}{w_1} \right).
\]

By symmetry, it follows

\[
E(d_n^n) \leq \ln \left( \frac{W_n}{w_n} \right).
\]

Let us come back to \( d_i^r \). Note that \( d_n^i \) is identical to the number of predecessors \( A_i \) of \( i \) in \( T_n \) without \( i \) itself. The set \( A_i \) can be split into

\[
A_i^{\leq} \overset{\text{def}}{=} \{ j \in A_i \mid j < i \} \quad \text{and} \quad A_i^{>} \overset{\text{def}}{=} \{ j \in A_i \mid j > i \}.
\]
which implies

\[(5.8) \quad E(d'_n) = E(|A'_n|) + E(|A'_n|)\]

\[
\leq \ln \left( \frac{w_1 + \cdots + w_i}{w_i} \right) + \ln \left( \frac{w_{i+1} + \cdots + w_n}{w_i} \right)
\]

\[
\leq 2 \cdot \ln \left( \frac{W_n}{w_i} \right).
\]

This discussion can be summarized in the following theorem.

**Theorem 5.3** Let \( T \) be a weighted random search tree with total weight \( W(T) \). Then the average cost to access a key \( x \) with weight \( w(x) \) is given by

\[ O \left( 1 + \log \left( \frac{W(T)}{w(x)} \right) \right). \]

Next, we consider \( S_n \), the number of nodes in the subtree of key 1 (including 1 itself). Let \( S_{n,t} \) be the probability that key 1 has \( t \) successors in a weighted random search tree \( T_n \). Since key 1 can only have \( n \) successors if 1 is the root, we obtain

\[(5.9) \quad S_{n,n} = \frac{w_1}{W_n}. \]

For \( \ell < n \) we find

\[(5.10) \quad S_{n,\ell} = \sum_{k=2}^{n} \frac{\Pr(S_n = \ell | r(T_n) = k) \cdot \Pr(r(T_n) = k)}{\Pr(S_{k-1} = \ell)} = \frac{w_k}{w_n}. \]

Because of \( S_{n,n+h} = 0 \) for \( h > 1 \) it follows from (5.10)

\[ S_{n,n-1} = \frac{w_n}{W_n} \cdot S_{n-1,n-1} = \frac{w_n}{W_n} \cdot \frac{w_1}{W_{n-1}} \]

and, equivalently,

\[(5.11) \quad S_{n,n-1} = w_1 \left( \frac{1}{W_\ell} - \frac{1}{W_{\ell+1}} \right). \]

From (5.10) we infer for \( \ell < n - 1 \)

\[(5.12) \quad S_{n,\ell} = \frac{w_n}{W_n} \cdot S_{n-1,\ell} + \frac{W_{n-1}}{W_n} \cdot \sum_{k=2}^{n-1} \frac{S_{k-1,\ell} \cdot w_k}{W_{n-1}} = S_{n-1,\ell}. \]
\[ = \frac{W_{n-1} + w_n}{W_n} S_{n-1,t} \]

and therefore

\[(5.13) \quad S_{n,t} = S_{n-1,t} = \cdots = S_{t+1,t} \overset{(5.11)}{=} w_1 \left( \frac{1}{W_\ell} - \frac{1}{W_{\ell+1}} \right).\]

Furthermore, expressions (5.9) and (5.13) imply

\[(5.14) \quad \Pr(S_n \geq \ell) = S_{n,t} + S_{n,t+1} + \cdots + S_{n,n} = w_1 \left( \sum_{i=\ell}^{n-1} \frac{1}{W_\ell} - \frac{1}{W_{\ell+1}} + \frac{1}{W_n} \right) = \frac{w_1}{W_\ell}\]

and consequently

\[(5.15) \quad \mathbb{E}(S_n) = \sum_{\ell=1}^{n} \Pr(S_n \geq \ell) = \sum_{\ell=1}^{n} \frac{w_1}{W_\ell}.\]

To find the cost for an insert (or a delete) operation we will need a parameter called (in [5]) the left spine of the right subtree which means the length of the path from a node \(v\) to the smallest key in its right subtree. To be more exact, we define \(R^1_n\) as the length of the path from key \(i\) to key \(i+1\) in \(T_n\) if \(i+1\) is the successor of \(i\). If the right subtree of \(i\) is empty then we define \(R^1_n = 0\).

With Theorem 5.3, (5.10) and (5.12) we find for \(R^1_n\)

\[\mathbb{E}(R^1_n) = \sum_{\ell=1}^{n} \mathbb{E}(R^1_n \mid S_n = \ell) \cdot \Pr(S_n = \ell).\]

Because of

\[\mathbb{E}(R^1_n \mid S_n = \ell) \overset{(5.5)}{=} \begin{cases} 0 & : \text{for } S_n = 1 \\ 1 & : \text{for } S_n = 2 \\ 1 + \sum_{h=3}^{\ell} \frac{w_h}{W_h - w_1} & : \text{otherwise} \end{cases}\]
we get with $W'_t \overset{\text{def}}{=} W_t - w_1$

(5.16) $\mathbf{E}(R_n^1) = 1 - S_{n,1} + S_{n,3} \cdot \frac{w_3}{W_3} + S_{n,4} \cdot \left( \frac{w_3}{W_3} + \frac{w_4}{W_4} \right) + \cdots$

+ $\cdots + S_{n,n} \cdot \left( \frac{w_3}{W_3} + \cdots + \frac{w_n}{W_n} \right) \overset{(5.14)}{=} \frac{w_1}{W_3}$

+ $\cdots + \frac{w_n}{W_n} \left( S_{n,n} \right) \overset{(5.9)}{=} \frac{w_1}{W_n}$

= $1 + \sum_{h=3}^{n} \frac{w_h}{W_h - w_1} - \sum_{h=2}^{n} \frac{w_h}{W_h}$.

Note that

(5.17) $\sum_{h=3}^{n} \frac{w_h}{W_h - w_1} \leq \sum_{h=3}^{n} \int_{W_h - w_1 - w_h}^{W_h - w_1} \frac{1}{x} \, dx$

= $\int_{w_2 - w_1}^{w_n - w_1} \frac{1}{x} \, dx$

= $\ln \left( \frac{W_n - w_1}{w_2} \right)$

and

(5.18) $\sum_{h=2}^{n} \frac{w_h}{W_h} \geq \sum_{h=2}^{n} \int_{W_h}^{W_h + w_h} \frac{1}{x} \, dx$

= $\int_{W_2}^{W_n + w_n} \frac{1}{x} \, dx$

= $\ln \left( \frac{W_n + w_n}{W_2} \right)$.

Using these bounds in (5.16) we observe

$\mathbf{E}(R_n^1) \leq 1 + \ln \left( \frac{W_n - w_1}{w_2} \right) - \ln \left( \frac{W_n + w_n}{w_1 + w_2} \right)$.
Similarly, we define $L_n^i$ as the right spine of the left subtree, namely the length of the path from key $i$ to the key $i - 1$ in $T_n$ if $i - 1$ is the predecessor of $i$, otherwise $L_n^i = 0$. Then it follows

\begin{equation}
E(\ell_n^i) \leq 1 + \ln(1 + \frac{w_i}{w_{i+1}})
\end{equation}

and

\begin{equation}
E(L_n^i) \leq 1 + \ln(1 + \frac{w_i}{w_{i-1}}).
\end{equation}

From (5.19) and (5.20) we derive an estimate for the sum $L_n^i + R_n^i$, the so called spine of $x_i$.

**Theorem 5.4** Let $T$ be a weighted random search tree with total weight $W(T)$. Further, let $x, y, z$ be nodes of $T$ with weights $w(x), w(y), w(z)$ such that $x$ ($z$) is the predecessor (successor) of $y$ with respect to the order of the keys. Then the expected value of the spine of $y$ is given by

\[ O\left(1 + \log\left(1 + \frac{w(y)}{w(x)}\right) + \log\left(1 + \frac{w(y)}{w(z)}\right)\right). \]

The next point is to show the relationship between the random search trees defined above and the randomized search trees proposed by Aragon and Seidel [5].

