COMPUTER PROGRAMMING IS AN ART, BECAUSE IT APPLIES ACCUMULATED KNOWLEDGE TO THE WORLD, BECAUSE IT REQUIRES SKILL AND INGENUITY, AND ESPECIALLY BECAUSE IT PRODUCES OBJECTS OF BEAUTY. A PROGRAMMER WHO SUBCONSCIOUSLY VIEWS HIMSELF AS AN ARTIST WILL ENJOY WHAT HE DOES AND WILL DO IT BETTER.

DONALD E. KNUTH

TO ME THE VERY ESSENCE OF EDUCATION IS CONCENTRATION OF MIND, NOT THE COLLECTION OF FACTS.

SWAMI VIVEKANANDA

. . . THE DESIGNER OF A NEW SYSTEM MUST NOT ONLY BE THE IMPLEMENTOR AND THE FIRST LARGE-SCALE USER; THE DESIGNER SHOULD ALSO WRITE THE FIRST USER MANUAL. . . IF I HAD NOT PARTICIPATED FULLY IN ALL THESE ACTIVITIES, LITERALLY HUNDREDS OF IMPROVEMENTS WOULD NEVER HAVE BEEN MADE, BECAUSE I WOULD NEVER HAVE THOUGHT OF THEM OR PERCEIVED WHY THEY WERE IMPORTANT.

DONALD E. KNUTH

ANIL MAHESHWARI

# NOTES ON ALGORITHM DESIGN

PUBLISHED BY USING THE STYLE FILES FROM THE TUFTE-LATEX DEVELOPERS



*First printing, August 2017*

# Contents

# *List of Figures*

# Preface

These notes extensively use material from the course notes of Lars Arge, David Mount, COMP 2805/3803 Notes of myself and Michiel Smid [1], CLRS book [2], Knuth's Art of Computer Programming [3], Kleinberg and Tardos Algorithms book [4], Leskovec, Rajaram and Ullman's book on Algorithms for Massive Data Sets [5]. These notes are updated occasionally. A substantial update was done in the Fall Term of 2013. Chapters on elementary probability, locality sensitive hashing, dimensionality reduction, and several exercises have been added. Moreover, as part of the offering of COMP 5703 in the Fall of 2013, several students have contributed significantly. Gregory Bint updated the chapter on the Minimum Spanning Trees, and has added a new section on spanning tree verification. Alexis Beingessner has provided a section on extension of the planar separator theorem. Andrew Wylie has updated the chapter on Locality Sensitive Hashing. He also added a section on image similarities and provided an extensive bibliography on locality sensitive hashing and its applications. Michael St. Jules has provided an entire chapter on Stable Matchings. I thank the Fall 2013 batch of COMP 5703 for providing valuable contributions. In Fall 2015 term I started to work on the chapter on Second Moment Method. The addition of this chapter was inspired by a comment from one of the referees of our paper [6] in ALGOSENSORS 2015, where s/he mentioned that the paper is a good introduction to the Second Moment Method at graduate level. So I have pasted the whole paper in that chapter and added a section on Cliques. Over time, hopefully, this chapter will evolve. Starting in Summer 2017, I plan to use a new style file from the Tufte-LaTeX Developers to modernize parts of these notes. I am also planning to update a few chapters and add more exercises.

I have used parts of this material for the graduate course COMP 5703 at Carleton. The aim of these notes is mainly to summarize the discussion in the lectures. They are not designed as stand alone chapters or a comprehensive coverage of a topic. The exercises are from numerous sources and most of them have been asked in either an exam or in an assignment. Since these chapters have been written over time, and many of the TeX tools weren't available in olden times, you will see that the initial chapters don't have elegant figures or texts. Hopefully, volunteers in the future will modernize this part.

If you spot any errors, or have suggestion which can help me to improve, I will be glad to hear them. If you wish to add material to these notes, including exercises, please do get in touch. Thanks in advance!

[1] A. Maheshwari and M. Smid. *Introduction to Theory of Computation*. Free Online, 2012

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009

[3] Donald E. Knuth. *The art of computer programming, volume 1-3*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998

[4] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005

[5] Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2011

[6] Ahmad Biniaz, Evangelos Kranakis, Anil Maheshwari, and Michiel Smid. Plane and planarity thresholds for random geometric graphs. In *Proc. ALGOSENSORS 2015 (Patras, Greece)*, Lecture Notes in Computer Science, Berlin, Germany, 2015. Springer

Anil Maheshwari (anil@scs.carleton.ca)

# 1
# *Preliminaries*

## *1.1 Introduction*

- This course is about *designing* and *analyzing algorithms*
  - *Algorithm*:
    - Mis-spelled *logarithm!*.
    - The first most popular algorithm is the Euclid's algorithm for computing the GCD of two numbers.
    - A well-defined procedure that transfers an input to an output.
    - Not a program (but often specified like it): An algorithm can often be implemented in several ways.
    - Knuth's, Art of Computer Programming, vol.1, is a good resource on the history of algorithms!. He says that an *algorithm* is a finite set of rules that gives a sequence of operations for solving a specific type of problem. Algorithm has five important features:
      *Finiteness:* must terminate after finite number of steps.
      *Definiteness:* each step is precisely described.
      *Input:* algorithm has zero or more inputs.
      *Output:* has at least one output!.
      *Effectiveness:* Each operation should be sufficiently basic such that they can be done in finite amount of time using pencil and paper.
  - *Design*: The focus of this course is on how to design good algorithms and how to analyze their efficiency. We will study methods/ideas/tricks for developing fast and efficient algorithms.
  - *Analysis*: Abstract/mathematical comparison of algorithms (without actually implementing, prototyping and testing them).

- This course will require proving the correctness of algorithms and analyzing the algorithms. Therefore MATH is the main tool. Math is needed in three ways:
  - Formal specification of problem
  - Analysis of correctness
  - Analysis of efficiency (time, memory use,...)

  Please review mathematical induction, what is a proof?, logarithms, sum of series, elementary number theory, permutations, factorials, binomial coefficients, harmonic numbers, Fibonacci numbers and generating functions [Knuth vol 1. or his book Concrete Mathematics is an excellent resource].
- Hopefully the course will show that **algorithms matter!**

## 1.2 *Model of Computation*

- Predict the resources used by the algorithm: running time and the space.
- To analyze the running time of an algorithm, we need a mathematical model of a computer:

  *Random-access machine (RAM) model*:
  - Memory consists of an infinite array of cells.
  - Each cell can store at most one data item (bit, byte, a record, ..).
  - Any memory cell can be accessed in unit time.
  - Instructions are executed sequentially
  - All basic instructions take unit time:
    - Load/Store
    - Arithmetic's (e.g. $+, -, *, /$)
    - Logic (e.g. $>$)
- *Running time* of an algorithm is the number of RAM instructions it executes.
- RAM model is not realistic, e.g.
  - memory is finite (even though we often imagine it to be infinite when we program)
  - not all memory accesses take the same time (cache, main memory, disk)

  - not all arithmetic operations take the same time (e.g. multiplications are expensive)
  - instruction pipelining
  - other processes
- But RAM model often is enough to give relatively realistic results (if we don't cheat too much).

## 1.3 *Asymptotics*

We do not want to compute a detailed expression of the run time of the algorithm, but rather will like to get a feel of what it is like? We will like to see the trend - i.e. how does it increase when the size of the input is increased - is it linear in the size of the input? or quadratic? or exponential? or who knows? The asymptotics essentially capture the rate of growth of the underlying functions describing the run-time. Asymptotic analysis assumes that the input size is large (since we are interested how the running time increases when the problem size grows) and ignores the constant factors (which are usually dependent on the hardware, programming smartness or tricks, compile-time-optimizations).

David Mount suggests the following simple definitions based on the limits for functions describing the running time of algorithms. We will describe the formal definitions later[1].

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009

Let $f(n)$ and $g(n)$ be two positive functions of $n$. What does it mean when we say that both $f$ and $g$ grow at roughly the same rate for large $n$ (ignoring the constant factors), i.e.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c,$$

where $c$ is a constant and is neither 0 or $\infty$. We say that $f(n) \in \Theta(g(n))$, i.e. they are asymptotically equivalent. What about $f(n)$ does not grow significantly faster than $g(n)$ or grows significantly faster? Here is the table of definitions from David Mount.

| Asymptotic Form | Relationship | Definition |
|---|---|---|
| $f(n) \in \Theta(g(n))$ | $f(n) \equiv g(n)$ | $0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$ |
| $f(n) \in O(g(n))$ | $f(n) \leq g(n)$ | $0 \leq \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$ |
| $f(n) \in \Omega(g(n))$ | $f(n) \geq g(n)$ | $0 < \lim_{n\to\infty} \frac{f(n)}{g(n)}$ |
| $f(n) \in o(g(n))$ | $f(n) < g(n)$ | $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ |
| $f(n) \in \omega(g(n))$ | $f(n) > g(n)$ | $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ |

Example: $T(n) = \sum_{x=1}^{n} x^2 \in \Theta(n^3)$. Why?

$$\lim_{n\to\infty} \frac{T(n)}{n^3} = \lim_{n\to\infty} \frac{(n^3 + 3n^2 + 2n)/6}{n^3} = 1/6,$$

and $0 < 1/6 < \infty$.

Just for fun show that $T(n) \in O(n^4)$ or $T(n) = n^3/3 + O(n^2)$.

### 1.3.1 O-notation

$O(g(n)) = \{f(n) : \exists\, c, n_0 > 0$ such that $f(n) \leq cg(n),\ \forall n \geq n_0\}$

- $O(\cdot)$ is used to asymptotically *upper bound* a function.
- $O(\cdot)$ is used to bound *worst-case* running time (see Figure 1.1).

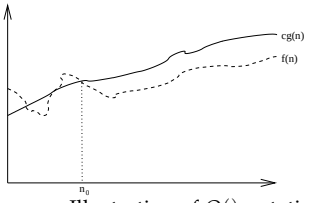


Figure 1.1: Illustration of $O()$ notation.

- Examples:
  - $1/3n^2 - 3n \in O(n^2)$ because $1/3n^2 - 3n \leq cn^2$ if $c \geq 1/3 - 3/n$ which holds for $c = 1/3$ and $n > 1$.
  - Let $p(n) = \sum_{i=0}^{d} a_i n^i$ be a polynomial of degree $d$ and assume that $a_d > 0$. Then $p(n) \in O(n^k)$, where $k \geq d$ is a constant. What are $c$ and $n_0$ for this?
- Note:
  - When we say "the running time is $O(n^2)$", we mean that the *worst-case* running time is $O(n^2)$ — best case might be better.
  - We often abuse the notation:
    - We write $f(n) = O(g(n))$ instead of $f(n) \in O(g(n))$.
    - We often use $O(n)$ in equations: e.g. $2n^2 + 3n + 1 = 2n^2 + O(n)$ (meaning that $2n^2 + 3n + 1 = 2n^2 + f(n)$ where $f(n)$ is some function in $O(n)$).
    - We use $O(1)$ to denote a constant.

### 1.3.2 Ω-notation (big-Omega)

$\Omega(g(n)) = \{f(n) : \exists\, c, n_0 > 0$ such that $cg(n) \leq f(n),\ \forall n \geq n_0\}$

- $\Omega(\cdot)$ is used to asymptotically *lower bound* a function (see Figure 1.2).

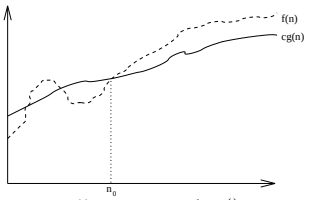


Figure 1.2: Illustration of $\Omega()$ notation.

- Examples:
  - $1/3n^2 - 3n = \Omega(n^2)$ because $1/3n^2 - 3n \geq cn^2$ if $c \leq 1/3 - 3/n$ which is true if $c = 1/6$ and $n > 18$.
  - Let $p(n) = \sum_{i=0}^{d} a_i n^i$ be a polynomial of degree $d$ and assume that $a_d > 0$. Then $p(n) \in \Omega(n^k)$, where $k \leq d$ is a constant. What are $c$ and $n_0$ for this?

  - Prove or disprove: $g(n) = \Omega(f(n))$ if and only if $f(n) = O(g(n))$.

- Note:
  - When we say "the running time is $\Omega(n^2)$", we mean that the *best case* running time is $\Omega(n^2)$ — the worst case might be worse.

### 1.3.3 $\Theta$-notation (Big-Theta)

$\Theta(g(n)) = \{f(n) : \exists\ c_1, c_2, n_0 > 0$ such that $c_1 g(n) \leq f(n) \leq c_2 g(n)\ \forall n \geq n_0\}$

- $\Theta(\cdot)$ is used to asymptotically *tight bound* a function.

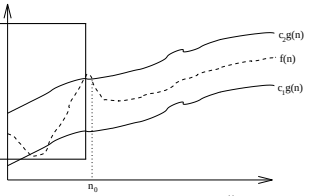


Figure 1.3: Illustration of $\Theta()$ notation.

$f(n) = \Theta(g(n))$ *if and only if* $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ (see Figure 1.3)

- Examples:
  - $6n \log n + \sqrt{n} \log^2 n = \Theta(n \log n)$:
    - We need to find $n_0, c_1, c_2$ such that $c_1 n \log n \leq 6n \log n + \sqrt{n} \log^2 n \leq c_2 n \log n$ for $n > n_0$ $c_1 n \log n \leq 6n \log n + \sqrt{n} \log^2 n \Rightarrow c_1 \leq 6 + \frac{\log n}{\sqrt{n}}$. Ok if we choose $c_1 = 6$ and $n_0 = 1$. $6n \log n + \sqrt{n} \log^2 n \leq c_2 n \log n \Rightarrow 6 + \frac{\log n}{\sqrt{n}} \leq c_2$. Is it ok to choose $c_2 = 7$? Yes, $\log n \leq \sqrt{n}$ if $n \geq 2$.
    - So $c_1 = 6$, $c_2 = 7$ and $n_0 = 2$ works.
  - Let $p(n) = \sum_{i=0}^{d} a_i n^i$ be a polynomial of degree $d$ and assume that $a_d > 0$. Then $p(n) \in \Theta(n^k)$, where $k = d$ is a constant.

## 1.4 How to analyze Recurrences?

There are many ways of solving recurrences. I personally prefer the recursion tree method, since it is visual! Here the recurrence is depicted in a tree, where the nodes of the tree represent the cost incurred at the various levels of the recursion. We illustrate this method using the following recurrence (so called the recurrence used in the Masters method).

Let $a \geq 1, b > 1$ and $c > 0$ be constants and let $T(n)$ be the recurrence

$$T(n) = aT\left(\frac{n}{b}\right) + cn^k,$$

defined for integer $n \geq 0$. Then

*Case 1* $a > b^k$ then $T(n) = \Theta(n^{\log_b a})$.

*Case 2* $a = b^k$ then $T(n) = \Theta(n^k \log_b n)$.

*Case 3* $a < b^k$ then $T(n) = \Theta(n^k)$.

The proof is fairly simple. We need to visualize the levels of the underlying recursion tree.

*Level 1:* $a$ subproblems are formed, each of size $n/b$, and the total cost is $cn^k$.

*Level 2:* $a^2$ subproblems are formed, each of size $n/b^2$, and the total cost is $a * c(n/b)^k$.

*Level 3:* $a^3$ subproblems are formed, each of size $n/b^3$, and the total cost is $a^2 * c(n/b^2)^k$.

...

*Level* $\log_b n$: $a^{\log_b n}$ subproblems are formed, each of constant size and the total cost is about $a^{\log_b n} c(\frac{n}{b^{\log_b n}})^k$.