**Lemma 5.5** A weighted randomized search tree is a weighted random search tree.

**Proof** Let $T$ be a weighted randomized search tree for the keys

\[ S = \{x_1, \ldots, x_n\}. \]

Furthermore, let $w(x_i)$ be the number of access operations on $x_i$ and $W(T)$ be the sum of all $w(x_i)$. For every access on $x_i$ we have chosen a random number from $[0, 1]$. Let $Z(x_i)$ be the set of all this numbers and

\[ Z(T) = Z(x_1) \cup \cdots \cup Z(x_n). \]
Any (fixed) key $x_i$ becomes the root of $T$ if
\[
\max(Z(x_i)) = \max(Z(T))
\]
which implies
\[
\Pr(r(T) = x_i) = \frac{|Z(x_i)|}{|Z(T)|} = \frac{w(x_i)}{W(T)}.
\]
Therefore the root of $T$ fulfills the definition of a weighted random search tree and the claim follows by induction on the height of $T$.

5.3 Algorithmic Aspects

So far, we have ignored the update operations for a weighted random search tree. Let us first consider the insertion of a new key $z$ with weight $w(z)$. We assume that for every key $x$ in $T$ the sum of all weights in the associated subtree $W(x)$ is stored together with $x$. To simplify the argumentation, we define
\[
W(x) = 0
\]
for any leaf $x$ in $T$. Further, we use a random generator $\text{COIN}(p)$ which returns "true" with probability $p$ and "false" with probability $1 - p$. Then the insert operation is described in Figure 5.4.

This procedure is similar to the insert operation proposed in [5]. First, we search for the key $z$ and we insert it in the corresponding leaf position. Then we rotate $z$ upward until a special position is reached. In contrast to [5], this position is determined by a top down process directly induced by Definition 5.1. For every subtree $T_v$ of $v$ considered during the search process, we toss a coin to decide whether $z$ should be the root of $T_v$ in the resulting tree and we take the first one with a positive answer.

It remains to show that the algorithm produces a weighted random search tree. We use induction on the number of operations and we assume that $T$ fulfills Definition 5.1 before the algorithm starts. The resulting tree is denoted by $T'$. Obviously, the new key $z$ becomes the root of $T'$ with probability
\[
\frac{w(z)}{W(T) + w(z)}.
\]
5.3. Algorithmic Aspects

**Figure 5.4: Insertion of an element in weighted search trees**

where \( W(T) \) stands for the total weight of \( T \). But what happens with another key \( x \) in \( T \)? Key \( x \) can only be the root \( r(T') \) of \( T' \) if \( z \) is not \( r(T') \). Therefore we obtain

\[
\Pr(r(T') = x) = \frac{w(x)}{W(T)} \cdot \left( 1 - \frac{w(z)}{W(T) + w(z)} \right)
\]

Hence, every key is chosen as \( r(T') \) in accordance with Definition 5.1 and it follows by induction on the height that \( T' \) is a weighted random search tree.

Until now, we have supposed that the key \( z \) is not already contained
in $T$. But this assumption is not really necessary and we can extend our insert operation similarly to [5] as follows: We stop the search for $z$ if we find it in $T$. At this moment node $u$ can either be defined as the new insert position for $z$ or it is left undefined. In the first case, we rotate $z$ upward to this new position, otherwise we do nothing. The weight of $z$ in the resulting tree corresponds to the sum of the weights of the old $z$ and the inserted one. It is evident that this strategy is consistent with Definition 5.1. Furthermore, if we insert only keys with weight 1 the observed tree is equivalent to that proposed by ARAGON and SEIDEL.

Next, we consider the deletion of a key $z$. We start again by finding the position of $z$. Then we rotate it downwards and delete the resulting leaf. The problem is to decide for every rotation of $z$, which of its children should replace $z$ as the root in the resulting subtree. In [5] this decision is determined by the values of the corresponding priorities. Let $T_l$ ($T_r$) denote the left (right) subtree of $z$, and let $T_{l,r}$ be the union of $T_l$ and $T_r$ in the final tree (without $z$). Furthermore, let $Z(T_l)$ ($Z(T_r)$) be the set of all random numbers chosen to determine the priorities for the keys in $T_l$ ($T_r$). Then in [5] we have $r(T_l) = r(T_{l,r})$ if and only if

$$\max(Z(T_l)) = \max(Z(T_l) \cup Z(T_r))$$

and this holds with probability

$$p_l \overset{\text{def}}{=} \frac{|Z(T_l)|}{|Z(T_l)| + |Z(T_r)|} = \frac{W(T_l)}{W(T_l) + W(T_r)}.$$

Hence, the random variable $T_{l,r}$ has $r(T_l)$ as root with probability $p_l$ and $r(T_r)$ with probability $1 - p_l$. According to this, we will determine the new root for $T_{l,r}$ with a corresponding coin tossing experiment and we obtain for any key $x$ from $T_{l,r}$

$$\Pr(r(T_{l,r}) = x) = \left\{ \begin{array}{lr} \Pr(r(T_{l,r}) = x | r(T_l) = x) \cdot \Pr(r(T_l) = x) \\ = \Pr(r(T_{l,r}) = r(T_l)) \cdot \frac{w(T_l)}{W(T_{l,r})} \\ = \frac{w(x)}{W(T_{l,r})} \end{array} \right.$$  

Since the same argumentation holds for any key of $T_r$, we observe that the root of $T_{l,r}$ satisfies Definition 5.1 which implies again that the final tree is a weighted random search tree.

**Theorem 5.6** Let $T$ be a weighted random search tree for the set $S$ with total weight $W = \sum_{x \in S} w(x)$, where $w(x)$ denote the weight of key
Furthermore, for any \( z \in \mathbb{R} \) let \( s(z) \) be the smallest key of \( S \) greater than \( z \) and correspondingly, let \( p(z) \) be the greatest element of \( S \) less than \( z \). Then it holds:

1. The expected time to access any key \( x \in S \) is
   
   \[
   O\left(1 + \log\left(\frac{W}{w(x)}\right)\right).
   \]

2. The average time to insert \( z \notin S \) into \( T \) is
   
   \[
   O\left(1 + \log\left(\frac{W + w(z)}{w(z)}\right) + \log\left(1 + \frac{w(z)}{w(p(z))}\right) + \log\left(1 + \frac{w(z)}{w(s(z))}\right)\right).
   \]

3. The deletion of \( y \in S \) from \( T \) needs an expected running time of
   
   \[
   O\left(1 + \log\left(\frac{W}{w(y)}\right) + \log\left(1 + \frac{w(y)}{w(p(y))}\right) + \log\left(1 + \frac{w(y)}{w(s(y))}\right)\right).
   \]

4. The expected number of rotations to insert or delete any key \( z \) is given by
   
   \[
   O\left(1 + \log\left(1 + \frac{w(z)}{w(p(z))}\right) + \log\left(1 + \frac{w(z)}{w(s(z))}\right)\right).
   \]

**Proof** It follows directly from the previous discussion. First, we have seen that insertions or deletions map a weighted random search tree into another one. Therefore, the expected access time is given by 1 plus the expected depth of a node. Hence, part 1 follows from Theorem 5.3. The number of rotations involved in an update operation for key \( z \) is bounded by its spine which shows that part 4 is given by Theorem 5.4. Since the running time for insertions or deletions is the sum of the corresponding access time and the time for the involved rotations, part 2 and 3 follow from 1 and 4.

### 5.4 Conclusion

The discussed realization of weighted randomized search trees allows arbitrary weight modifications for every update operation. Therefore, it is a proper extension of the original solution of Aragon and Seidel.
In order to handle the weights, we store with every key the sum of all weights in the corresponding subtree. Since the same technique is used for BB[α] or D-trees [31], it makes sense to consider weighted random search trees as a randomized version of D-trees. This is quite interesting because the original solution in [5] was proposed as an alternative to D-trees.