Therefore the total cost is

$$T(n) = O(n^{\log_b a}) + \sum_{i=0}^{\log_b n} a^i c \left(\frac{n}{b^i}\right)^k.$$

Now apply the various cases.

Note that you cannot analyse all types of recurrences using the above method. For example, consider the following recurrence

$$T(n) = T(n/3) + T(2n/3) + n.$$

We can assume $T(n) = O(1)$ for small values of $n$. This recurrence doesn't fit the format of recurrences discussed above. For these (or any) recurrences, we can try the substitution method. Here we guess a solution and verify that our guess is correct. Typically the complexity of an algorithm for a problem of size $n$ will be one of $\{O(\log n), O(n), O(n \log n), O(n^2), O(n^2 \log n), O(n^3), ...\}$, and hence using induction, we can try to see which one of these expressions work. For example, try to show that $T(n) = T(n/3) + T(2n/3) + n = O(n \log n)$[2].

[2] A solution can be found in T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009

## 1.5 Strassen's Matrix Multiplication

### 1.5.1 Matrix Multiplication

This is a classical example to illustrate the recurrences as well as the divide and conquer method. Consider Strassen's matrix multiplication method [3] as illustrated in the following. Let $A$, $B$ and $C$ be three $n \times n$ matrices, where $Z = X \cdot Y$. There are $n$ rows, $n$ columns and $n \times n$ entries in each of the matrices.

[3] D. Kozen. *The design and analysis of algorithms*. Springer, 1992; and V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969

- $X = \left\{ \begin{array}{cccc} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{1n} \\ \cdots & \cdots & \cdots & \cdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{array} \right\}$

- $Y = \left\{ \begin{array}{cccc} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{1n} \\ \cdots & \cdots & \cdots & \cdots \\ y_{n1} & y_{n2} & \cdots & y_{nn} \end{array} \right\}$

- We want to compute $Z = X \cdot Y$, where

$$z_{ij} = \sum_{k=1}^{n} x_{ik} \cdot y_{kj}.$$

- How many operations we require?

- In all we generate $n^2$ entries in the matrix $Z$ and each entry requires $n$ multiplications and $n-1$ additions. So the total number of operations can be bounded by $\Theta(n^3)$.

- Next we want to discuss a divide and conquer solution by Strassen which requires only $O(n^{\log_2 7})$ operations.

- Let's first analyze the recurrence

$$T(n) = 7T(n/2) + cn^2,$$

  where $c$ is a constant, $n$ is a positive integer, and $T(constant) = O(1)$.

- Using the simplified master method, $a = 7$, $b = 2$, $c = c$, $k = 2$ and $a > b^k$. Hence $T(n) = O(n^{\log_2 7})$.

### 1.5.2 *Strassen's Algorithm*

- Divide each of the matrices into four sub-matrices, each of dimension $n/2 \times n/2$. Strassen observed the following:

$$Z = \left\{ \begin{array}{cc} A & B \\ C & D \end{array} \right\} \cdot \left\{ \begin{array}{cc} E & F \\ G & H \end{array} \right\} = \left\{ \begin{array}{cc} (S_1 + S_2 - S_4 + S_6) & (S_4 + S_5) \\ (S_6 + S_7) & (S_2 + S_3 + S_5 - S_7) \end{array} \right\}$$

  where

$$
\begin{aligned}
S_1 &= (B-D)\cdot(G+H)\\
S_2 &= (A+D)\cdot(E+H)\\
S_3 &= (A-C)\cdot(E+F)\\
S_4 &= (A+B)\cdot H\\
S_5 &= A\cdot(F-H)\\
S_6 &= D\cdot(G-E)\\
S_7 &= (C+D)\cdot E
\end{aligned}
$$

  - Lets test that for $S_4 + S_5$, which is supposed to be $AF + BH$.

$$
\begin{aligned}
S_4 + S_5 &= (A+B)\cdot H + A\cdot(F-H)\\
&= AH + BH + AF - AH\\
&= AF + BH
\end{aligned}
$$

- This leads to a divide-and-conquer algorithm with running time $T(n) = 7T(n/2) + \Theta(n^2)$, since
  - We only need to perform 7 multiplications recursively. Additions/Subtractions only take $\Theta(n^2)$ time, and we need to do 18 of them for $n/2 \times n/2$ matrices for each step of the recursion.
  - Division/Combination can still be performed in $\Theta(n^2)$ time.

Matrix multiplication is a fundamental problem and it arises in almost all branches of Sciences, Social Sciences and Engineering. For example, high energy physicists multiply monstrous matrices. Strassen's is not the currently fastest known algorithm, there have been numerous improvements over that method. It is obvious that any algorithm for matrix multiplication needs to perform $\Omega(n^2)$ operations, since the output matrix $Z$ has that many entries. But only lower bound that is known for this problem is the trivial one, i.e. the $\Omega(n^2)$ bound. The currently best known upper bound is significantly larger than this (its about $O(n^{2.37\ldots})$). So a major open problem, whose solution will be of immense importance will be either to raise the lower bound or drop down the upper bound!

Here is something which may be interesting to look into. This is regarding verifying given three $n \times n$ matrices $A$, $B$ and $C$, whether $AB = C$? It turns out that there is a nice randomized algorithm that can do this and it is stated in the following Theorem (See Motwani and Raghavan [4] for details).

**Theorem 1.5.1** *Let $A$, $B$, and $C$ be $n \times n$ matrices over a Field $F$ such that $AB \neq C$. Then for $r$ chosen uniformly at random from $\{0,1\}^n$, probability that $Pr[ABr = Cr] \leq 1/2$.*

[4] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, New York, NY, USA, 1995

## 1.6 Exercises

**1.1** *Provide a proof for the simplified Masters theorem.*

**1.2** *Present examples for each of the three cases and present an example where this theorem is not applicable.*

**1.3** *This is from [23] and is based on Divide-and-Conquer Multiplication. (Do not use FFTs as such for this)*

1. *Show how to multiply two polynomials of degree 1, namely $ax + b$ and $cx + d$ using only three multiplications. (Note that $(a + b).(c + d)$ is considered as 1 multiplication.)*
2. *Give a divide-and-conquer algorithm for multiplying two polynomials of degree n that runs in $\Theta(n^{\log_2 3})$. You may think of dividing the coefficients into a high half and a low half, or in terms of whether the index is even or odd.*
3. *Show that two n-bit integers can be multiplied in $O(n^{\log_2 3})$ steps, where each step operates on at most a constant number of 1-bit values.*

**1.4** *Show that $1^d + 2^d + 3^d + .... + n^d$ is $O(n^{d+1})$ for a constant d? What is the value of c and $n_0$.*

**1.5** *Solve the recurrence relation*

$$T(n) = T(xn) + T((1 - x)n) + cn$$

*in terms of x and n where x is a constant in the range $0 < x < 1$. Is the asymptotic complexity the same when $x = 0.5, 0.1$ and $0.001$? What happens to the constant hidden in the $O()$ notation.*

**1.6** *Analyze the recurrence relation*

$$T(n) = \sqrt{n}T(\sqrt{n}) + n.$$

*(This is called as the rootish-divide-and-conquer and for example plays an important role in parallel computing.)*

**1.7** *V. Pan has discovered a way to multiply two $70 * 70$ matrices using only 143640 multiplications. Ignoring the additions, what will the asymptotic complexity of Pan's algorithm for multiplying two $n * n$ matrices? Is it better than Strassen's? Justify your answer.*

**1.8** *Suppose we have n real numbers, where no two are the same. We want to report the smallest i numbers in the sorted order, where $i < n$. Which algorithm you think is the best option (Justify your answer)?*

*A. Sort the numbers and list the first i.*

*B. Build a priority queue and then call Extract-Min i times.*

*C. Use the order statistics to find the i-th smallest number, partition the set according to this value, and then sort the i smallest numbers.*

**1.9** *Let $A$ and $B$ be two arrays, each consisting of $n$ distinct elements in sorted order (in an increasing order). Report the median of the set $A \cup B$ in $O(\log n)$ time.*

**1.10** *Given two binary strings $a = a_0 a_1 \dots a_p$ and $b = b_0 b_1 \dots b_q$, where each $a_i$ and $b_j$ are either 0 or 1. We say that $a \leq b$ if either of the following holds*

*(1) there exists an integer $j$, $1 \leq j \leq \min(p, q)$, such that $a_i = b_i$ for all $i = 0, 1, ..., j-1$ and $a_j < b_j$.*

*(2) $p < q$ and $a_i = b_i$ for all $i = 0, 1, 2, , , , p$.*

*Let $A \subseteq \Sigma^*$ be a set of distinct binary strings whose lengths sums up to $n$. Present an algorithm that can sort the binary strings in $O(n)$ time. (All the strings are not of the same length!)*

**1.11** *For any value of $x$, $0 < x < 1$, show that $x - (1+x)\ln(1+x) + \frac{1}{3}x^2 \leq 0$.*

# 2
# *Probability for CS*

This material is adapted from the book of Meyer [1]. (BTW, this was my textbook for the first undergraduate course in probability - way back in the Winter'83.)

[1] P.L. Meyer. *Introductory probability and statistical applications*. Addison-Wesley, Boston, MA, USA, 1970

## 2.1 *Basics*

**Sample Space and Events:** With each probabilistic experiment, the *sample space* is the set of all possible outcomes of that experiment. For example, for rolling a dice the set of all possible outcomes are $\{1,2,3,4,5,6\}$. An *event* is also a set of possible outcomes, and is a subset of the sample space. For example, for rolling a dice and getting an even number, the events are $\{2,4,6\}$. If the sample space consists of $n$ elements, then the total number of all possible events are $2^n$. Since events are sets, we can use associated operations on sets. For example, if $A$ and $B$ are events for a sample space $S$, then we can define $A \cup B$, $\bar{A}$, $A \cap B$, with the usual meaning. Two events $A$ and $B$ are said to be *mutually exclusive* if they cannot occur simultaneously, i.e. $A \cap B = \emptyset$.

**Probability:** Let $S$ be a sample space associated with an experiment. For each event $A \subseteq S$, associate a real number $0 \leq P(A) \leq 1$, called as the *probability* of $A$, and $P(A)$ satisfies following natural conditions

1. $P(S) = 1$,
2. If events $A$ and $B$ are mutually exclusive, $P(A \cup B) = P(A) + P(B)$ and $P(A) \cap P(B) = 0$.
3. $P(\bar{A}) = 1 - P(A)$, where $\bar{A} = S \setminus A$.
4. In general, for two events $A$ and $B$, $P(A \cup B) = P(A) + P(B) - P(A \cap B)$.
5. If $A \subset B$, $P(A) \leq P(B)$.

**Conditional Probability:** Let $A$ and $B$ be two events associated with an experiment. We say $P(B|A)$ to be the *conditional probability* of occurrence of event $B$ given that $A$ has occurred. Intuitively, computation of $P(B)$ is done with respect to a reduced sample space. For example, let $B$ be the event of getting 2 after rolling a dice. Then $P(B) = 1/6$. Let $A$ be the event of getting an even number after rolling a dice. Then $P(A) = 1/2$. What about $P(B|A)$? It is $1/3$, since the reduced sample space is $\{2, 4, 6\}$, and the probability of drawing a 2 is $1/3$ from this space. Note that $P(B|A) = \frac{P(A \cap B)}{P(A)}$ or equivalently $P(A \cap B) = P(B|A)P(A)$. Similarly we can define $P(A|B) = \frac{P(A \cap B)}{P(B)}$, or equivalently $P(A \cap B) = P(A|B)P(B)$. Note that if two events $A$ and $B$ are mutually-exclusive than $P(A|B) = P(B|A) = 0$. If $A \subset B$, than $P(B|A) = 1$. (For example, if $B$ is the event of obtaining an even number and $A$ is the event of getting a 2 on rolling a dice, than if $A$ has occurred, than for sure $B$ has occurred.) Now let us consider a classical theorem on conditional probabilities, called the *Bayes Theorem*.

Let $B_1, B_2, \cdots, B_k$ represent a partition of sample space $S$. Let $A$ be an event in $S$. Then

$$P(B_i|A) = \frac{P(A \cap B_i)}{P(A)} = \frac{P(A|B_i)P(B_i)}{\sum_{i=1}^{k} P(A|B_i)P(B_i)}.$$

Here is a classical example from Meyer [73] showing an application of this theorem.

**Example 2.1.1** *An item is produced by three factories - F1, F2, and F3. F2 and F3 produce the same number of items. F1 produces twice many items as F2 and F3. 2% of items produced by F1 and F2 are defective, and 4% of items produced by F3 are defective. All of these items are indistinguishable in terms of which factory they come from. First let us understand the partitioning by the following.*
*a) What is the probability that an item is defective?*

*Here $A$ = {item is defective}. $B_1$ = {item came from F1} and likewise define $B_2$ and $B_3$. Note that $P(B_1) = 1/2$ and $P(B_2) = P(B_3) = 1/4$. $P(A|B_1) = P(A|B_2) = 0.02$ and $P(A|B_3) = 0.04$. Then*

$$P(A) = P(A|B_1)P(B_1) + P(A|B_2)P(B_2) + P(A|B_3)P(B_3) = 0.025.$$

*Hence there is a 2.5% chance that an item is defective.*
*b) Suppose that an item is defective. What is the chance that it is made in F1?*

*Let us apply Bayes Theorem.*

$$P(B_1|A) = \frac{P(A|B_1)P(B_1)}{\sum_{i=1}^{k} P(A|B_i)P(B_i)} = \frac{0.02 * 1/2}{0.02 * 1/2 + 0.02 * 1/4 + 0.04 * 1/4} = 0.40.$$

*Hence there is a 40% chance.*

**Independent Events:** We say that the two events $A$ and $B$ are *independent* if $P(A \cap B) = P(A)P(B)$. Intuitively, this implies that the occurrence or nonoccurence of $A$ has no effect on the occurrence or non-occurrence of $B$. An example from Meyer [73] - its insightful:

**Example 2.1.2** *We are rolling two fair die and have three types of events.*
$A$ =*{The first die shows an even number}*
$B$ =*{The second die shows an odd number}*
$C$ =*{Both show an odd or both show an even number}*
*Observe that* $P(A) = P(B) = P(C) = 1/2$. *Note that* $P(A \cap B) = 1/4 = P(A)P(B)$. $P(A \cap C) = 1/4 = P(A)P(C)$. $P(B \cap C) = 1/4 = P(B)P(C)$. *But what about* $P(A \cap B \cap C)$*? Note that* $P(A \cap B \cap C) = 0 \neq P(A)P(B)P(C)$.

Hence the three events $A, B$ and $C$ are said to be *mutually independent* if and only if they are pairwise independent as well as $P(A \cap B \cap C) = P(A)P(B)P(C)$. In general $n$ events are said to be mutually independent, if and only if all combinations of them are mutually independent.

**Random Variable:** A function $X$ assigning every element of a sample space of an experiment to a real number is called a *random variable* (r.v.), i.e. $X : S \to \Re$. For example, $X$ may count the number of 1's in a random binary bit string of length $n$. A r.v. is called *discrete* if the number of possible values it can take is either finite or countably infinite. For each of those values, associate a real value between 0 and 1, i.e. its probability $P(x_i) = P(X = x_i)$. Each $P(x_i) \geq 0$ and $\sum_{i=1}^{\cdots} P(x_i) = 1$. The function $P$ is called the *probability mass (or density) function* and the collection of pairs $(x_i, P(x_i))$ are called the *probability distribution* of $X$. For example, if $X$ is a r.v. describing the number of heads in three tosses of a coin, than the range of $X$ is $\{0, 1, 2, 3\}$ and $P(0) = P(3) = 1/8$, $P(1) = P(2) = 3/8$. Consider the following example.