Randomized search trees offer a simple, efficient solution for the dictionary problem with average cost for every operation proportional to $\log n$. It is possible to reduce the time of the operations by exploiting the structure of the input. In the next chapter we will illustrate how to reach this with hashing.
Chapter 6

Hashing

6.1 Introduction

A function $h$ from a universe $U = \{0, \ldots, N - 1\}$ to a given range $\{0, \ldots, m - 1\}$ is called a hash function. The integers $0, \ldots, m - 1$ build the indices of an array $T$ noted as hash table. With respect to a given set $K \subseteq U$, the hash function $h$ is called perfect if $x \neq y$ implies $h(x) \neq h(y)$ for all $x$ and $y$, and, additionally, it is minimal if

$$m = |K| \overset{\text{def}}{=} n.$$ 

An element $x$ of $K$ is usually interpreted as a key with an associated information field, but this aspect can be ignored since it does not affect the algorithmic properties.

Hash functions offer an answer to the dictionary problem which can be considered conceptually opposite to search trees since they are not based on hierarchies. Like the search trees, they are a classic topic in computer science and have a vast literature, see [21] for a survey.

An important subset of hashing techniques is formed by perfect hashing which was first discussed by Sprugnoli [44]. Later, Fredman, Komlos, and Szemeredi [16] showed that for $m \geq 3 \cdot n$ simple perfect hash functions exist which can be determined deterministically in time$^1$

---

$^1$Throughout this chapter we use RAM-machines [1] (uniform cost assumption). Therefore, storage cells can hold arbitrary numbers and basic arithmetic and pointer operations take constant time.
The evaluation of \( h \) takes time \( O(1) \) in the worst case. The concept of Fredman, Komlos, and Szemeredi was extended by Dietzfelbinger, et al. \[10\] to dynamic perfect hashing, i.e. the key set \( K \) can be updated by insertions or deletions. Their algorithm achieves constant amortized expected time for update operations and constant worst case time for look-up operations. The required memory is still linear, but considerably increased.

If we consider the universe \( U \) to be indices of a one-dimensional field with nonempty entries \( K \), then hash functions can be interpreted as algorithms for table compression. Of course, this holds also in the opposite direction. From this point of view, the scheme proposed by Tarjan and Yao \[47\] to compress a large static \( a \times a \) matrix represents a special perfect hash function for the universe \( U = \{0, 1, \ldots, a^2 - 1\} \). They compress the matrix with \( n = O(a) \) nonempty entries by a system of arrays, each of size \( O(n) \), where the access time of \( O(1) \) (in the uniform cost measure) remains still valid. Their main result is a worst case strategy for finding a collision free overlapping of bit strings.

The solution of Tarjan and Yao has two handicaps. On the one hand, the authors themselves confess that their method is "a little complicated to use in practice" \[47, \text{p.} \, 606\]. On the other hand, their deterministic construction in time \( O(n^2 + a) \) is expensive in contrast to the probabilistic version of Fredman, Komlos, and Szemeredi \[16\] with construction time \( O(n) \).

These drawbacks can be avoided by randomizing the method, as shown in Section 6.3. The price is that the resulting method is a Monte Carlo algorithm which compresses the matrix only with high probability. But in this case, an optimal compression rate is almost surely achieved in linear expected time \( O(n) \). The technique can be used to design a perfect hashing method with

\[
K \subseteq U = \{0, 1, \ldots, N - 1\}.
\]

In Section 6.4, we show that for a static key set \( K \), the solution needs only a minimal table size \( m = n \) and one additional array of size \( n \), each entry less than \( n \). All other properties remain as in \[16\]. For a dynamic key set, the method has to be modified following the idea of Dietzfelbinger et al. \[10\]. Then, the expected size of the hash table can be maintained close to \( n \), and only four additional arrays of size \( m \), each entry less than \( m \), are needed (see Section 6.5). Note that a balanced search tree with \( n \) keys requires \( 2 \cdot n \) pointers and \( n \) weights.
6.2 Perfect Hashing and Table Compression

**Fredman, Komlos, and Szemeredi** [16] presented a variant of hashing which maps the keys into a table of size $m = 6 \cdot n$. They use the following class of hash functions

$$h_k(x) \equiv (k \cdot x \mod p) \mod q, \quad 1 \leq k \leq p - 1,$$

where $p$ is a prime greater than $N$ and $q$ is some integer (specified later). The parameter $k$ is chosen uniformly at random between 1 and $p - 1$. Then, the function $h_k$ distributes a value $h_k(x)$, $x \in K$, pseudo-randomly over the range $\{0, \ldots, q - 1\}$. For any $k$, the function $h_k$ is usually not perfect, i.e. there are elements $x$ and $y$ of $K$ such that $h_k(x) = h_k(y)$. For $q = O(n)$ the probability that such collisions exist is very high, but their number is small. Let

$$W_j = \{x \in K \mid h_k(x) = j\}, \quad j = 0, \ldots, q - 1,$$

then it follows for a random selection of $k \in \{1, \ldots, p - 1\}$

$$\mathbb{E}(\sum_{0 \leq j < q} |W_j|^2) < n + \frac{2 \cdot n^2}{q}$$

and further

$$\Pr(\sum_{0 \leq j < q} |W_j|^2 < n + \frac{4 \cdot n^2}{q}) \geq \frac{1}{2}.$$

Choosing $q = 2 \cdot n$, at least half of the values $k$ in $U$ lead to

$$\sum_{0 \leq j < q} |W_j|^2 < 3 \cdot n.$$

Such a function, say $h_k$, is used to partition $K$ into blocks $W_j$, for $0 \leq j < q$. This builds the first step for constructing the perfect hash function $h$. The final step arises again from (6.3), now applied to every $W_j$. With $q = 2 \cdot |W_j|^2$, at least half of the values of $k$ lead to an injective function $h_k$ when restricted to $W_j$. Let $k_j$ be such a value. Then the perfect hash function $h$ is given by

$$h(x) \equiv \left(\sum_{\ell=0}^{j-1} 2 \cdot |W_\ell|^2\right) + ((k_j \cdot x) \mod p) \mod 2 \cdot |W_j|^2$$
where
\[ j \overset{\text{def}}{=} ((k' \cdot x) \mod p) \mod 2 \cdot n. \]

Since \( 0 \leq h(x) < 6 \cdot n \), the hash table has size \( m = 6 \cdot n \). Further, the values \( k_j \) and
\[ c_j \overset{\text{def}}{=} 2 \cdot |W_0|^2 + \cdots + 2 \cdot |W_j|^2 \]
are stored in separate fields to allow the evaluation of \( h(x) \) in time \( O(1) \). The expected time to find \( h \) is linear in \( n \).

Improvements of the storage complexity are possible. For instance, Fredman, Komlos, and Szemeredi reduced the memory requirements to a table of size \( n \) and \( o(n) \) additional storage. However, they achieved this only deterministically in time \( O(n \cdot N) \). Mehlhorn [32] presents a probabilistic variant with \( m = 4 \cdot n \).

Let us turn to the Tarjan-Yao compression scheme [47]. Let \( A \) be an \( a \times a \) matrix with \( s = O(a) \) nonempty entries. For simplicity, we represent an empty entry by 0 and a nonempty one by 1, i.e. we suppose that \( A \) is a boolean matrix. We put the rows of \( A \), one after the other, into a one-dimensional array \( C \) such that no two nonzeros of \( A \) are mapped into the same position in \( C \). The start position \( r(i) \) of the row \( i \) is called the displacement of row \( i \). If \( A[i, j] \) is a nonzero, then it is mapped into \( C[r(i) + j] \). The problem is to find the displacements \( r(i) \) for \( 1 \leq i \leq a \).

The solution of Tarjan-Yao is rather complex. First, they showed that a row displacement satisfying \( 0 \leq r(i) \leq a \) for every row \( i \) can be computed easily if \( A \) has the following harmonic decay property
\[ a(\ell) \leq \frac{a}{\ell + 1}, \quad \text{for any } \ell \geq 0, \]
where \( a(\ell) \) stands for the total number of nonzeros in rows with more than \( \ell \) nonzeros. If \( A \) does not satisfy the harmonic decay, it is transformed into a new matrix \( B \) by column displacements, mapping each position \( A[i, j] \) into \( B[i + c(j), j] \). Finding values \( c(j) \) such that \( B \) satisfies the harmonic decay property is actually the central part of the Tarjan-Yao method (and also the most complex). They showed that column displacements can be determined such that
\[ 0 \leq c(j) \leq 4 \cdot s \cdot \log \log s + 9.5 \cdot s. \]

With this method the access time is \( O(1) \), the storage is \( a \) for the column displacement, \( 4 \cdot s \cdot \log \log s + a + 9.5 \cdot s \) for the row displacement, and
6.3 Table Compression with Random Entries

The compression algorithm of Tarjan-Yao is deterministic and is analyzed in the worst case. But what happens if we are interested only in the average performance? The goal is to design a simple approach for table compression on the assumption that the \( n \) nonzero entries in the matrix \( A \) are uniformly distributed at random. For simplification, we suppose that \( A \) is a \( n \times n \) matrix. On these conditions, we look for suitable row displacements in order to compress \( A \) into an array \( T[0, \ldots, n - 1] \). In contrast to Tarjan-Yao, we store a row \( i \) of \( A \) in a circular way, i.e.

\[
A[i, j] \quad \text{corresponds to} \quad T[(rd(i) + j) \mod n],
\]

where the displacement \( rd(i) \) marks the start position of row \( i \), cf. Figure 6.1. Note that no column displacements are used. Therefore, the main problem is how to determine the values \( rd(i) \). To do this, we divide the rows of \( A \) into two groups \( R_{\geq 2} \) and \( R_{\leq 1} \). The first group \( R_{\geq 2} \) contains all rows with two or more nonzero entries and \( R_{\leq 1} \) contains the other ones. The displacements \( rd(i) \) are first computed for all \( i \in R_{\geq 2} \) (in arbitrary order). Let \( u_1, u_2, \ldots, u_n \) be a random permutation of \( \{0, \ldots, n - 1\} \) (uniformly distributed). Then the value \( rd(i) \) is given by the first number of the sequence \( u_1, u_2, \ldots, u_n \) allowing a placement of row \( i \) without any collision with earlier placed rows from \( R_{\geq 2} \). On condition that all the rows of \( R_{\geq 2} \) are placed, the rows in \( R_{\leq 1} \) are simply placed in the remaining gaps of \( T \). Hence, if we have finished successfully, then

\[
A[i, j] \neq 0 \quad \Rightarrow \quad T[(rd(i) + j) \mod n] = A[i, j].
\]

Similarly to Tarjan and Yao, we have to store the indices — or the keys like Fredman, Komlos, and Szemeredi — together with \( A[i, j] \) in order to identify the given entry. Apparently, the worst case access
Figure 6.1: Compression example. The rows of the input matrix (left) are shifted circularly according to the row displacement \( rd \), such that every column contains at most one nonzero entry (i.e. the rows are not overlapping). Then, the entry in column \( j \) is stored in \( T[j] \).

Time is \( O(1) \) and the additional storage requirement is given by the values \( rd(i), 0 \leq i < n \).

Now, the central question is whether we find indeed a collision free row displacement. The empirical answer is given in Figure 6.2. The diagram suggests that the probability to find such a placement tends to 1 for \( n \to \infty \). In the following we will motivate this experimental result.

At first, we have to consider the nonzero entries in \( T \). Of course, the algorithmic construction produces a special distribution of these entries which influences the probability \( S_n \) that a collision free displacement exists for all rows in \( R_{\geq 2} \). However, \( S_n \) does not differ significantly from \( \hat{S}_n \), where \( \hat{S}_n \) denotes the analogue probability assuming that the nonzeros in \( T \) are uniformly distributed at random whenever a new row is placed.

We examined this by an experimental estimation of \( S_n \) and \( \hat{S}_n \), for \( 5 < n < 60 \). Each value was determined by \( 10'000'000 \) trials. The results are plotted in Figures 6.3-6.4. We observe that \( \hat{S}_n \) is mostly a lower bound (Figure 6.4), and in general it is a good approximation. Therefore, in the following we will analyze \( \hat{S}_n \) instead of \( S_n \).

We consider the probability \( Q_k(\nu, n, \ell) \) that a given row \( i \) with \( k \) nonzero entries cannot be placed at any of \( \nu \) pairwise disjoint choices for \( rd(i) \) in a uniformly distributed table \( T \) of size \( n \) containing \( \ell \) nonzeros.
Two possible positions \( d_1 \) and \( d_2 \) for the row \( i \) of \( A \) are called disjoint if and only if for all \( j \in \{0, \ldots, n-1\} \) it holds

\[
A[i, (j + d_1) \mod n] = 0 \lor A[i, (j + d_2) \mod n] = 0,
\]

i.e. the row shifted by \( d_1 \) does not overlap with the row shifted by \( d_2 \). Let us now consider the subset \( J \subseteq \{0, 1, \ldots, n-1\} \) containing pairwise disjoint elements. Each nonzero in a row shifted by \( j, j \in J \), can overlap with at most \( k - 1 \) other shifts of that row. Since every set \( J \subseteq \{0, 1, \ldots, n-1\} \) of disjoint positions is associated to \( |J| \cdot k \) nonempty entries, there are at most \( |J| \cdot k^2 \) positions overlapping with elements of \( J \). It follows that any subset \( I \subseteq \{0, 1, \ldots, n-1\} \) contains at least

\[
|I| \left\lfloor \frac{|I|}{k^2} \right\rfloor
\]

pairwise disjoint positions. Therefore, \( Q_k(\lfloor n/k^2 \rfloor, n, \ell) \) is an upper bound for the probability that a row with \( k \) nonzero entries cannot be placed in a table with \( \ell \) nonzeros.
Figure 6.3: Comparison between the real failure probability $1 - S_n$ and the corresponding approximation $1 - \hat{S}_n$.

Figure 6.4: Difference between real and approximated success probability $\hat{S}_n - S_n$.

The probability $Q_k(\nu, n, \ell)$ can be determined by

\begin{equation}
Q_k(\nu, n, \ell) = \sum_{j=1}^{k} H(n, k, \ell, j) \cdot Q_k(\nu - 1, n - k, \ell - j)
\end{equation}
where

\[ H(B, W, b, w) \overset{\text{def}}{=} \frac{\binom{W}{w} \binom{B-W}{b-w}}{\binom{B}{b}}, \]

\( w = \max\{0, b + W - B\}, \ldots, \min\{W, b\} \), is the \textit{hypergeometric distribution} corresponding to the following urn model. There are \( B \) balls in the urn, \( W \) of which are white and \( B - W \) are black. If we choose \( b \) balls at random (without replacements), then (6.8) expresses the probability that we observe \( w \) white balls. In our case, the white balls corresponds to the nonzero entries in the row and the black ones to its zeros. Each picked ball corresponds to a (random) position of a nonzero entry in \( T \). Therefore, \( j \) corresponds to the number of collisions. Note that

\[ Q_k(1,n,\ell) = \sum_{j=1}^{k} \binom{k}{j} \cdot \frac{(n-k)^{\ell-j}}{\binom{n}{\ell}} \cdot Q_k(0,n-k,\ell-j) = 1 - \frac{(n-k)^{\ell}}{\binom{n}{\ell}} \]

is the probability that the row cannot be placed at the first tested position. On condition that the first position is unsuccessful, we test a second disjoint position. This time we know that some of the \( \ell \) nonzero entries in \( T \) are not among the \( k \) newly tested entries (since the positions are disjoint). Therefore, the conditional probability that the row cannot be placed at the second position is smaller than \( 1 - \frac{(n-k)^{\ell}}{\binom{n}{\ell}} \). Hence,

\[ Q_k(2,n,\ell) \leq \left( 1 - \frac{(n-k)^{\ell}}{\binom{n}{\ell}} \right)^2. \]

This argument iterated \( \nu \) times gives

\[ Q_k(\nu,n,\ell) \leq \left( 1 - \frac{(n-k)^{\ell}}{\binom{n}{\ell}} \right)^\nu. \]

which implies the following lemma.