**Example 2.1.3** *Suppose you want to generate several strings, each string consists of several a's followed by a single b. The random character generator spits out an a with probability* $2/3$ *and b with probability* $1/3$. *The string is generated by repeatedly invoking the 'sputter' till it outputs the first b. Let X be the random variable denoting the number of times the sputter is invoked. Note that X can take values* $1, 2, 3, \ldots$. *Observe that* $P(X = 1) = 1/3$, $P(X = 2) = 2/3 * 1/3$, *and* $P(X = i) = (2/3)^{i-1}1/3$. *Note that* $\sum_{i=1}^{\infty} P(X = i) = 1/3[\sum_{i=0}^{\infty}(2/3)^i] = 1$.

A r.v. $X$ is said to be *continuous* if there exists a function $f$, called the probability mass function, satisfying (a) $f(x) \geq 0$, for all $x$. (b)

$\int_{-\infty}^{+\infty} f(x)dx = 1$ and (c) For all $-\infty < a < b < +\infty$, $P(a < x < b) = \int_a^b f(x)dx$, i.e. the area of the curve under $f(x)$ between $x = a$ and $x = b$. For example, let $X$ be a continuous r.v. with probability mass function given by $f(x) = 2x$, for $0 \leq x \leq 1$, and is 0 elsewhere. Observe that $f(x)$ satisfies the three requirements as (a) $f(x) \geq 0$ for all $x$, (b) $\int_{-\infty}^{+\infty} f(x)dx = \int_0^1 2xdx = 1$, and (c) $-\infty < a < b < +\infty$, $P(a < x < b) = \int_a^b f(x)dx = \int_a^b 2xdx = b^2 - a^2$. For example, if $a = 0$ and $b = 1/2$, then $P(0 \leq x \leq 1/2) = 1/4$.

**Binomial Distribution:** The *Binomial distribution* is defined as follows. Consider an experiment, where $A$ is an event. Let $P(A) = p$ and $P(\bar{A}) = q = 1 - p$. Let us repeat the experiment $n$ times and define a random variable $X$ indicating the number of times $A$ occurs. It is assumed that each experiment is independent of others. What is the probability that $X = k$, for some $1 \leq k \leq n$? It is easy to see that $P(X = k) = \binom{n}{k} p^k q^{n-k}$. $X$ is called a *binomial random variable* with parameters $p$ and $n$. Individual experiments (trials) are called *Bernoulli trials*. Also observe that $\sum_{k=0}^{n} P(X = k) = \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k} = (p + q)^n = 1^n = 1$

**CDF - Cumulative Distribution Function:** In Statistics we typically perform random experiments. They can be repeated any number of times with a known set of outcomes. We associate random variables and probability distributions to such experiments. For each random variable, this association is achieved by a function called the *cumulative distribution function*.

For example, again consider the following simple experiment. We flip a fair coin three times, and let the r.v. $X$ be the number of 'Heads'. Note that the sample space

$$S = \{HHH, THH, HTH, HHT, TTH, THT, HTT, TTT\}.$$

The random variable $X$ maps $S \to \{0, 1, 2, 3\}$. Note that $X(THH) = 2$ and $X(HTT) = 1$. We have already seen the probability mass distribution function $P$. Now define the cumulative distribution function $F$ for $X$ as $F(x) = P(X \leq x)$, i.e. the probability of $X$ being less than or equal to $x$. Observe that $F(x) = 0$ if $x < 0$; $F(x) = P(0) = 1/8$ if $x \leq 0$; $F(x) = P(0) + P(1) = 1/2$ if $x \leq 1$; $F(x) = P(0) + P(1) + P(2) = 7/8$ if $x \leq 2$; and $F(x) = 1$ if $x \leq 3$.

The CDF of a continuous r.v. $X$ with probability mass function $f$ is given by $F(x) = P(X \leq x) = \int_{-\infty}^{x} f(s)ds$.

**Expected Value:** Expected value of a discrete r.v. $X$ is defined as

$E[X] = \sum_{i=1}^{\cdots} x_i P(x_i)$. If this series converges, than $E[X]$ is also called the *mean value* of $X$. Expected value of a continuous r.v. $X$ is defined analogously, i.e., $E[X] = \int_{-\infty}^{+\infty} xf(x)dx$.

For example, consider $X$ to be uniformly distributed over the interval $[a, b]$. We know that the probability distribution function

$$f(x) = \begin{cases} \frac{1}{b-a} & a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

Hence $E[X] = \int_a^b \frac{x}{b-a} dx = (a+b)/2$

Next we show that for a Binomial distribution $X$ with parameters $n$ and $p$, $E[X] = np$. Note that $P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$. Thus,

$$\begin{aligned} E[X] &= \sum_{k=0}^{n} k \frac{n!}{k!(n-k)!} p^k (1-p)^{n-k} \\ &= \sum_{k=1}^{n} \frac{n!}{(k-1)!(n-k)!} p^k (1-p)^{n-k}, \end{aligned}$$

As, for $k = 0$, $k\frac{n!}{k!(n-k)!} p^k (1-p)^{n-k} = 0$. We will perform change of variables and set $s = k - 1$. Now we have

$$\begin{aligned} E[X] &= \sum_{s=0}^{n-1} n \frac{(n-1)!}{s!(n-1-s)!} p^{s+1} (1-p)^{n-1-s} \\ &= np \sum_{s=0}^{n-1} \frac{(n-1)!}{s!(n-1-s)!} p^s (1-p)^{n-1-s} \\ &= np(p + (1-p))^{n-1} \\ &= np \end{aligned}$$

**Independent Random Variables:** Two random variables defined over the same sample space are said to be independent if the outcome of one doesn't depend on the outcome of the other. In case of discrete r.v., by independence we mean, $P(X = x_i, Y = y_j) = P(X = x_i) \cdot P(Y = y_j)$ for all possible values of $i$ and $j$. Alternatively, we can say that $P(X = x_i | Y = y_j) = P(X = x_i)$ and $P(Y = y_i | X = x_j) = P(Y = y_i)$ for all values of $i$ and $j$. As an example, let us toss a fair coin four times. Define r.v. $X$ to be the number of heads obtained in the first two tosses and r.v. $Y$ to be the number of heads obtained in the last two tosses. Observe that $P(X = 0, Y = 0) = P(X = 0)P(Y = 0) = 1/16$, $P(X = 0, Y = 1) = P(X = 0)P(Y = 1) = 1/8$, $P(X = 1, Y = 1) = P(X = 1)P(Y = 1) = 1/4$, $P(X = 2, Y = 1) = P(X = 2)P(Y = 1) = 1/8$, etc. In each of the possibilities we observe that $P(X = x_i, Y = y_j) = P(X = x_i)P(Y = y_j)$ and thus $X$ and $Y$ are independent random variables.

**Linearity of Expectation:** Let $X$ and $Y$ be two random variables mapping elements of a sample space $S$ to real numbers. Assume that $E[X]$ and $E[Y]$ are finite. Linearity of Expectation says that $E[aX + bY] = aE[X] + bE[Y]$ for constants $a$ and $b$. (Note that $X$ and $Y$ need not be independent.) The proof is fairly straightforward. Observe that, by definition,

$$\begin{aligned} E[aX + bY] &= \sum_{\omega \in S} (a \cdot X[\omega] + b \cdot Y[\omega]) \cdot P(\omega) \\ &= \sum_{\omega \in S} (a \cdot X[\omega] \cdot P(\omega) + b \cdot Y[\omega] \cdot P(\omega)) \\ &= a \sum_{\omega \in S} X[\omega] \cdot P(\omega) + b \sum_{\omega \in S} Y[\omega] \cdot P(\omega) \\ &= aE[X] + bE[Y] \end{aligned}$$

This easily generalizes to the linear combination of $n$ random variables, i.e. expectation of the linear combination of $n$ variables is same as the linear combination of the expectation of these variables. Given this, it is trivial to compute the expectation of a Binomial r.v. $X$. Note that $X = X_1 + X_2 + \cdots + X_n$, where each $X_i$ is a Bernoulli indicator random variable, with success probability $p$. The expected value of each of the indicator variable is $E[X_i] = 1.p + 0.(1 - p) = p$. Thus $E[X] = E[X_1 + \cdots X_n] = E[X_1] + \cdots + E[X_n] = np$.

**Variance:** The *variance* of a r.v. $X$ is defined to be the expected value of the r.v. $(X - E[X])^2$. We will write this as $V[X] = E[X - E[X]]^2$. One of the exercises asks for verifying that $V[X] = E[X - E[X]]^2 = E[X^2] - (E[X])^2$. Let us calculate the variance of the r.v. that is uniformly distributed over the interval $[a, b]$. We know that $E[X] = (a + b)/2$. Note that $E[X^2] = \int_a^b \frac{x^2}{b-a} dx = \frac{b^3 - a^3}{3(b-a)}$. Since $V[X] = E[X^2] - [E[X]]^2$ (see Exercises), we obtain $V[X] = \frac{b^3 - a^3}{3(b-a)} - (\frac{a+b}{2})^2 = \frac{(b-a)^2}{12}$.

Observe the following. Let $X$ and $Y$ be two independent r.v. defined over the same sample space. Now we can express the variance of their sum as

$$\begin{aligned} V[X + Y] &= E[(X + Y)^2] - (E[X + Y])^2 \\ &= E[X^2 + Y^2 + 2XY] - (E[X] + E[Y])^2 \\ &= E[X^2] + E[Y^2] + E[2XY] - (E[X]^2 + E[Y]^2 + 2E[X]E[Y]) \\ &= E[X^2] - E[X]^2 + E[Y^2] - E[Y]^2 \\ &= V[X] + V[Y] \end{aligned}$$

Note that due to independence $E[XY] = E[X]E[Y]$. Next, we compute $V[X]$ of a binomial r.v. $X$ with parameters $n$ and $p$. Note that r.v.

$X = X_1 + X_2 + \cdots + X_n$, where each $X_i$'s are identical independent indicator r.v. Thus, $V[X] = nV[X_i]$. Let us now evaluate $V[X_i]$. Note that

$$\begin{aligned} V[X_i] &= E[X_i^2] - E[X_i]^2 \\ &= \{1^2.p + 0^2.(1-p)\} - p^2 \\ &= p - p^2 \end{aligned}$$

Thus, $V[X] = nV[X_i] = n(p - p^2) = np(1-p)$.

## 2.2 *Chebyshev's Inequality and Law of Large Numbers*

Now let us look at a famous inequality due to Chebyshev.

**Theorem 2.2.1** *Let $X$ be a random variable and let $c$ be a real number. If $E[X-c]^2$ is finite and $\epsilon > 0$, then*

$$P(|X - c| \geq \epsilon) \leq \frac{1}{\epsilon^2} E[X-c]^2.$$

**Proof.** Let $X$ be a continuous r.v. Let $R = \{x : |x - c| \geq \epsilon\}$. Note that $P(|X-c| \geq \epsilon) = \int_R f(x)dx$. Observe that $|x-c| \geq \epsilon$ implies $\frac{(x-c)^2}{\epsilon^2} \geq 1$. Thus we have

$$\begin{aligned} P(|X-c| \geq \epsilon) &= \int_R 1 \cdot f(x)dx \\ &\leq \int_R \frac{(x-c)^2}{\epsilon^2} f(x)dx \\ &\leq \int_{-\infty}^{+\infty} \frac{(x-c)^2}{\epsilon^2} f(x)dx \\ &= \frac{1}{\epsilon^2} E[X-c]^2 \end{aligned}$$

∎

For an application of this inequality consider the following. Suppose we repeat an experiment multiple times. Then the relative frequency of the occurrence of an event should converge to its actual probability. For example, a factory is producing items. Suppose we don't know what is the failure probability. One way to estimate this probability is to take a large sample and see what percentage are faulty. This percentage will be a good estimate of the failure probability. Of course this should be taken with a grain of salt. This really depends upon what kind of sample is chosen. Essentially what we are heading towards is that if the elements in the sample are chosen randomly, than the percentage of faulty items will be a true indicator of failure probability.

Consider a particular event $A$ in an experiment. This experiment is repeated $n$ times, and each run is independent of other runs. Let $n_A$ be the number of times $A$ occurs in the runs. Let $f_A = n_A/n$. Let $P(A) = p$. For a positive $\epsilon > 0$,

$$P(|f_A - p| \geq \epsilon) \leq \frac{p(1-p)}{n\epsilon^2},$$

or equivalently,

$$P(|f_A - p| < \epsilon) \geq 1 - \frac{p(1-p)}{n\epsilon^2}.$$

This can be shown as follows. Observe that $n_A$ is a binomially distributed random variable. $E[n_A] = np$ and $V[n_A] = np(1-p)$. Since $f_A = n_A/n$, $E[f_A] = p$ and $V[f_A] = p(1-p)/n$. Recall Chebyshev's inequality (Theorem 2.2.1 and also see Exercise 2.21) which states that $P(|X - \mu| < \epsilon) \geq 1 - \frac{Var(X)}{\epsilon^2}$. Substituting $X = f_A$, $p = \mu$, we obtain

$$P(|f_A - p| < \epsilon) \geq 1 - \frac{V[f_A]}{\epsilon^2}.$$

Set $V[f_A] = p(1-p)/n$, and we obtain

$$P(|f_A - p| < \epsilon) \geq 1 - \frac{p(1-p)}{n\epsilon^2}.$$

Note that

$$\lim_{n\to\infty} P(|f_A - p| < \epsilon) = 1.$$

This is essentially the meaning of probability of an event $A$, i.e. the relative frequency converges to $P(A)$ when an experiment is repeated for a large number of times. This is what is called as the *Law of Large Numbers*. To get some more intuition, think of what could have happened if this wasn't true - i.e., no matter how many times you repeat the experiment, the frequency doesn't converge to $P(A)$. What will be the state of various fields such as Physics, Nature, Evolution, ...?

Here is a different type of question. How many times should we repeat the experiment, so that the relative frequency $f_A$ differs from $P(A)$ by at most 0.01 with probability at least 0.9?

We need to choose $n$ so that for $\epsilon = 0.01$, $1 - \frac{p(1-p)}{n\epsilon^2} = 0.9$. This implies that $n = \frac{p(1-p)}{0.1\epsilon^2}$. For example if $p = 1/2$, than $n = 25000$. What this means is that if we toss a coin 25000 times, than we are 90% sure that the relative frequency of getting a head is within 0.01 of the theoretical probability.

## 2.3 Normal Distribution, mgf, and the Central Limit Theorem

Next, let us discuss Normal distributions, as we will need them in later chapters. Random variable $X$ has a *normal distribution* if its probability density function is of the form

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}, -\infty < x < \infty.$$

Usually, it is denoted by $\mathcal{N}(\mu, \sigma^2)$, where $-\infty < \mu < +\infty$ is the mean (expected value) and $\sigma > 0$ is the standard deviation, i.e. positive square-root of the variance. It is also referred to as Gaussian or bell-shaped distribution. See Figure 2.1 for an illustration. It is a valid distribution, as $f(x) \geq 0$ for all values of $x$ and $\int_{-\infty}^{+\infty} f(x)dx = 1$ (see Exercises). The function $f$ is symmetric around $x = \mu$. If we trace the boundary of the function, it changes from being convex to concave, and these points of inflection are at $x = \mu \pm \sigma$. The distribution $\mathcal{N}(0,1)$ is referred to as a *standardized normal distribution*.