\textbf{Lemma 6.1} The probability that a row with \( k \) nonzero entries cannot be placed at any of \( \nu \) disjoint positions in a table containing \( \ell \) nonzero entries satisfies

\[ Q_k(\nu,n,\ell) \leq \left( 1 - \left( 1 - \frac{\ell}{n-k} \right)^k \right)^\nu. \]
In order to apply (6.9), we need some facts about the relevant values of $k$ and $\ell$. First, note that the number of zeros in $T$ is greater than the number $\rho$ of rows containing exactly one nonzero. This number is again given by a hypergeometric distribution, which implies

$$E(\rho) = n \cdot \frac{n \cdot \left( \frac{n(n-1)}{n-1} \right)}{(n^2)} \geq n \cdot \left( 1 - \frac{1}{n+1} \right)^{n-1} \geq \frac{n}{e}.$$ 

Next, we have

$$\Pr(\ell \leq n - n/\sqrt{e} + n^{7/8}) \geq \frac{\ell - n/\rho}{n} = E(\rho) \geq \frac{n}{e} \Pr(n - \rho \leq n - n/\sqrt{e} + n^{7/8}).$$

Cantelli's inequality, together with $\text{Var}(\rho) = \Theta(n)$, cf. [20], implies

$$\Pr(\rho - E(\rho) < -n^{7/8}) = O(n^{1-2.7/8}) = O(n^{-3/4}).$$

Hence

$$(6.10) \quad \Pr(\ell \leq n - \frac{n}{\sqrt{e}} + n^{7/8}) \geq 1 - \Pr(\rho - E(\rho) < -n^{7/8}) \equiv 1 - O(n^{-3/4}).$$

Next, the expected number of rows containing more than $k$ nonzero entries is

$$\sum_{j>\ell} \frac{n \cdot \left( \begin{array}{c} n \\ j \end{array} \right) \cdot \left( \begin{array}{c} n(n-1) \\ n-j \end{array} \right)}{(n^2)} \leq \sum_{j>\ell} \frac{n \cdot O\left( \frac{1}{j!} \right)}{k!} \sum_{j>0} O\left( \frac{1}{j!} \right) = O\left( \frac{n}{k!} \right)$$

which approaches 0 if

$$k \geq \alpha \cdot \frac{\ln n}{\ln \ln n}, \quad \alpha > 1.$$ 

It follows that the maximal number of nonzero entries in a row is bounded by $O(\ln n / \ln \ln n)$ with high probability, cf. [3, 20]. Thus, it is sufficient to consider

$$k \leq \frac{1}{3} \ln n \equiv \hat{k}.$$
which holds also with probability $1 - O(n^{-3/4})$. Then, $k \leq \hat{k}$ applied to (6.6) implies the existence of at least

$$\hat{\nu} = \frac{n}{k^2} = 9 \cdot \frac{n}{(\ln n)^2}$$

pairwise disjoint positions for a possible row displacement.

For that reasons, the probability that a row cannot be placed – on conditions $\ell \leq \hat{\ell}$ and $k \leq \hat{k}$ – is asymptotically bounded by

$$Q_k(\hat{\nu}, n, \hat{\ell}) \leq \left(1 - \left(\frac{1}{e} \cdot (1 + o(1))\right)^{\frac{1}{3} \log n} \frac{n^{\frac{3}{2}}}{(\log n)^2}\right) = O\left(\frac{1}{n^2}\right)$$

which implies the following proposition.

**Proposition 6.2** The compression of a random $n \times n$ matrix containing $n$ nonzero entries fails with probability less than

$$n \cdot Q_k(\hat{\nu}, n, \hat{\ell}) + O\left(\frac{1}{n^{3/4}}\right) = O\left(\frac{1}{n^{3/4}}\right).$$

Now, we turn to the running time of our algorithm. To simplify the analysis, we assume that $\ell \leq \hat{\ell}$ (which can be tested in linear time). If $\ell > \hat{\ell}$, then we consider the algorithm as unsuccessful.

**Proposition 6.3** The compression is successful with high probability and can be executed in linear expected time $O(n)$.

**Proof** A row $i$ of the input matrix $A$ is stored as a linear list $L[i]$ where $j$ is added to $L[i]$ if and only if $A[i, j] \neq 0$. This representation is relevant for finding $rd(i)$, $i \in R_{=2}$. In order to test if a choice $rd(i) = u$ does not overlap with nonzero entries in $T$, we have to access $T[(u + j) \mod n]$ at most for every $j \in L[i]$. Hence, one position is tested in time $O(|L[i]|)$. Next, let $X_k$, $k = |L[i]|$, be the number of positions which are checked for finding $rd(i)$. We already know from (6.6) that among $X_k$ positions there exist at least $X_k/k^2$ which are disjoint. Therefore, from (6.9) we infer

$$\Pr(X_k > t) \leq Q_k\left(\lfloor \frac{t}{k^2} \rfloor, n, \hat{\ell}\right) \leq (1 - e^{-k})^{\lfloor \frac{t}{k^2} \rfloor}$$

and further

$$E(X_k) = \sum_{0 \leq t < n} \Pr(X_k > t) \leq \sum_{0 \leq t < n} (1 - e^{-k})^{\lfloor \frac{t}{k^2} \rfloor}.$$
Since for \( i = 0, 1, \ldots, k^2 - 1 \) we have

\[
\left\lfloor \frac{j \cdot k^2 + i}{k^2} \right\rfloor = j,
\]

we obtain

\[
\mathbb{E}(X_k) \leq \sum_{j \leq n/k^2} k^2 \cdot (1 - e^{-k})^j = O(k^2 \cdot e^k).
\]

The probability that a row contains \( k \) nonzero entries satisfies

\[
\Pr(|L[i]| = k) = O\left(\frac{1}{k!}\right),
\]

cf. [20]. The expected number of access operations needed to place the \( i \)-th row in \( R_{\geq 2} \), or to verify that this is impossible, is bounded by

\[
\sum_{k > 1} k \cdot \mathbb{E}(X_k) \cdot \Pr(|L[i]| = k) = O\left(\sum_{k > 1} k^3 \cdot e^k \cdot \frac{1}{k!}\right) = O(1).
\]

At the end, we can place all non-critical rows in \( R_{\leq 1} \) simply by filling the remaining gaps in \( T \). Then, the lists \( L[i], i = 0, \ldots, n - 1 \), representing the table \( A \) can be deleted. Thus, the compression is successful with high probability and can be executed in linear expected time.

In contrast to other algorithms with nice asymptotic properties, the performance of this method is excellent over the whole range of \( n \). Experimentally, the compression needs on average less than \( 5 \cdot n \) access operations on \( T \), cf. Figure 6.5.

Remark. This analysis explains also why other simple greedy compression schemes work well in practice – e.g. the ones of ZIEGLER [49], and CHANG ET AL. [7]. See also [18] for further discussions.

### 6.4 Tetris-Hashing

The preceding compression scheme can be used to design a hashing concept which we call **Tetris-Hashing**. Because of the ambivalence between hashing and table compression, this new concept corresponds also to a randomized version of the compression algorithm.
Let us describe Tetris-hashing. Recall that the keys $K$, with $|K| = n$, are defined as a subset of the universe 

$$U = \{0, \ldots, N - 1\}.$$ 

In a first step, we use the class of functions 

$$h_k(x) = (k \cdot x \mod p) \mod n^2, \quad 1 \leq k \leq p - 1,$$

where $p$ is a prime greater than $N$, in order to distribute $K$ over the range $\{0, 1, \ldots, n^2 - 1\}$ pseudo-randomly. This corresponds to the functions used by Fredman, Komlos, and Szemeredi with $q = n^2$. Then, $h_k$ is injective when restricted to $K$ for about one half of the possible values of $k$ (cf. Figures 6.10-6.13). To determine such a $k$ we compute the values $h_k(x)$ for each $x \in K$. We interpret $\{0, 1, \ldots, n^2 - 1\}$ as indices of the $n \times n$ matrix $A$ where $h_k(K)$ stands for the nonempty entries. The position $(i, j)$ in $A$ corresponds to the $s$ satisfying

$$i = \left\lfloor \frac{s}{n} \right\rfloor \quad \text{and} \quad j = (s \mod n).$$

The matrix $A$ is represented by a set of lists $L[i], i = 0, \ldots, n - 1$, which are computed in time $O(n)$. Then, by using bucket sort (relative to the
column number) we can test in time $O(n)$ whether $h_k$ is injective or not. After we have determined a suitable $k$ (by random selection), the second step begins, namely we apply our compression technique on $A$.

We repeat these two steps until the compression is successful. The whole construction leads to a perfect hash function $h$ with

$$\forall x \in \{0, \ldots, N - 1\}: \quad h(x) \overset{\text{def}}{=} (rd(i) + j) \mod n$$

where

$$i = \left\lfloor \frac{h_k(x)}{n} \right\rfloor \quad \text{and} \quad j = h_k(x) \mod n$$

and $x \in K$ if and only if $T[h(x)]$ contains $x$. Apparently, we have a worst case access time of $O(1)$, and we need an array of size $n$ to store the values of $rd(i)$. Since the keys correspond basically to the indices, we do not need to store the row number – in contrast to the compression scheme. The expected construction time is linear in $|K|$.

As we mentioned at the beginning of this section, Tetris-hashing can be understood as a randomized compression algorithm. In order to compress an arbitrary $s \times t$ matrix $U$ into an array of minimal size

- we map $U$ into a one dimensional field $U'$ (e.g. with the standard mapping $U[i,j] = U'[i \times t + j]$)
- we interpret the nonempty entries in $U'$ as a key set $K$, and
- we apply Tetris-hashing to $K$.

Tetris-hashing reaches the well known advantages of randomized algorithms. In particular, the probability that the compression fails is reduced significantly. We executed 400,000 experiments\(^2\) and the algorithm was always successful. However, if after a fixed number of compression-steps the algorithm is not yet successful, other (time expensive) compression schemes can be used. Furthermore, the performance of our algorithm does not depend significantly on the input characteristic. Although $h_k$ is far from being a perfect random generator, its quality is sufficient for practical purposes. In our experiments, we need in general less than 2 choices for $k$ (assuming that $h_k$ is injective) to compress the matrix successfully; see Figures 6.6-6.9. Similar observations can be made for the expected construction time. The compression

---

\(^2\)See p. 95 for an exact description of the experiments.
of \( h_k(K) \) need less than \( 9 \cdot |K| \) access operations on \( T \) (for random keys even less), see Figure 6.14. The number of \( k \) tried until \( h_k \) is injective when restricted to \( K \) is generally less than 3, see Figures 6.10-6.13.

### 6.5 Dynamic Tetris-Hashing

Tetris-Hashing can be easily extended to dynamic hashing (i.e. we allow additionally insertion and deletions of keys). This is not a great challenge since update operations can be realized with a small modification of the compression algorithm. The access operation remains unchanged relative to the static case. Since in the dynamic case minimality for the table size is not achieved, \( m \) will again denote the table size.

We maintain the linear lists \( L[i] \), containing the positions of the nonzero entries in row \( i \) during the whole execution time. The storage needed for \( L[i], 0 \leq i \leq m - 1 \), can be reduced by an old-fashioned implementation with two arrays \( head \) and \( next \). Let the key \( x \) be stored in \( T[s] \) with

\[
 s = (rd(\lfloor h_k(x)/m \rfloor) + h_k(x) \mod m) \mod m, \\
 =i \\
 =j
\]

then \( T[next[s]] \) contains the successor of \( j \) in \( L[i] \). The index position of the first element of \( L[i] \) is stored in \( head[i] \). Finally, the empty entries of \( T \) are also stored in a linear list \( zeros \) (which can also be implemented with a single array of size \( m + 1 \)).

The (actual) value of the table size \( m \) is determined in the procedure \( RehashAll(m, K) \) which basically solves the static problem for size \( m \), i.e. the keys are mapped into an \( m \times m \) matrix which is successively compressed into a table of size \( m \). To avoid degenerated cases, we choose always \( m \geq m_0 \) (e.g. \( m_0 = 10 \)). Between two calls of \( RehashAll \) the table size \( m \) remains fixed (like in DIETZFELBINGER, ET AL. [10]).

On these conditions, the deletion of \( x \in K \) can be realized in a straight-forward manner. The entries for \( x \) in \( T, head, \) and \( next \) are removed. Additionally, the released index position in \( T \) is added to the list \( zeros \). Since the mean value of \( |L[i]|, 0 \leq i \leq m - 1 \), is less than 1, this can be executed in expected time \( O(1) \). Further, if the number of keys becomes too small (i.e. \( |K| < \alpha \cdot m \)) then a \( RehashAll(\delta \cdot |K|, K) \) is executed, where \( \alpha \) and \( \delta \) are fixed and satisfy \( 0 < \alpha < 1/\delta < 1 \). This happens after at least \( (1/\delta - \alpha) \cdot n \) keys are removed. Therefore, the amortized expected time for delete operations remains constant.
Next, we consider the insert operation. To simplify the algorithm, in the static case the distinction between $R_{\leq 1}$ and $R_{>2}$ was made offline. Now this has to be done online: Whenever we try to place a row in $R_{>2}$, we ignore collisions with nonzero entries in $T$ belonging to rows in $R_{<1}$. Note that each nonempty entry $T[s]$ contains its key $x$ and belongs to a row in $R_{<1}$ if and only if

$$|L[h_k(x) \mod m]| = 1.$$  

This can be tested in time $O(1)$. The ignored nonzero entries – if they exist – are simply moved to some other gaps in the table. For this purpose, we maintain the linear list zeros containing the free positions in $T$. In order to avoid unnecessary execution time for insert operations, a zero entry in $T$ filled during a placement of a row in $R_{>2}$ is not removed immediately from zeros. This removal will be caught up during the search for free positions in the list and can be done in amortized time $O(1)$. Hence, the insertion of a row in $T$ becomes completely incremental without changing the asymptotic execution time.

So far, we have seen how to (re)place a row in a given position. But when a new key is added, we first test if $|h_k(K)| = |K|$, i.e. $h$ remains injective, then we try to (re)place the corresponding row. If $h$ is not injective or the row cannot be (re)placed, then the insert operation is realized as a $RehashAll(\delta \cdot |K|, K)$. The amortized expected time is constant, as shown in Figures 6.15-6.16. Additionally, there is a trade-off between memory and time complexity: If $\delta$ is small, then less memory is used but more accesses on $T$ are required for update operations. These results depend on the number of RehashAll. Figures 6.17-6.18 show that the RehashAll calls are mostly executed because the table is nearly full. The calls due to the non-injectivity of $h_k$ are significantly smaller (about 0).

### 6.6 Experiments

Figures 6.6-6.13 show the behavior of Tetris-hashing in the static case. We executed a simulation for several sizes of the key set

$$n = |K| = 50, 60, \ldots, 250$$

and $p = 62501$ (we have chosen $p$ prime and greater than $n^2$). For every point we executed 5'000 experiments. We considered different input
distributions:

- random keys, uniformly distributed in \( \{0, 1, \ldots, N - 1\} \), where \( N = p \),
- one block of size \( n \), i.e.
  \[
  K = \{i, i + 1 \pmod{N}, \ldots, i + n - 1 \pmod{N}\},
  \]
  where \( i \) is a random number, and
- random blocks of size \( s \) (\( s = 13 \) and \( s = 25 \)).

Note that the presence of blocks mostly has a negative influence on the probability of success.
Figure 6.6: Avg. number of compressions for random keys.

Figure 6.7: Avg. number of compressions for random blocks of size \( s = 13 \).

Figure 6.8: Avg. number of compressions for random blocks of size \( s = 25 \).

Figure 6.9: Avg. number of compressions for a random block of size \( s = n \).
6.6. Experiments

**Figure 6.10:** Avg. number of choices of $k$ until $h_k$ is injective for random keys.

**Figure 6.11:** Avg. number of choices of $k$ until $h_k$ is injective for random blocks of size $s = 13$.

**Figure 6.12:** Avg. number of choices of $k$ until $h_k$ is injective for random blocks of size $s = 25$.

**Figure 6.13:** Avg. number of choices of $k$ until $h_k$ is injective for a random block of size $s = n$. 
Figure 6.14: Average running time (cell comparisons) required for a compression step for the different input distributions.