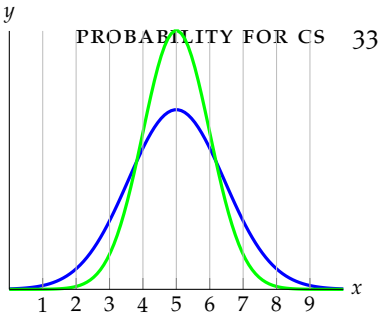


Figure 2.1: Illustration of $\mathcal{N}(5, 1.44)$ and $\mathcal{N}(5,1)$ in blue and green, respectively

Some quick facts about Normal distribution that are helpful includes

1. If $X$ is $\mathcal{N}(\mu, \sigma^2)$ and $Y = aX + b$ then $Y$ is $\mathcal{N}(a\mu + b, a^2\sigma^2)$.
2. If $X$ has distribution $\mathcal{N}(\mu, \sigma^2)$ and if $Y = \frac{X-\mu}{\sigma}$, then $Y$ has distribution $\mathcal{N}(0,1)$.
3. If $X$ has distribution $\mathcal{N}(0,1)$, then $Pr(a \leq X \leq b) = \frac{1}{2\pi}\int_a^b e^{-x^2/2}dx$.

### 2.3.1 A special case of the central limit theorem

We state a theorem, which is a special case of the Central Limit Theorem (see Meyer [73]). Intuitively, the following theorem states that the arithmetic mean of $n$ independent observations from the same random variable, has for large $n$, a normal distribution.

**Theorem 2.3.1** *Let $X_1, X_2, \cdots, X_n$ be $n$ independent r.v. all of which have the same distribution. Let $\mu = E[X_i]$ and $\sigma^2 = V[X_i]$ be their common expectation and variance, respectively. Define a r.v. $S = \sum_{i=1}^n X_i$. Then $E[S] = n\mu$ and $V[S] = n\sigma^2$. Moreover, for large $n$, $T_n = \frac{S-n\mu}{\sqrt{n}\sigma}$ has essentially the distribution $\mathcal{N}(0,1)$.*

Before we discuss a proof sketch of this theorem, let us look at an example from Meyer's [73] to gain some intuition.

**Example 2.3.2** *Consider a box containing three types of balls - 20 balls labelled with a zero, 30 with a one and 50 with a two. In this experiment, we draw a random ball, note its label, and then place it back. We will repeatedly pick the balls, and eventually report the average of their labels. We are*

*interested in understanding the distribution of the averages when we repeat this experiment for a large number of rounds. The above theorem claims that the average behaves like a normal distribution. Let $X_i$ be the label of the ball drawn in Round i of this experiment. We are interested in the random variable $M_i = \frac{1}{i}\sum_{k=1}^{i} X_k$, denoting the averages of the labels drawn up to and including the Round i. Note that each $X_i$ takes values $0, 1,$ and $2$, with probabilities $0.2, 0.3$ and $0.5$, respectively. Observe that $M_1 = X_1$ and it takes values $0, 1$ or $2$ with probability $0.2, 0.3$ and $0.5$, respectively. Next let us look at $M_2$.*

| $m = \frac{X_1+X_2}{2}$ | 0 | 1/2 | 1 | 3/2 | 2 |
|---|---|---|---|---|---|
| $P(M_2 = m)$ | 0.04 | 0.12 | 0.29 | 0.30 | 0.25 |

| $m = \frac{X_1+X_2+X_3}{3}$ | 0 | 1/3 | 2/3 | 1 | 4/3 | 5/3 | 2 |
|---|---|---|---|---|---|---|---|
| $P(M_3 = m)$ | 0.008 | 0.036 | 0.114 | 0.207 | 0.285 | 0.225 | 0.125 |

*Hopefully, this example convinces us that for large values of n, $M_n$ converges to a Normal distribution.*

We will sketch a proof of Theorem 2.3.1 using the *moment generating functions (mgf)* and their interesting properties. Let us first briefly explore mgf's.

### 2.3.2 *Moment Generating Functions*

For a random variable $X$, its moment generating function $M_X(t) = E[e^{tX}]$. Formally,

**Definition 2.3.3** *Let X be a discrete r.v. taking values $\{x_1, x_2, x_3 \dots\}$ with probabilities $\{p_1, p_2, p_3 \dots\}$, respectively. The moment generating function (mgf) of X is*

$$M_X(t) = \sum_{i=1}^{\infty} e^{tx_i} p_i.$$

*If X is a continuous r.v. with probability distribution function $f$, then the mgf is given by*

$$M_X(t) = \int_{-\infty}^{+\infty} e^{tx} f(x) dx.$$

Consider following standard examples of mgf's.

**Example 2.3.4** *Let r.v. X be uniformly distributed in the interval $[a, b]$ on real line. The mgf is given by*

$$M_X(t) = \int_a^b \frac{e^{tx}}{b-a} dx = \frac{1}{(b-a)t}(e^{bt} - e^{at}).$$

**Example 2.3.5** *Let r.v. X be binomially distributed with parameters n and p. The mgf of X is*

$$M_X(t) = \sum_{k=0}^{n} e^{tk} \binom{n}{k} p^k (1-p)^{n-k} = [pe^t + (1-p)]^n.$$

**Example 2.3.6** *Let r.v. X has the Normal distribution $\mathcal{N}(\mu, \sigma^2)$. Its mgf is given by*

$$M_X(t) = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{+\infty} e^{tx} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} dx = e^{t\mu + \sigma^2 t^2/2}.$$

By the Maclaurin series expansion, $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$. Therefore, $e^{tx} = 1 + tx + \frac{(tx)^2}{2!} + \frac{(tx)^3}{3!} + \cdots$. By definition,

$$M_X(t) = E[e^{tX}] = E[1 + tX + \frac{(tX)^2}{2!} + \frac{(tX)^3}{3!} + \cdots].$$

If we assume that the linearity of expectation will carry over for infinite sums, then

$$M_X(t) = E[e^{tX}] = 1 + tE[X] + \frac{t^2 E[X^2]}{2!} + \frac{t^3 E[X^3]}{3!} + \cdots.$$

Next we will take some higher order derivatives, and substitute the value of $t$ as 0, to observe the following (note that this assumes that we can take derivatives of infinite series): $M'_X(0) = E[X]$, $M''_X(0) = E[X^2]$, and in general $M_X^{(n)}(0) = E[X^n]$. In fact this is the reason these functions are called the moment generating functions.

**Example 2.3.7** *Recall that when X has binomially distribution with parameters n and p, its mgf is $M_X(t) = [pe^t + (1-p)]^n$. Observe that $M'_X(t) = n(pe^t + (1-p))^{n-1} pe^t$ and $M'_X(0) = np = E[X]$. Similarly, $M''_X[0] = E[X^2] = np[(n-1)p + 1]$. We can also observe that $Var[X] = M''_X(0) - [M'_X(0)]^2 = np(1-p)$.*

**Example 2.3.8** *Let r.v. X has the Normal distribution $\mathcal{N}(\mu, \sigma^2)$. Its mgf is $M_X(t) = e^{t\mu + \sigma^2 t^2/2}$. Now $M'_X(t) = (\mu + \sigma^2 t) e^{t\mu + \sigma^2 t^2/2}$ and $M'_X(0) = E[X] = \mu$. Similarly, $M''_X(0) = E[X^2] = \mu^2 + \sigma^2$. Note that $Var[X] = M''(0) - M'(0)^2 = \sigma^2$.*

**Observation 2.3.9** *Let r.v. X has mgf $M_X$. Let r.v. $Y = \alpha X + \beta$. where $\alpha$ and $\beta$ are constants. Mgf of Y is given by*

$$M_Y(t) = E[e^{tY}] = E[e^{t(\alpha X + \beta)}] = e^{t\beta} M_X(t\alpha).$$

**Observation 2.3.10** *Let X and Y be independent r.v. with mgf's $M_X$ and $M_Y$, respectively. Let r.v. $Z = X + Y$. Then mgf of Z is given by*

$$M_Z(t) = E[e^{tZ}] = E[e^{t(X+Y)}]] = E[e^{tX}] E[e^{tY}] = M_X(t) M_Y(t).$$

Generalizing the above observation to $n$ independent random variables, we obtain

**Theorem 2.3.11** *Let $X_1, \ldots, X_n$ be $n$ independent random variables with mgf's $M_{X_1}, \ldots, M_{X_n}$, respectively. Let r.v. $Z = X_1 + \cdots + X_n$. Then mgf of $Z$ is given by $M_Z(t) = M_{X_1}(t) \cdots M_{X_n}(t)$.*

**Observation 2.3.12** *Let $X$ and $Y$ be independent r.v. with Normal distributions $\mathcal{N}(\mu_1, \sigma_1^2)$ and $\mathcal{N}(\mu_2, \sigma_2^2)$, respectively. Let r.v. $Z = X + Y$. Then by Theorem 2.3.11 mgf of $Z$,*

$$M_Z(t) = M_X(t)M_Y(t) = e^{t\mu_1 + \sigma_1^2 t^2/2} e^{t\mu_2 + \sigma_2^2 t^2/2} = e^{t(\mu_1+\mu_2) + (\sigma_1^2+\sigma_2^2)t^2/2}.$$

*But that corresponds to the mgf of the Normal distribution $\mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$. Thus the sum of two independent Normal distributions is a Normal distribution.*

From the above discussion we can conclude that

**Theorem 2.3.13** *Let $X_1, \ldots, X_n$ be $n$ independent random variables with Normal distributions $\mathcal{N}(\mu_i, \sigma_i^2)$, for $i = 1, \ldots, n$. Let r.v. $Z = \sum_{i=1}^n X_i$. Then $Z$ has a Normal distribution $\mathcal{N}(\sum_{i=1}^n \mu_i, \sum_{i=1}^n \sigma_i^2)$. Thus the sum of independent Normal distributions is a Normal distribution.*

### 2.3.3 *Proof of Theorem 2.3.1*

Now we are in a position to sketch the proof of the special case of the Central Limit Theorem 2.3.1. Recall that we want to show the following:

Let $X_1, X_2, \cdots, X_n$ be $n$ independent r.v. all of which have the same distribution (need not be Normal). Let $\mu = E[X_i]$ and $\sigma^2 = V[X_i]$ be their common expectation and variance, respectively. Let $S = \sum_{i=1}^n X_i$. Note that $E[S] = n\mu$ and $V[S] = n\sigma^2$. For large $n$, we need to show that $T_n = \frac{S - n\mu}{\sqrt{n}\sigma}$ has the distribution $\mathcal{N}(0, 1)$. We will show that the mgf of $T_n$ is same as that of the mgf of $\mathcal{N}(0, 1)$.

**Proof.** The mgf of $X_i$ is $M_{X_i}(t) = E[e^{tX_i}]$. Since $X_i$'s are independent and from Observation 2.3.10, mgf of $S$ is given by $M_S(t) = (M_{X_i}(t))^n$. Note that $T_n = \frac{S - n\mu}{\sqrt{n}\sigma} = \sum_{i=1}^n (\frac{X_i - \mu}{\sqrt{n}\sigma})$ is a linear function of $S$, and hence by Observation 2.3.9 mgf of $T_n$ is given by

$$M_{T_n}(t) = e^{-(\frac{\mu\sqrt{n}}{\sigma}t)} \left[M_{X_i}(\tfrac{t}{\sigma\sqrt{n}})\right]^n \tag{2.1}$$

$$\ln M_{T_n}(t) = -\tfrac{\mu\sqrt{n}}{\sigma}t + n \ln \left[M_{X_i}(\tfrac{t}{\sigma\sqrt{n}})\right] \tag{2.2}$$

As we have seen earlier, by the Maclaurin series expansion $M_{X_i}(t) = E[e^{tX_i}] = 1 + tE[X_i] + \frac{t^2 E[{X_i}^2]}{2!} + \frac{t^3 E[{X_i}^3]}{3!} + \cdots$. Since $E[X_i] = \mu$ and $E[{X_i}^2] = \mu^2 + \sigma^2$, we have

$$M_{X_i}(t) = 1 + \mu t + \frac{(\mu^2 + \sigma^2)}{2} t^2 + R,$$

where $R$ consists of all other remaining terms. Substituting this in Equation 2.2, we obtain

$$\ln M_{T_n}(t) \quad = -\tfrac{\mu\sqrt{n}}{\sigma}t + n\ln\left[1 + \tfrac{\mu t}{\sqrt{n}\sigma} + \tfrac{(\mu^2+\sigma^2)}{2n\sigma^2}t^2 + R\right]. \qquad (2.3)$$

Note that $\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots$ for $|x| < 1$. For large values of $n$, $\frac{\mu t}{\sqrt{n}\sigma} + \frac{(\mu^2+\sigma^2)}{2n\sigma^2}t^2 + R < 1$. Hence,

$$\ln M_{T_n}(t) = -\frac{\mu\sqrt{n}}{\sigma}t + n\big[(\frac{\mu t}{\sqrt{n}\sigma} + \frac{(\mu^2+\sigma^2)}{2n\sigma^2}t^2 + R) - \frac{1}{2}(\frac{\mu t}{\sqrt{n}\sigma} + \frac{(\mu^2+\sigma^2)}{2n\sigma^2}t^2 + R)^2 + \cdots\big].$$

Omitting the algebraic manipulation, the above expression for $n \to \infty$ results in

$$\lim_{n\to\infty} \ln M_{T_n}(t) = t^2/2.$$

Equivalently,

$$\lim_{n\to\infty} M_{T_n}(t) = e^{t^2/2}.$$

Since the mgf of $\mathcal{N}(0,1)$ is $e^{t^2/2}$, hence the limiting distribution of $T_n$ is $\mathcal{N}(0,1)$. ∎

## 2.4 *More Distributions*

We will briefly explore a few more standard discrete and continuous distributions.

**Poisson Distribution:** Let $X$ be a discrete r.v. taking values $\{0, 1, 2, \dots\}$. $X$ has Poisson distribution with parameter $\alpha > 0$ if $P(X = k) = \frac{e^{-\alpha}\alpha^k}{k!}$.

It turns out that $\sum_{k=0}^{\infty} P(X = k) = 1$, $E[X] = V[X] = \alpha$. One of the key properties of Poission distribution is that it approximates the Binomial distribution for large values of $n$ and small values of $p$. Let $X$ be a r.v. with Binomial distribution, where $P(X = k) = \binom{n}{k}p^k(1-p)^{n-k}$. As $n \to \infty$ and $np = \alpha$, we have that

$$\lim_{n\to\infty} P(X = k) = \frac{e^{-\alpha}\alpha^k}{k!}.$$

We know that $E[X] = np$ and $V[X] = np(1-p)$. For large values of $n$, small values of $p$, and $\alpha = np$, observe that $E[X] = V[X] = \alpha$.

**Geometric Distribution:** This distribution captures the scenario where an experiment is repeatedly executed, independent of previous executions, till a particular event occurs. Let $A$ be the event we want and let us assume that $p$ is the probability of its occurrence. Hence it doesn't occur with probability $1 - p$. Define the r.v. $X$ that counts the number of times the experiment is repeated till $A$ occurs. $X$ takes values $\{1, 2, 3, \dots\}$. $X$ is said to have geometric distribution where $P(X = k) = (1-p)^{k-1}p$. It turns out that $\sum_{k=0}^{\infty} P(X = k) = 1$,

$E[X] = 1/p$ and $V[X] = (1-p)/p^2$. In particular, on the average, we need to execute the experiment $\lceil \frac{1}{p} \rceil$ times to see the event $A$.

**Gamma Distribution:** For any $p > 0$, the Gamma function is defined as

$$\Gamma(p) = \int_0^\infty x^{p-1} e^{-x} dx.$$

Let $X$ be a continuous r.v. taking positive values. $X$ has a Gamma distribution with parameters $r > 0$ and $\alpha > 0$, if for all $x > 0$, its probability density function is given by

$$f(x) = \frac{\alpha}{\Gamma(r)} (\alpha x)^{r-1} e^{-\alpha x}.$$

If $r = 1$, then $f(x) = \alpha e^{-\alpha x}$ and this is pdf of an *exponential distribution*. If $\alpha = 1/2$ and $r = n/2$, for some positive integer $n$, then we obtain the *Chi-Square Distribution*. Its pdf is given by $f(x) = \frac{1}{2^{n/2}\Gamma(n/2)} x^{n/2-1} e^{-x/2}$. Its expected value and variance is given by $n$ and $2n$, respectively.

## 2.5 *Chernoff Bounds*

Suppose we toss a fair coin 1000 times. We expect about 500 tails. What is the probability that we will get over 750 tails? or over 900 tails? Chernoff bounds help us in determining probabilities of extreme events in a large collection of independent events. In this section we will establish these bounds. This section is derived from [2] and notes of Michiel Smid. First let us establish Markov's inequality.

[2] Torben Hagerup and Christine Rüb. A guided tour of Chernoff bounds. *Inf. Process. Lett.*, 33(6):305–308, 1990

**Theorem 2.5.1** *Let $X$ be a non-negative discrete r.v. and $s > 0$ be a constant. Then $P(X \geq s) \leq E[X]/s$.*

**Proof.** Note that $E[X] = \sum_{i=0}^{\infty} i.P(X = i) \geq \sum_{i=s}^{\infty} i.P(X = i) \geq s \sum_{i=s}^{\infty} P(X = i) = sP(X \geq s)$. Hence $P(X \geq s) \leq E[X]/s$. ∎

Let $X_1, \cdots X_n$ be identical 0-1 independent r.v's, such that $P(X_i = 1) = p_i$ and $P(X_i = 0) = 1 - p_i$. Define $X = \sum_{i=1}^n X_i$ and let $m = \sum_{i=1}^n p_i$. Note that

$$E[X] = E[\sum_{i=1}^n X_i] = \sum_{i=1}^n E[X_i] = \sum_{i=1}^n p_i = m.$$

Suppose we have a fair coin, which we toss $n$ times. We define 0-1 r.v's $X_1, \cdots, X_n$, where $X_i = 1$ if the $i$-th toss was a head otherwise $X_i = 0$. $P(X_i = 0) = P(X_i = 1) = 1/2$. Let $X = \sum_{i=1}^n X_i$. Now $E[X] = n/2 = m$. We are interested in estimating the probability of $X$ deviating from $(1 \pm \epsilon)m$, for $0 < \epsilon < 1$. From Markov's inequality we obtain $P(X > (1+\epsilon)m) \leq 1/(1+\epsilon)$. We will show the following,

which are collectively known as *Chernoff bounds* in the algorithms community.

$$P(X \geq (1+\epsilon)m) \leq \exp(-\epsilon^2 m/3)$$

$$P(X \leq (1-\epsilon)m) \leq \exp(-\epsilon^2 m/2).$$

Let $t > 0$. Let us first deal with proving $P(X \geq (1+\epsilon)m) \leq \exp(-\epsilon^2 m/3)$.

$$\begin{aligned} P(X \geq (1+\epsilon)m) &= P(e^{tX} \geq e^{t(1+\epsilon)m}) && (2.4)\\ &= e^{-t(1+\epsilon)m} e^{t(1+\epsilon)m} P(e^{tX} \geq e^{t(1+\epsilon)m}) && (2.5)\\ &\leq e^{-t(1+\epsilon)m} E[e^{tX}] && (2.6) \end{aligned}$$

Equation 2.6 follows from Markov inequality applied to $P(e^{tX} \geq e^{t(1+\epsilon)m}) \leq e^{-t(1+\epsilon)m} E[e^{tX}]$. Observe that $E[e^{tX}] = E[e^{t\sum_{i=1}^{n} X_i}] = E[\Pi_{i=1}^{n} e^{tX_i}] = \Pi_{i=1}^{n} E[e^{tX_i}]$, as $X_i$'s are independent. $E[e^{tX_i}] = p_i e^t + (1-p_i)e^0$. Hence $E[e^{tX}] = \Pi_{i=1}^{n}(p_i e^t + (1-p_i)) = \Pi_{i=1}^{n}(1 + p_i(e^t - 1)) \leq \Pi_{i=1}^{n} e^{p_i(e^t-1)} = e^{\sum_{i=1}^{n} p_i(e^t-1)} = e^{m(e^t-1)}$. (Note that for this we used the fact that $e^x \geq 1 + x$.) Hence,

$$P(X \geq (1+\epsilon)m) \leq e^{-t(1+\epsilon)m + m(e^t-1)}.$$

This expression holds for all values of $t \in R$. It is minimized for $t = \ln(1+\epsilon)$. To see this, one can write $e^{-t(1+\epsilon)m+m(e^t-1)} = [e^{-t(1+\epsilon)+(e^t-1)}]^m$. To minimize this, one needs to minimize $-t(1+\epsilon) + (e^t - 1)$. Differentiate this with respect to $t$, and we obtain that $-(1+\epsilon) + e^t = 0$ or $t = \ln(1+\epsilon)$.

Thus,

$$P(X \geq (1+\epsilon)m) \leq (1+\epsilon)^{-(1+\epsilon)m} e^{m\epsilon} = \left(\frac{e^\epsilon}{(1+\epsilon)^{1+\epsilon}}\right)^m \leq e^{-\epsilon^2 m/3}.$$

To show the last inequality, one needs to prove that $\epsilon - (1+\epsilon)\ln(1+\epsilon) \leq -\frac{1}{3}\epsilon^2$, which is left as an exercise.

Next, we show that $P(X \leq (1-\epsilon)m) \leq \exp(-\epsilon^2 m/2)$. The proof is along the same lines as the previous one.

$$\begin{aligned} P(X \leq (1-\epsilon)m) &= P(m - X \geq \epsilon m) && (2.7)\\ &= P(e^{t(m-X)} \geq e^{t\epsilon m}) && (2.8)\\ &\leq e^{-t\epsilon m} E[e^{t(m-X}] && (2.9)\\ &= e^{tm(1-\epsilon)} E[e^{-tX}] && (2.10) \end{aligned}$$

Note that

$$E[e^{-tX}] = \Pi_{i=1}^{n} E[e^{-tX_i}] \tag{2.11}$$
$$= \Pi_{i=1}^{n} [p_i e^{-t} + (1 - p_i)] \tag{2.12}$$
$$. = \Pi_{i=1}^{n} [1 - p_i(1 - e^{-t})] \tag{2.13}$$
$$\leq \Pi_{i=1}^{n} [e^{-p_i(1-e^{-t})}] \tag{2.14}$$
$$= e^{-\sum_{i=1}^{n} p_i(1-e^{-t})} \tag{2.15}$$
$$= e^{-m(1-e^{-t})} \tag{2.16}$$

Therefore,

$$P(X \leq (1-\epsilon)m) \leq e^{-m(1-e^{-t})} e^{tm(1-\epsilon)}.$$

This is minimized for $t = -\ln(1-\epsilon)$. Hence,

$$P(X \leq (1-\epsilon)m) \leq (1-\epsilon)^{-m(1-\epsilon)} e^{-m\epsilon} = \left[ \left( \frac{1}{1-\epsilon} \right)^{1-\epsilon} e^{-\epsilon} \right]^{m} \leq e^{-\epsilon^2 \frac{m}{2}}.$$

## 2.6 Exercises

**2.1** *Consider the following three events and assume that each of the dice is fair and outcome of the roll of one die is independent of the outcome of roll of any other die.*

*A: Roll six dice and there is at least one 6.*

*B: Roll twelve dice and there are at least two 6s.*

*C: Roll eighteen dice and there are at least three 6s.*

*Which of the above events has the highest chance of occurring?*
*What will happen if the dice are not fair?*

**2.2** *Let $X$ be a binomially distributed r.v. with parameters $n$ and $p$. Then $E[X] = \sum_{k=0}^{n} k\binom{n}{k} p^k (1-p)^{n-k} = np$.*

**2.3** *Let the r.v. $X$ be such that $X = c$ where $c$ is a constant. Show that $E[X] = c$.*

**2.4** *Let $X$ and $Y$ be two r.v. Show that $E[X + Y] = E[X] + E[Y]$.*

**2.5** *Let $X$ and $Y$ be two independent random variables. Show that $E[XY] = E[X]E[Y]$.*

**2.6** *Let $X$ be a discrete r.v. and let $g$ be a function such that $g : \mathbb{R} \to \mathbb{R}$. Show that*

1. *$E[g(X)] = \sum_x g(x) P(X = x)$.*
2. *Let $X$ be a r.v. that represents the value of the roll of a fair die. Show that $E[X] = 3.5$. Consider the function $g : \mathbb{R} \to R$ such that $g(x) = x^2$. Show that $E[g(x)] = 15.167$.*

**2.7** *Show that the variance of a r.v.* $V[X] = E[X^2] - [E[X]]^2$.

**2.8** *If* $X$ *and* $Y$ *are independent r.v. than* $V[X+Y] = V[X] + V[Y]$.

**2.9** *Let* $X$ *be a r.v. and* $c$ *a constant.*

1. *Show that* $V[X+c] = V[X]$.
2. *Show that* $V[cX] = c^2 V[X]$.
3. *Show that* $V[X] \geq 0$.
4. *Show that* $V[X] = 0$ *if and only if the r.v.* $X$ *is a constant.*

**2.10** *Show that for the Normal distribution* $f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left[\frac{x-\mu}{\sigma}\right]^2} \geq 0$ *for all values of* $-\infty < x < \infty$.

**2.11** *Show that for the Normal distribution* $f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left[\frac{x-\mu}{\sigma}\right]^2}$, $\int_{-\infty}^{+\infty} f(x)dx = 1$. *This is not straightforward. It is best to consider first the case where you take the product of two standard normal distributions and show that* $\int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-y^2/2} dy = 1$.

**2.12** *Let* $Z$ *be a r.v. with standard normal distribution. In programming language C, we have the function* rand() *that returns a (pseudo-)random number uniformly distributed, say in the range from* 0 *to* 1. *Using Central Limit Theorem, can you think of a method for generating values for* $Z$ *with the help of* rand() *function? Following is a possibility. Execute* rand() $k$ *times and let the values returned be* $x_1, x_2, \ldots, x_k$. *Assign* $z = x_1 + x_2 + \cdots + x_k - k/2$. *Do the values generated by the above procedure have* $\mathcal{N}(0,1)$ *distribution as* $k \to \infty$*?*

**2.13** *Show that for a Poisson r.v.* $X$ *with parameter* $\alpha > 0$ *and* $P(X = k) = \frac{e^{-\alpha}\alpha^k}{k!}$, $\sum_{k=0}^{\infty} P(X = k) = 1$ *and* $E[X] = V[X] = \alpha$.

**2.14** *Assume that one of the online store gets 10 Million hits where the potential customers are trying to identify what they want to buy. We can assume all the hits are independent. What is the probability that* 0.001% *of these hits will result in actual orders? (Hint: Think of Poisson.)*

**2.15** *Show that for a geometric random variable with* $P(X = k) = (1-p)^{k-1}p$, $\sum_{k=0}^{\infty} P(X = k) = 1$, $E[X] = 1/p$ *and* $V[X] = (1-p)/p^2$.

**2.16** *Show that for a geometric random variable with* $P(X = k) = (1-p)^{k-1}p$, $P(X \geq s+t | X > s) = P(X > t)$, *where* $s$ *and* $t$ *are positive integers. Alternatively, we can say that the geometric distribution is memoryless.*

**2.17** *Show that for an integer* $p > 0$, *the Gamma function* $\Gamma(p) = (p-1)\Gamma(p-1)$.

**2.18** *This exercise proves the Cauchy-Schwarz inequality. Let $X$ and $Y$ be two r.v. with finite mean and variances. Let $t$ be any real number.*

1. *Show that* $E[(Y - tX)^2] = E[Y^2] - 2tE[XY] + t^2E[X^2] \geq 0$.
2. *Show that* $t = \frac{E[XY]}{E[X^2]}$ *minimizes* $E[Y^2] - 2tE[XY] + t^2E[X^2]$.
3. *Show that* $\sqrt{E[X^2]E[Y^2]} \geq |E[XY]|$.

**2.19** *A function $g$ is said to be* convex *if all lines that are tangent to $g$ lie below $g$. Let $X$ be a r.v. and $g$ be a convex function. Show the following:*

1. *Using the convexity of $g$, show that for $\mu = E[X]$ the tangent line to $g$ at the point $(\mu, g(\mu))$ is below $g$.*
2. *Let the equation of the above tangent line be $y = mx + b$. Show that for any $x \in \mathbb{R}$, $g(x) \geq mx + b$.*
3. *Show that* $E[mX + b] = g(\mu) = g(E[X])$.
4. *Show that* $E[g(X)] \geq g[E[X]]$.

**2.20** *This exercise provides an alternate and straightforward proof of Markov's inequality (see Theorem 2.5.1). Let $X$ be a non-negative discrete r.v. and $s > 0$. Show the following.*

1. *Show that for an indicator random variable $I$,* $E[I] = P[I = 1]$.
2. *Define an Indicator r.v. $I_s$ such that $I_s = 1$ if $X \geq s$ and 0 otherwise. Show that* $sI_s \leq X$.
3. *Show that* $sE[I_s] \leq E[X]$.
4. *Show that* $P(X \geq s) \leq \frac{E[X]}{s}$.

**2.21** *This exercise provides an alternate proof of Chebyshev's inequality (see Theorem 2.2.1) using Markov's inequality. Let $X$ be a random variable with mean $\mu$ and variance $\sigma^2$. Let $s > 0$ be a constant.*

1. *Show that* $P(|X - \mu| \geq s) = P((X - \mu)^2 \geq s^2)$
2. *Show that* $P((X - \mu)^2 \geq s^2) \leq \frac{E[(X-\mu)^2]}{s^2}$
3. *Show that* $P(|X - \mu| \geq s) \leq \frac{\sigma^2}{s^2}$

**2.22** *Let $X$ be a r.v. and let $s > 0$ and $t > 0$ be constants. Use Markov's inequality to show the following:*

1. $P(X \geq s) = P(e^{tX} \geq e^{ts})$
2. $P(X \geq s) \leq \frac{E[e^{tX}]}{e^{ts}}$

**2.23** *In the last three exercises we have established various upper bounds for* $P(X \geq s)$. *Let the r.v. X obeys the standard normal distribution* $\mathcal{N}(0,1)$. *Here we are interested to know what is the cumulative density of X taking values larger than three times the standard deviation. By definition, this value is* $P(X \geq 3) = \int_3^\infty \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} dx = 0.0015$. *Estimate the value of* $P(X \geq 3)$ *using the bounds from the last three exercises and compare them against the value obtained above. (Recall that* $\mathcal{N}(0,1)$ *is symmetric. Evaluation of* $P(X \geq 3) \leq \frac{E[e^{tX}]}{e^{3t}}$ *will require some work, but this quantity is minimized when* $t = 3$. *)*

**2.24** *For a fair coin, let Y be the indicator r.v. denoting whether the outcome of toss is a Head (*$Y = 1$*) or Tail (*$Y = 0$*). Note that* $E[Y] = 1/2$ *and* $V[Y] = 1/4$. *Suppose we toss a fair coin n times and record the total number of heads obtained in a r.v.* $S \in \{0, \ldots, n\}$. *By the Central Limit Theorem, for large values of n, we know that* $T_n = \frac{S - n\mu}{\sqrt{n}\sigma}$ *has a standard normal distribution* $\mathcal{N}(0,1)$. *For* $n = 100$, *we have that* $E[S] = 50$ *and* $V[S] = 25$. *What is the probability that (a)* $45 \leq S \leq 55$, *(b)* $40 \leq S \leq 60$, *and (c)* $35 \leq S \leq 65$*? (The answers should correspond to probability density of one, two, and three standard deviations around the mean for a Normal Distribution, respectively.)*

# 3
# *Introduction to Graphs*

## *3.1 Introduction and Definitions*

Graphs were discovered by Euler (Königsberg bridge problem)[1], Kirchoff (electrical networks)[2] and Cayley (enumeration of organic chemical isomers) [3] in different contexts. Graphs are combinatorial structures used in computer science. Lists, Trees, Directed Acyclic Graphs, Flow Charts, Control Flow Graphs, Planar Graphs, web, unit disk graphs, and communication networks are examples of graphs that are widely used in computer science. Most often, practical problems, can be cast into some sort of graph problem. Examples include the Traveling Salesperson problem (finding a route of the cheapest cost through many cities), or coloring a map so that no two neighboring countries receive the same color or finding shortest path from Carleton to National Art Gallery, or navigating hyperlinks in webpages. There are excellent books and thousands of papers discussing several aspects of graphs (definitions, connectivity, coloring, independent sets, matchings, Kuratowski's theorem, four color theorem, ...). Some of the classical books include [45, 11, 12, 67]. We need to get used to some of the basic definitions.