Figures 6.15-6.18 show the average time (accesses on $T$) and the average number of RehashAll calls, respectively, required to increment the number of keys from 10 to 20, 30, ..., 210 for several values of $\delta = 4/3, 5/3, 6/3$. For each point, 1'000 experiments have been executed. We considered different input distributions:

- random keys, uniformly distributed in \{0, 1, ..., $N - 1$\}, where $N = p$ and

- one block of size $n$, i.e.

$$ K = \{i, i + 1 (\text{mod} N), \ldots, i + n - 1 (\text{mod} N)\}, $$

where $i$ is a random number.

Note that the negative influence of blocks observed in the static case is less pronounced in the dynamic case.
6.6. Experiments

Figure 6.15: Average running time (cell comparisons) required to increment the number of keys from 10 to 20, 30, \ldots, 210 for several values of $\delta = 4/3, 5/3, 6/3$ (random keys).

Figure 6.16: Average running time (cell comparisons) required to increment the number of keys from 10 to 20, 30, \ldots, 210 for several values of $\delta = 4/3, 5/3, 6/3$ (block keys).

Figure 6.17: Average number of RehashAll calls (random keys).

Figure 6.18: Average number of RehashAll calls (block keys).
6.7 A Related Problem

The placement process of a row with \( k \) nonzero entries into a table with \( \ell \) nonzero entries is closely related with pattern matching. In fact, the row can be interpreted as a (circular) pattern \( P \). The table \( T \) corresponds to the (circular) text in which we search an occurrence of \( P \) (but, in contrast to the traditional pattern matching, for every one in \( P \) we look for a zero in \( T \)). Hence, the probability that the row can be placed corresponds to the probability that \( P \) occurs in \( T \).

In the preceding sections we assumed that \( T \) contains \( \ell \) nonzero entries uniformly distributed at random. To simplify the arguments, we will now consider another model. We assume that

\[
T[i] = \begin{cases} 
1 & \text{with probability } p \\
0 & \text{with probability } 1 - p 
\end{cases}
\]

independently of the other entries. The pattern \( P \) contains \( k \) nonzeros in arbitrary position. In the following,

\[
\Pr(P \in T)
\]

will denote the probability that \( P \) occurs in \( T \). The text \( T \) and the pattern \( P \) have the same length, denoted by \( n \). As in the previous sections, we will say that the row can be placed at position \( d \) if the row shifted by \( d \) does not overlap with any one in \( T \). We will say also that \( P \) occurs in \( T \) at position \( d \).

**Lemma 6.4** Let \( P \) and \( T \) be a pattern and a text, respectively, as described above. Then

\[
\Pr(P \in T) \geq 1 - \left(1 - p^k\right)^\left\lfloor \frac{n}{k} \right\rfloor
\]

**Proof** The arguments are identical as for Lemma 6.1. There are at least \( \left\lfloor \frac{n}{k} \right\rfloor \) disjoint positions in which the pattern \( P \) can occur. The probability that \( P \) occur in none of them is \( \left(1 - p^k\right)^\left\lfloor \frac{n}{k} \right\rfloor \).

The probability that \( P \) occurs in \( T \) depends on the positions of the ones in \( P \). Assume that \( k \) divides \( n \). Then let \( P_{\text{per}} \) be a periodic pattern, i.e. the \( k \) ones are placed with regular distances. Formally,

\[
P_{\text{per}}[i] = 1 \iff i \mod \frac{n}{k} = 0.
\]
6.7. A Related Problem

Let $P_{\text{run}}$ be a pattern in which all ones are consecutive, namely

$$P_{\text{run}}[i] = 1 \Leftrightarrow i \in \{0, 1, \ldots, k - 1\}.$$  

It is easy to see that

$$\Pr(P_{\text{per}} \in T) = 1 - \left(1 - p^k\right)^{\frac{n}{k}},$$

since there are $\frac{n}{k}$ disjoint positions and two positions which are not disjoint are identical. For the other pattern, there are $\frac{n}{k}$ disjoint positions, e.g. $D = \{0, k, 2k, \ldots, n-k\}$. But this time, there are further positions, each overlapping with two positions in $D$, which are not automatically excluded whenever positions in $D$ are excluded. Therefore,

$$\Pr(P_{\text{run}} \in T) \geq \Pr(P_{\text{per}} \in T).$$

In general, it seems that regular pattern are more difficult to be placed. But what does it mean regular? To answer, we introduce a concept well known in pattern matching: The autocorrelation [15, 22].

**Definition 6.5** Let $P$ be a pattern with $k$ ones. Let $\vartheta_i$ denote the number of ones that $P$ and a copy of $P$ rotated by $i$ have in common, namely

$$\vartheta_i \equiv \vartheta_i(P) \overset{\text{def}}{=} |\{j \mid P[j] = 1 \land P[(i + j) \mod n] = 1\}|.$$

Then, the autocorrelation of $P$ is the polynomial

$$A_P(x) \overset{\text{def}}{=} \sum_{i=0}^{n-1} x^{\vartheta_i}.$$  

The autocorrelation of a pattern is closely related with the expected number of occurrences of $P$ in $T$. Let

$$I_i(P; T) \equiv I_i(P) \overset{\text{def}}{=} \begin{cases} 1 & : \text{if } P \text{ occurs at position } i \\ 0 & : \text{otherwise} \end{cases}$$

and

$$X(P; T) \equiv X(P) \overset{\text{def}}{=} \sum_{i=0}^{n-1} I_i(P)$$

the number of occurrences of $P$ in $T$. 

Theorem 6.6

\[(6.11) \quad \Pr(P \in T) = \Pr(X(P) > 0),\]
\[(6.12) \quad \mathbb{E}(X(P)) = n \cdot p^k,\]
and
\[(6.13) \quad \mathbb{E}(X(P) \mid I_0(P) = 1) = p^k \cdot A_P(1/p).\]

**Proof** The pattern $P$ occurs at position $i$ with probability $p^k$ (i.e. if the entries corresponding to the $k$ nonzeros in $P$ are empty in $T$). Therefore, due to the linearity of expectation we obtain

\[\mathbb{E}(X(P)) = \sum_{i=0}^{n-1} \Pr(I_i(P) = 1) = n \cdot p^k.\]

If $P$ occurs at position 0, then the probability that $P$ occurs also at position $i$ is $p^{k-\vartheta_i}$, since $\vartheta_i$ of the $k$ entries considered are surely empty. Therefore

\[\mathbb{E}(X(P) \mid I_0(P) = 1) = p^k \cdot \sum_{i=0}^{n-1} \frac{1}{p^{\vartheta_i}},\]

which shows (6.13).

Consider two pattern $P_1$ and $P_2$, both with $k$ nonzeros. Assume

\[(6.14) \quad A_{P_1}(1/p) \geq A_{P_2}(1/p),\]

i.e. $P_1$ is more autocorrelated than $P_2$ with respect to $p$. Then, from (6.13) we get

\[\mathbb{E}(X(P_1) \mid I_0(P_1) = 1) \geq \mathbb{E}(X(P_2) \mid I_0(P_2) = 1)\]

In words, an occurrence of $P_1$ in $T$ (at a given position) increases the number of further occurrences more than $P_2$ does. Since the table is considered circularly, it is not relevant where the pattern occurs. This suggests

\[\mathbb{E}(X(P_1) \mid X(P_1) > 0) \geq \mathbb{E}(X(P_2) \mid X(P_2) > 0)\]

which together with (6.11), (6.12), and

\[\mathbb{E}(X(P_j)) = \mathbb{E}(X(P_j) \mid X(P_j) > 0) \cdot \Pr(X(P_j) > 0)\]

for $j = 1, 2$ implies

\[(6.15) \quad \Pr(P_1 \in T) \leq \Pr(P_2 \in T).\]
Lemma 6.7 Assume that \( k \) divides \( n \). Let \( P_{\text{per}} \) be the periodic pattern with \( k \) nonzeros, and let \( P \) be some pattern with \( k \) nonzeros. Then

\[
A_{P_{\text{per}}}(x) \geq A_P(x)
\]

for any \( x \geq 1 \).