[1] Leonhard Euler, 1707-1783

[2] Gustav Kirchoff, 1824-1887: At every node in an electrical circuit the sum of all currents should be equal to zero, i.e., the charge cannot accumulate at a node - or what comes in must go out!

[3] Arthur Cayley, 1821-1895

*Graph* A graph $G = (V, E)$ consists of a finite set of *vertices* $V$ and a finite set of *edges* $E$. See Figure 3.1 for an illustration.

- *Undirected graph*: $E$ is a set of unordered pairs of vertices $\{u, v\}$ where $u, v \in V$ (see Figure 3.1).
- *Directed graph*: $E$ is a set of ordered pairs of vertices $(u, v)$ where $u, v \in V$ (see Figure 3.2).

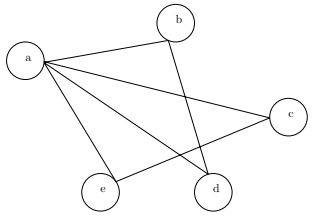



Figure 3.1: An example of a undirected graph. The edge $\{a, b\}$ is incident to the vertex labelled $a$ and to the vertex labelled $b$. The degree of vertex $a$ is 4, degree of vertex $b$ is 2. A path from the vertex $b$ to the vertex $e$ consists of edges $< bd, da, ae >$. This graph is connected and has only one connected component. This graph is simple.

*Incidence* An edge $\{u, v\}$ is *incident* to $u$ and $v$.

*Degree* of vertex in undirected graph is the number of edges incident to it.

*In (Out) degree* of a vertex in directed graph is the number of edges entering (or leaving) it.

*Path* A *path* from $u$ to $v$ is a sequence of vertices $< u = v_0, v_1, v_2, \cdots, v_k = v >$ such that $(v_i, v_{i+1}) \in E$ (or $\{v_i, v_{i+1}\} \in E$)

- We say that $v$ is reachable from $u$
- The *length* of the path is $k$
- It is a *cycle* if $u = v$

*Connected* An undirected graph is connected if every pair of vertices are joined by a path.

*Component* The connected components are the equivalence classes of the vertices under the "reachability" relation.

*Strongly Connected* A directed graph is *strongly connected* if every pair of vertices are reachable from each other. See Figure 3.2 for an illustration.

*Strongly connected components* The *strongly connected components* are the equivalence classes of the vertices under the "mutual reachability" relation.

*Simple connected undirected graph* An undirected connected graph is called simple, if between every pair of vertices there is at most one edge, and no vertex contains a self loop (i.e. a vertex connected to itself by an edge). In this course, a graph specified without any qualifications is an undirected, connected, and a simple graph!

*Complete Graphs* An undirected graph is called a complete graph, if every pair of (distinct) vertices are joined by an edge. Examples include $K_1$ (just a single vertex), $K_2$ (a pair of vertices joined by an edge), $K_3$ (a triangle), $K_5$ (graph on five vertices), ... $K_5$ is the smallest (in terms of vertices) non-planar graph (i.e. no matter how one draws it in the plane, there is a crossing). See Figure 3.3.

*Bipartite Graphs* A graph is called bipartite if the vertex set $V$ can be partitioned into two subsets $S \cup T = V$, such that for any edge $\{a, b\}$, $a \in S$ and $b \in T$. A bipartite graph is complete if every vertex in $S$ is connected to each vertex in $T$ by an edge. For example, $K_{mn}$ refers to a complete bipartite graph consisting of vertices $V = S \cup T$, where $|S| = m$ and $|T| = n$. Interestingly $K_{3,3}$ is the smallest graph (in terms of edges) which is non-planar.

*Kuratowski's Theorem* A graph is planar if and only if it has no subgraph homeomorphic to $K_5$ or $K_{3,3}$. [4] Two graphs are homeomorphic if both can be obtained from the same graph by a sequence

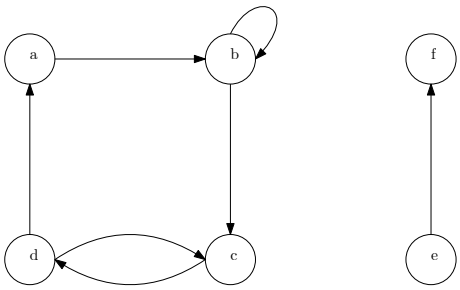
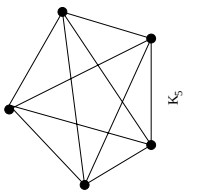



Figure 3.2: An example of a directed graph. This graph is not strongly connected since there is no way to reach from the vertex labelled *a* to the vertex labelled *e*. The strongly connected components are {{a,b,c,d},{e},{f}}.



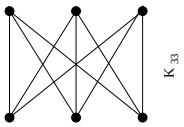




Figure 3.3: $K_{33}$ and $K_5$.

[4] This is one of the fundamental theorems in Graph Theory.

of subdivisions of edges (insertion of a vertex on an edge). For example any two cycles are homeomorphic.

## 3.2 *How to represent graphs in a computer?*

There are two standard ways of representing graphs in computers: Adjacency list and Adjacency Matrix. Let $G = (V, E)$ be the graph under consideration (assume that it is undirected - for directed the same representation works as well).

In the *adjacency matrix* representation of a graph $G = (V, E)$, we form a $|V| \times |V|$ matrix $A$ of 0s and 1s, where the $A[i, j]$-th entry is 1 if and only if there is an edge from the vertex $v_i$ to the vertex $v_j$. Formally,

$$A[i, j] = \begin{cases} 1 & v_i v_j \in E \\ 0 & v_i v_j \notin E \end{cases}$$

It is easy to see that this matrix will be symmetric for undirected graphs. Also given a pair of vertices $v_i$ and $v_j$, it takes constant time to check whether there is an edge joining them by inspecting the $ij$-th entry in the matrix $A$. Moreover, this representation is independent of the number of edges in $G$. The main drawback is that this representation requires $O(|V|^2)$ memory space whereas the graph $G$ may have very few edges! Just for fun and to get some insight, try to see what it means by taking products $A \times A$ or $A \times A \times A$,..., where the '.' refers to the boolean AND and '+' refers to the boolean OR? Try it for small graphs. We will come back to that later.

The other most common representation is the *adjacency list* representation. The adjacency list for a vertex $v$ is a list of all vertices $w$ that are adjacent to $v$. To represent the graph we have in all $|V|$ lists, one for each vertex. This representation requires optimal storage, i.e. $O(|V| + |E|)$. But to check whether the two vertices $v$ and $w$ are connected, we need to check in lists of $v$ (or $w$) whether the vertex $w$ (resp. $v$) is present. Searching in a list requires, in the worst case, time proportional to the size of the list, and hence searching will require $O(\min\{\text{degree}(v), \text{degree}(w)\})$ time.

## 3.3 *Graph traversal*

Once we have a graph, represented inside the computer, what do we do with it? When we go to a new city, what is the best way to explore all the bars? What is the best way to search for a particular web page just following the hyperlinks (assume no search engine and assume that we can go from any web page to any other web page)? How to solve those maze problems?

- There are two standard and simple ways of traversing all vertices/edges in a graph in a systematic way
  - Breadth-first search (bfs)
  - Depth-first search (dfs)
- They are used in many fundamental algorithms as a preprocessing step.

### 3.3.1 *Breadth-first search (BFS)*

- Main idea of breadth-first search is (see Figure 3.4):
  - Start at a source vertex and visit,
  - All vertices at distance 1 (i.e. vertices that are neighbors of source),
  - Followed by all vertices at distance 2 (neighbors of neighbors of source),
  - Followed by all vertices at distance 3 (neighbors of neighbors of neighbors of source),
  - ⋮
- BFS corresponds to computing *shortest path* distance (number of edges) from $s$ to all other vertices in the graph.

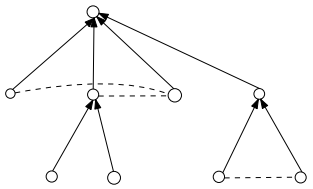


Figure 3.4: Illustration of BFS. Solid edges are tree edges and dashed edges are non-tree edges.

See Algorithm 3.1 for details. It is easy to see that Algorithm 3.1 runs in $O(|V| + |E|)$ time since each vertex is inserted (enqueued) once in the queue Q and each edge $\{x, y\}$ is explored twice, once when the vertex $x$ is dequeued from Q and once when the vertex $y$ is dequeued. Insertion and Deletion of a vertex in Q can be achieved in constant time since Q is a FIFO Queue, and can be maintained as a doubly-connected list. Also adjacency list representation will suffice and the correctness is left as an assignment problem. We can summarize this as follows:

**Theorem 3.3.1** *Let $G = (V, E)$ be a simple graph. A breadth-first search traversal of $G$, and its corresponding tree, can be computed in $O(|V| + |E|)$ time.*

## 3.4 *Topological sort and DFS*

### 3.4.1 *Topological Sort*

Let $G = (V, E)$ be a directed acyclic graph (DAG) on the vertex set $V$ with directed edge set $E$. In a DAG, there are no directed cycles.

**Algorithm 3.1:** Breadth-First Search
**Input:** Graph $G = (V, E)$ and a source vertex $v \in V$
**Output:** A breadth-first search tree $T$ rooted at $v$. Each vertex $u$ stores its parent $p(u)$ in $T$.

```
1  Initialize a Queue Q of vertices, maintained in First-In-First-Out
     order.
2  T, Q ← ∅;
3  mark[v] ← visited;
4  Q ← v;
5  while Q ≠ ∅ do
6      x ← FRONT(Q);
7      REMOVE(x, Q); // Remove x from Q.
8      foreach y ∈ V adjacent to x do
9          if mark[y] = unvisited then
10             mark[y] ← visited;
11             Insert y at the END of Q;
12             Insert the directed edge (x, y) in the tree T, where
                 p(y) ← x;
13         end
14     end
15 end
```

But, between a pair of nodes, there may be multiple directed paths. A *topological sort* of a DAG is a linear ordering of all its vertices such that if $G$ contains a directed edge $(u, v)$, then $u$ appears before $v$ in the ordering. One can think of this process as assigning a number $f : V \to \{1, \cdots, |V|\}$ to each vertex such that for every directed edge $(u, v)$, $f(u) < f(v)$ (see Figure 3.5).

We will sketch two algorithms, first a slower one followed by an optimal one. You need to verify the correctness as well as the complexities of both of them. Try to decide yourself what should be a suitable graph representation for these algorithms.

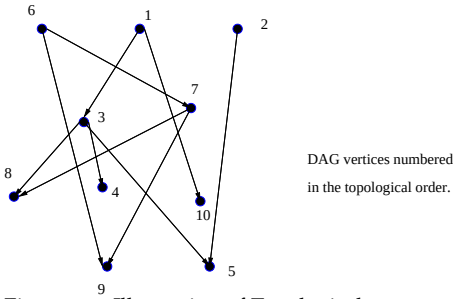



Figure 3.5: Illustration of Topological Sort

Algorithm 1: Topological sort in $O(|V|^2)$ time.

*Step 1:* Start from any vertex and follow edges backwards until a vertex $v$ is found, such that $v$ has no incoming edges.

*Step 2:* Make $v$ the next vertex in the total order.

*Step 3:* Delete $v$ and all of its outgoing edges.

*Step 4:* If the graph is non-empty, go to Step 1.

Observe that in Step 1 we will find a vertex $v$, having no incoming edges, as DAGs are acyclic and have a finite number of vertices. Moreover a vertex is assigned a number when it has no incoming edges. This should be sufficient to prove that the generated linear ordering of vertices satisfy the requirements of topological sort.

Algorithm 2: Topological sort in $O(|V| + |E|)$ time using adjacency list representation, where for each vertex maintain a separate list of incoming edges and outgoing edges.

*Step 1:* Form a queue $Q$ of vertices which have no incoming edges.

*Step 2:* Pick a vertex $v$ from $Q$, and make $v$ the next vertex in the order.

*Step 3:* Delete $v$ from $Q$ and delete all of its outgoing edges. Let $(v, w)$ be an outgoing edge. If the list of incoming edges of $w$ becomes empty then insert $w$ in $Q$.

*Step 4:* If $Q$ is not empty then GOTO Step 2.

Which invariant(s) are maintained by Algorithm 2? Why is it correct? Why does it run in $O(|V| + |E|)$ time?

### 3.4.2 *Depth First Search*

Depth First Search (DFS) is another way of exploring a graph. Like BFS, DFS traversal will take linear time, will produce a DFS spanning tree and this tree will possess very interesting, useful and beautiful properties.

Assume that we have an undirected connected simple graph $G = (V, E)$. Informally DFS on $G$ performs the following steps:

1. Select a vertex $v$ of $G$ which is initially unvisited.
2. Make $v$ visited.
3. Each unvisited vertex adjacent to $v$ is searched in-turn using DFS recursively.

DFS partitions the edges in $G$ into two sets, the set of DFS spanning tree edges, say $T$, and the set of back edges, say $B$, where $E = T \cup B$, and $T \cap B = \emptyset$. Next we formally describe the algorithm of Aho, Hopcroft, Ullman [5] for DFS. Each vertex in $G$ will be assigned a DFS-number, i.e., the order in which they are first visited in the DFS (see Figure 3.6).

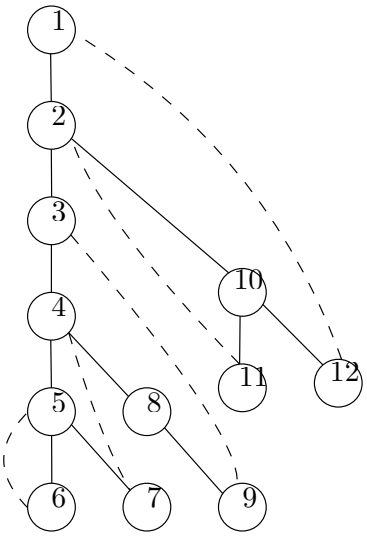



Figure 3.6: Illustration of dfs and low-numbers. Solid edges are tree edges and dashed edges are back edges. $low(6) = 5$, $low(4) = 3$, $low(10) = 1$, $low(3) = 3.$, $; ow(2) = 1$.

[5] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, 1974

DFS Algorithm

Input: A (undirected simple connected) graph $G = (V, E)$, represented by adjacency list $L[v]$ for each vertex $v \in V$.
Output: Partition of $E = T \cup B$. Tree edges are given as directed edges from a child to its parent. All edges not in $T$ are considered to be in $B$. DFS-number, an integer in the range $1..|V|$, assigned to each vertex.

1. $T := \emptyset$; COUNT:=1;
2. for all $v \in V$ do mark $v$ as *unvisited*;
3. while there exists an unvisited vertex do SEARCH(v)

procedure SEARCH(v)

1. mark $v$ as *visited*;
2. DSF-number[v]:=COUNT;
3. COUNT:=COUNT+1;
4. for each vertex $w$ on $L[v]$ do
   if $w$ is unvisited then
   (a) add $(w, v)$ to $T$; /*edge $(w, v)$ is a DFS tree edge */
   (b) SEARCH(w); /* Recursive call */

*Complexity Analysis:* Why does the above algorithm runs in $O(|V| + |E|)$ time?
We call the procedure SEARCH(v), $|V|$ times, once for each vertex. The total running time of SEARCH(v), exclusive of the recursive calls, is proportional to the degree of $v$. Hence the total time complexity is $O(|V| + \sum_{v \in V}(degree(v))) = O(|V| + |E|)$.

*Property of Back edges:* If $\{w, v\} \in B$ is a back edge, then either $w$ is an ancestor of $v$ or $v$ is an ancestor of $w$ in the DFS tree $T$. Why?
Suppose, without loss of generality, $v$ has a lower DFS-number than $w$, i.e., the vertex $v$ is visited before the vertex $w$. Therefore, when SEARCH(v) is invoked, the vertex $w$ is labeled *unvisited*. All the *unvisited* vertices visited by SEARCH(v) will become descendants of $v$ in the DFS tree. Therefore, $w$ will become descendant of $v$, since $w \in L[v]$ and each vertex in $L[v]$ is looked at while executing SEARCH(v).

### 3.4.3 *Computation of $low(v)$*

We introduce a quantity, called $low(v)$, for each vertex $v \in V$ with respect to the DFS tree $T$ and the back edges $B$. This quantity will be used in checking whether a graph is biconnected and finding its biconnected components. We will deal with biconnectivity in the next section.

Let us first define $low(v)$. Relabel the vertices of $G$ by their DFS-numbers. For each vertex $v \in V$, define $low(v)$ as follows:

$low(v) = MIN(\{v\} \cup \{w|$there exists a back edge $(x, w) \in B$ such that $x$ is a descendant of $v$ and $w$ is an ancestor of $v$ in the DFS tree$\})$

Intuitively, $low(v)$ is trying to capture the following. Consider the subtree $T_v$ of the DFS-tree $T$, rooted at the vertex $v$. What is the vertex closest to the root of $T$ which we can reach by using back edges emerging in $T_v$, and going to the ancestors of $v$? If there are no back edges going out of $T_v$, then $low(v) = v$; otherwise it is the minimum (i.e. closest to the root) among the set of ancestors of $v$, which are joined by back edges from the vertices in $T_v$.

To compute $low(v)$, we will compute three quantities. These quantities can be computed by simple modification to the DFS algorithm. The three quantities are

1. $w = v$; i.e. the case when there are no back edges going out of the subtree $T_v$.

2. $w = low(c)$ and $c$ is a child of $v$; i.e. the case when $low(v)$ is the same as $low$ value of one of its children.

3. $(v, w)$ is a back edge in $B$; i.e. the back edges associated to vertex $v$ itself.

Then, the $low(v)$ value is given by

$low(v) = MIN(\{v\} \cup \{low(c)|$ c is a child of $v\} \cup \{w|(v, w) \in B\})$.

The modified SEARCH(v) procedure that computes the low values is as follows:

```
procedure SEARCH(v)

1. mark v as visited;

2. DSF-number[v]:=COUNT;

3. COUNT:=COUNT+1;

4. low(v):=DFS-number[v]; /* low(v) is at least equal to the DFS-
   number of v */
```

5. for each vertex $w$ on $L[v]$ do
   if $w$ is unvisited then

   (a) add $(w, v)$ to $T$; /*edge $(w, v)$ is a DFS tree edge */

   (b) SEARCH(w); /* Recursive call */

   (c) $low(v) := \min(low(v), low(w))$ /*Compare the low value of $v$ with its child $w$ */

   else if $w$ is not the parent of $v$ then
   $low(v) := \min(low(v), \text{DFS-number}[w])$; /* $(v, w)$ is a back edge */

Given that the DFS algorithm runs in $O(|V| + |E|)$ time, it is easy to see that this algorithm runs within the same time complexity.

## 3.5 Biconnectivity

### 3.5.1 Equivalence Relation

Before we talk about biconnectivity, we need to recall what is an equivalence relation.

*Relation* Let $A$ and $B$ be finite sets. A binary relation $R$ from $A$ to $B$ is a subset of the cross product of $A$ and $B$, i.e. $R \subseteq A \times B$. A relation on a set $A$ is a relation from $A$ to $A$.
Example: Let $A = \{1, 2, 3, 4\}$. Let $R = \{(a, b) | a \text{ divides } b, \text{ where } a, b \in A\}$, i.e. $R = \{(1, 1), (2, 2), (3, 3), (4, 4), (1, 2), (1, 3), (1, 4), (2, 4))\}$.

*Reflexive* A relation $R$ on $A$ is reflexive if $(a, a) \in R$ for every element $a \in A$. The relation in the divide example is reflexive.

*Symmetric* A relation $R$ on $A$ is called symmetric if $(b, a) \in R$ whenever $(a, b) \in R$, where $a, b \in A$. The relation in the divide example is not symmetric.

*Transitive* A relation $R$ on $A$ is called transitive if whenever $(a, b) \in R$ and $(b, c) \in R$, then $(a, c) \in R$, for $a, b, c \in A$. The relation in the divide example is transitive.

*Equivalence Relation* A relation on a set $A$ is an equivalence relation if it is reflexive, symmetric, and transitive.

*Equivalence Classes* Let $R$ be an equivalence relation on a set $A$. The set of all elements that are related to an element $a \in A$ is called the equivalence class of $a$, denoted by $[a]$.

*Property of Equivalence Classes* Let $R$ be an equivalence relation on $A$. Then if $(a, b) \in R$, then $[a] = [b]$.

*Partition of A* Let $R$ be an equivalence relation on $A$. Then the equivalence classes of $R$ form a partition of $A$.

Example: Let $R$ be a relation on the set of integers such that $(a, b) \in R$ if and only if $a = b$ or $a = -b$. The equivalence class of integer 4 will be $[4] = \{4, -4\}$. Similarly, $[7] = \{-7, 7\}$. Observe that since $(4, -4) \in R$, then $[4] = [-4] = \{4, -4\}$. Also observe that the set of integers can be partitioned by $R$ as follows: $\{(-1, 1), (0), (-2, 2), (-3, 3), ...\}$.

There are many books in Discrete Mathematics discussing equivalence relations, for example, see Rosen [6].

[6] Kenneth H. Rosen. *Discrete Mathematics and Its Applications*. McGraw-Hill Higher Education, 5th edition, 2002

### 3.5.2 *Biconnectivity*

Most of the material in this section is from Kozen [7] and Aho, Hopcroft and Ullman [8]. Assume that the graph $G = (V, E)$ is undirected and connected. We start with definitions.

[7] D. Kozen. *The design and analysis of algorithms*. Springer, 1992

[8] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, 1974

*Articulation vertex* A vertex $v \in V$ is called an articulation vertex, if its removal disconnects the graph. Equivalently, vertex $v$ is an articulation vertex if there exists vertices $a$ and $b$, so that every path between $a$ and $b$ goes through $v$ and $a, b$, and $v$ are all distinct.

*Biconnected* A connected graph is biconnected if any pair of distinct vertices lie on a simple cycle (one with no repeated vertices). Equivalently, for every distinct triple of vertices $v, a$, and $b$, there exists a path between $a$ and $b$ not containing $v$. Observe that $G$ is biconnected if and only if it has no articulation vertices. Note that a graph just consisting of a single edge (two vertices joined by an edge) is biconnected!

*Relation on edges* For edges $e, e' \in E$, define that the two edges are equivalent, $e \equiv e'$, if $e$ and $e'$ lie on a simple cycle.

**Lemma 3.5.1** *The relation $\equiv$ defined above is an equivalence relation.*

**Proof.** To prove that it is an equivalence relation, we need to show that it is reflexive, symmetric and transitive.

*Reflexive:* Obviously $e \equiv e$, $\forall e \in E$, since an edge by itself lies on a cycle.

*Symmetric:* If $e \equiv e'$, then $e' \equiv e$, since they are on the same cycle.

*Transitivity:* Suppose that $e \equiv e'$ and $e' \equiv e''$. We want to show that $e \equiv e''$. Say $e$ and $e'$ are on a simple cycle $X$ and $e'$ and $e''$ are on a simple cycle $Y$. If $e''$ is also on the cycle $X$ then we are done because $e$

is on that cycle too. Otherwise, follow the edges of $Y$ from one end of $e''$ until it hits $X$. Call that intersection vertex $p$. Do the same for the other end of $e''$, and call that intersection vertex $q$. Now $p$ and $q$ are distinct and there are two disjoint paths between them using edges of $X$. One of these paths contains edge $e$. Combing this path with the ones of $Y$ we used to get from $e''$ to $X$ gives us a simple cycle containing both $e$ and $e''$. [9]

[9] Thanks to Danny Sleator for providing this proof.

■

Now we have an equivalence relation. What are the equivalence classes of this relation with respect to the edge set of $G$.

*Biconnected Components* The equivalence classes of the relation $\equiv$ are the biconnected components of $G$.

Now we discuss critical lemmas, which relate articulation vertices, biconnected components, DFS number and low values. These lemmas are '*if and only if*' type - or '*necessary and sufficient*' type. Such types of lemmas are extremely useful, especially in computer science, since they provide a complete characterization of the object/structure under consideration, and often lead to an algorithm.

**Lemma 3.5.2** *A vertex $v$ is an articulation vertex if and only if it is contained in at least two distinct biconnected components.*

**Proof.** Suppose $v$ is an articulation vertex. Then its removal disconnects $G$. That means that there are two vertices $a$ and $b$ neighboring $v$ so that each path between $a$ and $b$ goes through $v$. See Figure **??**. Then the edges $(a, v)$ and $(v, b)$ cannot lie on a simple cycle, and hence they belong to two distinct biconnected components. This implies that $v$ is contained in at least two distinct biconnected components.

Now suppose that $v$ is contained in two distinct biconnected components, and is adjacent to vertices $a$ and $b$ in these components, respectively. Then $(va) \not\equiv (vb)$. Then all paths between $a$ and $b$ goes through $v$, and hence removing $v$ disconnects $G$. So $v$ is an articulation vertex. ■

**Lemma 3.5.3** *Let $(uv)$ and $(vw)$ be two adjacent tree edges in a DFS tree $T$ of $G$. Then $(uv) \equiv (vw)$ if and only if there exists a back edge from some descendant of $w$ to some ancestor of $u$.*

**Proof.** Recall that the descendants of $w$ are $w$ and all the vertices in the subtree rooted at $w$. Ancestors of $u$ include $u$ and all vertices on the path from $u$ to the root of the DFS tree.

If there exists a backedge from some descendant $w'$ of $w$ to some ancestor $u'$ of $u$, then $(uv) \equiv (vw)$, since there is a simple cycle

consisting of the tree path between $u'$ and $w'$ and the backedge $w'u'$. See Figure **??**.

Suppose $(uv) \equiv (vw)$. By definition, there is a simple cycle that contains both of them. The edges $(uv)$ and $(vw)$ must appear in this order in the cycle, as the vertex $v$ appears exactly once on the cycle. This implies that there is an edge (actually a backedge) from some vertex $w'$ in the subtree rooted at $w$ to some ancestor $u'$ of $u$. (This ancestor could be the vertex $u$ or a vertex on the path from $u$ to the root of the DFS tree.) ∎

**Lemma 3.5.4** *Vertex $v$ is an articulation vertex if and only if either*

*(a) $v$ is the root of the DFS tree and has more than one child.*

*(b) $v$ is not the root, and for some some child $w$ of $v$ there is no backedge between any descendant of $w$ (including $w$) and a proper ancestor of $v$.*

Part (a) is easy to prove and part (b) follows from the previous lemma! The modifications to the DFS procedure to compute the biconnected components are as follows:

See Figure 3.7 for an illustration for the biconnected components computed by the following search procedure.

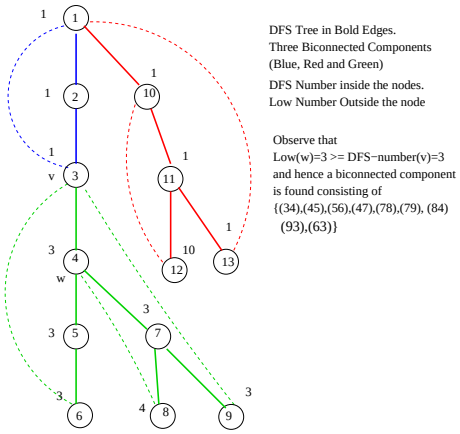



Figure 3.7: Illustration of biconnected components

procedure SEARCH(v)
begin

1. mark $v$ as *visited*;
2. DSF-number[$v$]:=COUNT;
3. COUNT:=COUNT+1;
4. $low(v)$:=DFS-number[$v$]; /* $low(v)$ is at least equal to the DFS-number of $v$ */
5. for each vertex $w$ on $L[v]$ do
   if $w$ is unvisited then
   (a) add $(w, v)$ to $T$; /*edge $(w, v)$ is a DFS tree edge */
   (b) SEARCH($w$); /* Recursive call */
   (c) If $low(w) \geq$ DFS-number$[v]$ then a biconnected component has been found;
   (d) $low(v) := \min(low(v), low(w))$ /*Compare the low value of $v$ with its child $w$ */

   else if $w$ is not the parent of $v$ then
   $low(v) := \min(low(v)$,DFS-number$[w])$; /* $(v, w)$ is a back edge */

end

Why does the above algorithm compute biconnected components? Actually we will need a STACK to figure out the edges in a biconnected component! How do we do that? When a vertex $w$ is encountered in the SEARCH procedure, put the edge $(v, w)$ on the STACK if it is not there. After discovering a pair $(v, w)$, such that $w$ is a child of $v$ and $low(w) \geq$ DFS-number$[v]$, POP all the edges from the STACK up to and including $(v, w)$. These edges form a biconnected component. This extra step can be accomplished in linear time as well. To prove that the above SEARCH procedure indeed computes the biconnected components, we need to argue by induction on the number of biconnected components.