Proof Let

\[
A_P(x) = \sum_{i=0}^{n-1} x^{\theta_i},
\]

be the autocorrelation of \( P \). Assume without loss of generality

\[
k \geq \vartheta_0 \geq \vartheta_1 \geq \cdots \geq \vartheta_{n-1}.
\]

Consider the following operation:

- choose \( u < k : \vartheta_u < k \), and
- choose \( v \geq k : \vartheta_v > 0 \), if possible.

Then define

\[
\vartheta^*_i = \begin{cases} 
\vartheta_i & : i \neq u, v, \\
\vartheta_i + 1 & : i = u, \\
\vartheta_i - 1 & : i = v.
\end{cases}
\]

For any \( a \geq b \geq 0 \) and \( x \geq 1 \) we have

\[
x^a + x^{b+1} \leq x^{a+1} + x^b
\]

which together with \( u < v \) and, by assumption, \( \vartheta_u \geq \vartheta_v \) implies

\[
\sum_{i=0}^{n-1} x^{\vartheta^*_i} = \sum_{i \neq u, v} x^{\vartheta_i} + (x^{\vartheta_u+1} + x^{\vartheta_v-1}) \geq \sum_{i=0}^{n-1} x^{\vartheta_i}.
\]

This step is repeated for \( \vartheta^* = (\vartheta^*_0, \ldots, \vartheta^*_{n-1}) \) until no suitable pair \( u, v \) exists. At the end, we have transformed \( \vartheta \) into a new vector \( \vartheta^* \) such that

\[
\sum_{i=0}^{n-1} x^{\vartheta^*_i} \geq \sum_{i=0}^{n-1} x^{\vartheta_i}.
\]
Furthermore, since each of the \( k \) nonzero entries shifted by \( 0, \ldots, n - 1 \) collides \( k \) times with some nonzero entry, we have

\[
\sum_{i=0}^{n-1} \mathcal{V}_i^* = k^2
\]

which implies

\[
\forall u < k : \quad \mathcal{V}_u^* = k \quad \Leftrightarrow \quad \forall v \geq k : \quad \mathcal{V}_v^* = 0.
\]

Hence, at the end

\[
\mathcal{V}_i^* = \begin{cases} 
  k & : i < k, \\
  0 & : i \geq k.
\end{cases}
\]

From

\[
\sum_{i=0}^{n-1} x^{\mathcal{V}_i^*} = k \cdot x^k + (n - k) = A_{P_{\text{per}}}(x)
\]

we get

\[
A_{P_{\text{per}}}(x) \geq A_P(x)
\]

for any \( x \geq 1 \).

This means that \( P_{\text{per}} \) is the most autocorrelated pattern, independently of \( p \), which leads to the following conjecture.

**Conjecture** If \( k \) divides \( n \), then

\[
(6.16) \quad \Pr(P \in T) \geq \Pr(P_{\text{per}} \in T) \geq 1 - \left(1 - p^k\right)^\frac{k}{x}
\]

and, in general,

\[
(6.17) \quad \Pr(P \in T) \geq 1 - \left(1 - p^b\right)^\left\lfloor \frac{x}{k} \right\rfloor.
\]

Furthermore, these observations explain why the presence of blocks in the key set has a negative influence on the performance of Tetris-hashing. Let \( a, a + 1, b, b + 1 \) be four keys. Then, they are mapped into a matrix according to

\[
h_s(x) = (s \cdot x) \mod N \mod n^2
\]
where $s$ is a random number in $\{1, \ldots, N-1\}$ and $N$ is the size of the universe. Assume that $a$ and $b$ are mapped in the same row of the matrix and let
\[
c \overset{\text{def}}{=} h_s(a) - h_s(b).
\]
Then, it is rather probable that also $a+1$ and $b+1$ are mapped in the same row and
\[
h_s(a+1) - h_s(b+1) = c.
\]
In other words, the matrix contains regularities which will be propagated in the table after the corresponding rows are placed. This increases the autocorrelation of the table and, consequently, it reduces the probability of success. If the key set contains blocks of consecutive keys, this phenomenon becomes highly probable. However, it can be avoided by modifying the function $h_s$. In fact, by using a quadratic function
\[
h_{s,t} \overset{\text{def}}{=} (s \cdot x^2 + t \cdot x) \mod N \mod n^2
\]
this kind of correlations are significantly decreased. Experimentally, the performance of Tetris-hashing for random key sets and block key sets, mapped with quadratic functions, are nearly identical.

### 6.8 Conclusion

We have presented a simple Monte Carlo algorithm for optimal table compression. Its randomized version has been used to design a perfect hashing scheme. The algorithms can be improved in various ways. For instance, the success probability of the compression can be increased by sorting the rows of the matrix by the number of nonzero entries. Then, the probability of success is already 1 for $n > 100$. The influence on the performance of Tetris-hashing of the input distribution of the key set can be reduced by using appropriate pseudo-random generators.

Several questions remain open in the theoretical analysis, especially the ones related to the distribution of the table during the compression. Experimentally, the difference between $S_n$ and $\hat{S}_n$ is nearly undetectable which justifies that we replaced $S_n$ by $\hat{S}_n$ in the mathematical analysis. However, it is not clear if $S_n \geq \hat{S}_n$ (at least for $n$ large). The reason is that $\hat{S}_n - S_n$ approaches 0 for large values of $n$ which constitutes a serious problem for the experimental measurements.
The discussions in the previous section suggest an optimistic answer. We know that the table $T$ is not uniformly distributed. Indeed, the placement of the rows tends to avoid configurations which are highly autocorrelated. These configurations are also the ones which mostly reduce the probability of success.
Chapter 7

Conclusion

Average case analysis has shown to be a fruitful tool to investigate the efficiency of algorithms. In particular, an accurate analysis can lead to improvements of the solutions or, if not possible, it can show the corresponding limits (in form of lower bounds). Furthermore, the interpretation as stochastic processes allows us to derive alternative, elegant proofs (e.g. algorithmic proofs) for several problems.

During the analysis, the processes are simplified: They become so essential that their practical interest seems limited. However, the reached abstraction facilitates the generalization of the problems (models). Moreover, the processes become interesting independently of the algorithmic aspects because they possibly reveal new amazing problems in other areas.

In this thesis, we illustrated this with some examples related to search problems. This work can be continued in several directions.

- The disintegration process introduced for the analysis of BFS can be used to investigate other problems, e.g. boolean matrix multiplication. SCHNORR [42] proposed a strategy to multiply two boolean matrices in expected time $O(n^2 \log n)$ assuming that the ones in each column of the left matrix are distributed at random. It is possible to reduce this problem to a disintegration process. This model is rather optimistic. Therefore, the algorithm should be extended and adapted to other input models. However, the similarity with the disintegration process seems to be an impor-
tant aspect of this problem.

- The non-homogeneous random graphs defined in Chapter 4 show new perspectives in the theory of random graphs – in fact, all traditional problems can be reconsidered in this model, e.g. a possible threshold for connectedness, Hamiltonian cycles, maximal cliques, etc. Note that the analysis of SSSP and MST can be extended to non-homogeneous random graphs on condition that $p$ is chosen in such a way that all nodes are reachable from the starting point with high probability (e.g. if $G_{n,p}$ is defined according to Lemma 4.14).

- The open problems in pattern matching connected to hashing (see Chapter 6) represent a challenging research theme – in particular, the relationship between the occurrence probability and the autocorrelation of a pattern.
Bibliography


Curriculum Vitæ

Name:       Nicola GALLI
Date of Birth:  28th of May 1971
Place of Birth:  Besazio
Nationality:  Swiss

1977-1982       Scuola Elementare (Arzo)
1982-1986       Scuola Media (Mendrisio)
1986-1990       Liceo Cantonale (Mendrisio)
                Tipo A
1990-1995       ETH (Zurich)
                Dipl. Informatik-Ing. ETH
1995           Industriepraktikum
                (Fides Informatik, Zurich)
1995-1998      PhD student (ETHZ)