## *3.6 Exercises*

**3.1** *This problem is related to the representation of graphs. Assume that the number of edges in the graph $G = (V, E)$ is small, i.e., it is a sparse graph. In the adjacency matrix representation of G, the normal tendency is to first initialize the matrix, requiring $O(|V|^2)$ time. Is there any way we can initialize the adjacency matrix in time proportional to $O(|E|)$ and still have $O(1)$ adjacency test?*

**3.2** *Provide a clear, concise, and complete proof for the correctness of the BFS algorithm.*

**3.3** *Let G=(V,E) be a directed acyclic graph with two designated vertices, the start and the destination vertex. Write an algorithm to find a set of paths from the start vertex to the destination vertex such that (a) no vertex other than the start or the destination vertex is common to two paths. (b) no additional path can be added to the set and still satisfy the condition (a). Note that there may be many sets of paths satisfying the above conditions. You are not required to find the set with the most paths but any set satisfying the above conditions. Your algorithm should run in $O(|V| + |E|)$ time. State the algorithm, its correctness and analyze the complexity.*

**3.4** *Clearly describe the modifications you need to make in the SEARCH procedure to compute and output the biconnected components. Prove that your algorithm is correct, i.e. it computes all the biconnected components.*

**3.5** *How can we find in $O(|V| + |E|)$, whether a graph $G = (V, E)$ is a bipartite graph (Hint: Use BFS).*

**3.6** *An Euler circuit for an undirected graph is a path that starts and ends at the same vertex and uses each edge exactly once (vertices might be repeated). A connected, undirected graph G has an Euler circuit if and only if every vertex is of even degree. Give an $O(|E|)$ algorithm to find an Euler circuit in G, if it exists.*

**3.7** *Assume that you are given $n$ positive integers, $d_1 \geq d_2 \geq \cdots \geq d_n$, each greater than 0. You need to design an algorithm to test whether these integers form the degrees of an $n$ vertex simple undirected graph $G = (V, E)$ (Think of a greedy algorithm.)*

**3.8** *Show that in a depth-first search, if we output a left parenthesis '(' when a node is accessed for the first time and output a right parenthesis ')' when a node is accessed for the last time, then resulting parenthesization (or bracketing sequence) is proper. Each left '(' is properly matched with each right ')'.*

**3.9** *Assume that $G = (V, E)$ is biconnected. Our task is to identify those edges $E' \subseteq E$, so that if we remove any edge $e \in E'$ from $G$, then the resulting graph is not biconnected. Intuitively, edges in $E'$ are essential in maintaining the biconnectivity of $G$. It is fairly straightforward to test whether an edge $e \in E$ is critical, by just removing $e$ from $G$ and running the biconnectivity algorithm to test whether the resulting graph is biconnected. A question worth trying, but is likely to be nontrivial, is to compute the set $E' \subseteq E$ in $o(|E|(|V| + |E|))$ time.*

# 4
# *Matrices with Applications to CS*

— Under Construction —

## *4.1 Basics*

Let $A$ be a matrix consisting of 3 rows and 3 columns on real numbers. We view each row or column as a vector in $\mathbb{R}^3$. For example, let

$$A = \begin{bmatrix} 2 & 2 & 0 \\ 2 & 4 & 8 \\ 10 & 16 & 24 \end{bmatrix}$$

The three rows are $r_1 = (2, 2, 0)$, $r_2 = (2, 4, 8)$, and $r_3 = (10, 16, 24)$. The three columns are $c_1 = (2, 2, 10)$, $c_2 = (2, 4, 16)$, and $c_3 = (0, 8, 24)$. All of them are vectors in $\mathbb{R}^3$. We further observe that $r_3 = 2r_1 + 3r_2$. (Two typical operations on vectors include scalar multiplication and vector addition. In a scalar multiplication of a vector $v = (2, 3, 7)$ by a scalar $c = 3$, we obtain $cv = (6, 9, 21)$. In vector addition of two vectors $u = (1, 3, -5)$ and $v = (-6, 4, 1)$, we obtain $u + v = (-5, 7, -4)$.) Let us find the row reduced echelon form (RREF) of $A$ by finding its pivots. We will denote the entry in the $i$-th row and the $j$-th column of $A$ by $a_{ij}$. Since $a_{11} \neq 0$, it is the first pivot. Now we take the suitable multiples of $r_1$ and subtract them from $r_2$ and $r_3$ so that the only non-zero entry that remains in the first column is in the first row. This can be achieved by replacing $r_2$ by $r_2 - r_1$, and $r_3$ by $r_3 - 5r_1$, and we obtain the following matrix:

$$\begin{bmatrix} 2 & 2 & 0 \\ 0 & 2 & 8 \\ 0 & 6 & 24 \end{bmatrix}$$

Next we find the second pivot. The entry in 2nd row and 2nd column is non-zero, hence that is the pivot. We replace $r_3$ by $r_3 - 3r_2$ and

obtain

$$\begin{bmatrix} 2 & 2 & 0 \\ 0 & 2 & 8 \\ 0 & 0 & 0 \end{bmatrix}$$

The last row only contains zero's. Therefore, $A$ doesn't have a non-zero third pivot. To obtain the RREF, we will like the pivots to be 1, and moreover the sub-matrix consisting of the pivot rows and columns to be the identity matrix. To obtain the RREF form, we divide the first row by 2, the second row by 2, and obtain

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 4 \\ 0 & 0 & 0 \end{bmatrix}$$

We are almost there, except that the sub-matrix formed by the first two rows and first two columns is not an identity matrix. To obtain the RREF, we replace $r_1$ by $r_1 - r_2$ and obtain

$$R = \begin{bmatrix} 1 & 0 & -4 \\ 0 & 1 & 4 \\ 0 & 0 & 0 \end{bmatrix}$$

Next, we will make several observations on $A$ and its RREF $R$. (For a deeper understanding on various properties of $A$ and $R$, refer to the textbook of Gilbert Strang [1].) Since the number of non-zero pivots is 2, the rank $r$ of $A$ is $r = 2$. Moreover, the dimension of its row space is $r = 2$ (row space is the vector space consisting of all linear combinations of row vectors). The basis vectors of the row space are the rows corresponding to the non-zero pivots in $R$, i.e. $v_1 = \begin{bmatrix} 1 \\ 0 \\ -4 \end{bmatrix}$ and $v_2 = \begin{bmatrix} 0 \\ 1 \\ 4 \end{bmatrix}$. Similarly, the column space, i.e. the vector space formed by linear combinations of the columns of $A$ has dimension $r = 2$. Its basis vectors are the columns of $A$ corresponding to the non-zero pivots. In our example, it will be the first and the second column of $A$, i.e. $u_1 = \begin{bmatrix} 2 \\ 2 \\ 10 \end{bmatrix}$ and $u_2 = \begin{bmatrix} 2 \\ 4 \\ 16 \end{bmatrix}$. respectively. Interestingly, now $A$ can be expressed as the sum of its rank 1 components as follows:

$$A = u_1 v_1^T + u_2 v_2^T = \begin{bmatrix} 2 \\ 2 \\ 10 \end{bmatrix} \begin{bmatrix} 1 & 0 & -4 \end{bmatrix} + \begin{bmatrix} 2 \\ 4 \\ 16 \end{bmatrix} \begin{bmatrix} 0 & 1 & 4 \end{bmatrix}$$

Next we discuss briefly the null spaces of $A$. There is a column null space (usually referred to as the null space) and there is a left null space. The null space of $A$ represents all vectors $x$, such that $Ax = 0$. In other words what linear combinations of the vectors corresponding to the columns will result in a $o$ vector. For our example, $x \in \mathbb{R}^3$, and the vector $[0, 0, 0]$ is in the null space. Are there

[1] Gilbert Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, Wellesley, MA, fifth edition, 2016

any non-zero vectors in null space of $A$? In other words, are there $x_1, x_2, x_3$, such that $x_1 \begin{bmatrix} 2 \\ 2 \\ 10 \end{bmatrix} + x_2 \begin{bmatrix} 2 \\ 4 \\ 16 \end{bmatrix} + x_3 \begin{bmatrix} 0 \\ 8 \\ 24 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$. We can see that $x_1 = 1$, $x_2 = -1$, and $x_3 = 1/4$, satisfies this condition. Hence the vector $(1, -1, 1/4)$, or any of its scalar multiples, is in the null-space of $A$. In fact the dimension of the null-space of $A$ is the number of its columns minus its rank. In our case, the dimension of the null-space will be $3 - 2 = 1$. Now for the left null space, we are looking for vectors $y \in \mathbb{R}^3$ such that $A^T y = 0$ (equivalently, $y^T A = 0$). This represents what linear combinations of row vectors can result in a $o$ vector. In our example, we have $\begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} \begin{bmatrix} 2 & 2 & 0 \\ 2 & 4 & 8 \\ 10 & 16 & 24 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$. Note that the vectors $y = [0, 0, 0]$, and $[2, 3, -1]$ or any of their linear combinations satisfies the left nullity conditions. The dimension of the left null space is the number of rows minus the rank of $A$, i.e. $3 - 2 = 1$ in our example.

In general, for an $m \times n$ matrix $A$, we have the following. Assume that its RREF is $R$ and it consists of $r \leq \min\{m, n\}$ non-zero pivots. Then, rank of $A$ is $r$. The column space is a subspace of $\mathbb{R}^m$ of dimension $r$, and its basis vectors are the columns of $A$ corresponding to the non-zero pivots in $R$. The row space is a subspace of $R^n$ of dimension $r$, and its basis vectors are the rows of $R$ corresponding to the non-zero pivots. $A$ can be expressed as sum of $r$ rank one matrices. The null-space of $A$ consists of all the vectors $x \in \mathbb{R}^n$ satisfying $Ax = 0$. They form a subspace of dimension $n - r$. Lastly, the left null space of $A$, i.e. all the vectors $y \in \mathbb{R}^m$ such that $A^T y = 0$ (or equivalently $yA^T = 0$, and thus the name). For the null space, we looked at which linear combinations of columns yield a zero vector. For the left null-space, we are interested to know which linear combinations of rows yield a zero vector. Its dimension is $m - r$.

The fundamental theorem of linear algebra states that the column and row spaces of $A$ have dimension $r$, and the two null spaces have dimensions $n - r$ and $m - r$, respectively. Furthermore, we can choose basis for these vector spaces in such a way that the $r$ vectors forming the basis for the column space are orthonormal, and the $n - r$ vectors forming the basis for the null space are orthonormal and they are also orthogonal to the column basis vectors. Together they form an orthonormal basis for $\mathbb{R}^n$. Analogously, we can choose $r$ orthonormal vectors forming the basis for the row space, and $m - r$ orthonormal vectors forming the basis of the left null-space and together they form an orthonormal basis for $\mathbb{R}^m$.

## 4.2 Introduction to Eigenvalues

Consider a square matrix $A$ of dimension $n \times n$ on real numbers. Let $x$ be a vector of dimension $n$. Let $A = (a_{ij})$, where $i = 1, \ldots, n$, $j = 1, \ldots, n$, $a_{ij} \in \mathbb{R}$, and let $x = (x_1, x_2, \ldots, x_n)$, $x_i \in \mathbb{R}$. Consider the product of $A$ and $x$, and we know that it results in another vector $y = (y_1, y_2, \ldots, y_n)$ of dimension $n$ such that $y_i = \sum_{j=1}^{n} a_{ij}x_j$, for $i = 1, 2, \ldots, n$. We can view $A$ as a function that transforms a vector $x$ to another vector $y$. We are in particular interested in those vectors $x$, such that the product $Ax$ results in a vector that is parallel to $x$. Clearly, if $x = 0$, it is true that $Ax = x = 0$. We are interested to know if there are non-zero vectors $x$ such that $Ax = \lambda x$, for constant $\lambda$. Such vectors are called *eigenvectors* and the corresponding constant value $\lambda$ is called the *eigenvalue*. As it will turn out that there are several applications of eigenvalue-eigenvectors in many fields of Sciences and Engineering. Let us first see a couple of examples.

**Example 4.2.1**

$$A = \begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix}$$

*Observe that*

$$\begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix} = 5 \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

*and*

$$\begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = 1 \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

*Thus, $\lambda_1 = 5$ and $\lambda_2 = 1$ are the eigenvalues of $A$ and the corresponding eigenvectors are $v_1 = [1, 3]$ and $v_2 = [1, -1]$, respectively, as $Av_1 = \lambda_1 v_1$ and $Av_2 = \lambda_2 v_2$. Note that $v_1$ is stretched five times when multiplied by $A$, whereas $v_2$ is left unchanged.*

**Example 4.2.2** *Let us consider the same example as above, but now the rows are permuted. Let*

$$B = \begin{bmatrix} 3 & 4 \\ 2 & 1 \end{bmatrix}$$

*Observe that*

$$\begin{bmatrix} 3 & 4 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = 5 \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

*and*

$$\begin{bmatrix} 3 & 4 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = -1 \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

*Thus, $\lambda_1 = 5$ and $\lambda_2 = -1$ are the eigenvalues of B and the corresponding eigenvectors are $v_1 = [2, 1]$ and $v_2 = [1, -1]$, respectively, as $Bv_1 = \lambda_1 v_1$ and $Bv_2 = \lambda_2 v_2$. Here $v_2$ flips its direction when multiplied by B, but its magnitude remains the same.*

Let us consider the eigenvalues and eigenvectors of $A^2$. Since, $Av_i = \lambda_i v_i$. Note that by multiplying by $A$ on both the sides on the left, we have

$$A^2 v_i = A(Av_i) = A(\lambda_i v_i) = \lambda(Av_i) = \lambda(\lambda v_i) = \lambda^2 v_i.$$

Thus for $A^2$, the eigenvectors are the same as that of $A$, but eigenvalues are squared. In fact for an integer $k > 0$, $A^k$ has the same eigenvectors as $A$, but the eigenvalues are $\lambda^k$.

Let us see how to compute the eigenvalues and eigenvectors of an $n \times n$ matrix $A$. We are interested to find vectors $x$ (especially the non-zero vectors) such that $Ax = \lambda x$, or equivalently,

$$(A - \lambda I)x = 0, \tag{4.1}$$

where $I$ is an $n \times n$ identity matrix. Note that all the eigenvectors $x$ that satisfy $(A - \lambda I)x = 0$ constitutes the null space of $A - \lambda I$. (Null space of a matrix $B$ is the set of all the vectors $v$ such that $Bv = 0$. Clearly $v = 0$ is in this set and this set is closed under vector addition and scalar multiplication. Hence the null space forms a vector space.) If $(A - \lambda I)x = 0$ has a non-zero solution, then the matrix $(A - \lambda I)$ is singular (i.e. it is not invertible). This implies that the determinant of $(A - \lambda I)$, $det(A - \lambda I) = 0$. The equation $det(A - \lambda I) = 0$ results in a polynomial of degree $n$, and this equation has $n$ (real or complex) roots. The polynomial $det(A - \lambda I)$ is referred to as the *characteristic polynomial*. Note that the roots of the characteristic polynomial may not be distinct (e.g. consider the eigenvalues of identity matrix).

**Example 4.2.3** *Consider the matrix A given above. Its characteristic polynomial is given by*

$$det(A - \lambda I) = det \begin{bmatrix} 2 - \lambda & 1 \\ 3 & 4 - \lambda \end{bmatrix} = (2 - \lambda)(4 - \lambda) - 3 = \lambda^2 - 6\lambda + 5$$

*Roots of $\lambda^2 - 6\lambda + 5 = (\lambda - 5)(\lambda - 1) = 0$ are $\lambda_1 = 5$ and $\lambda_2 = 1$ and are distinct.*

Next, let us see how to find the eigenvector $v$ corresponding to an eigenvalue $\lambda$. Since $v$ is in the null space of $A - \lambda I$, we solve the system of equations for $(A - \lambda I)v = 0$ to find all the components of $v$. For the above example, let us find the eigenvector $v_1 = [a, b]$ corresponding to the eigenvalue $\lambda_1 = 5$.

$$(A - \lambda_1 I)v_1 = \begin{bmatrix} 2-5 & 1 \\ 3 & 4-5 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} -3 & 1 \\ 3 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Observe that the rows in the matrix $A - \lambda_1 I$ are dependent. Now we obtain the equation $-3a + b = 0$ or equivalently $3a = b$, and thus the vector $v_1 = [1, 3]$ (or any of its scalar multiple) is an eigenvector corresponding to $\lambda_1 = 5$. Similarly we can compute that $v_2 = [1, -1]$ is an eigenvector corresponding to $\lambda_2 = 1$ by solving the following:

$$(A - \lambda_2 I)v_2 = \begin{bmatrix} 2-1 & 1 \\ 3 & 4-1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 3 & 3 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

We conclude this section with the following example of rotation matrix.

**Example 4.2.4**

$$Q = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

*Note that this matrix rotates any vector by $90^\circ$ in anticlockwise direction. Since each vector $v$ is rotated, there cannot be any non-zero real vector $v$ such that $Qv$ is parallel to $v$. The characteristic polynomial of $Q$ is $det(Q - \lambda I) = \lambda^2 + 1$. Note that this does not have real roots and thus the eigenvalues of $Q$ are imaginary numbers $\lambda_1 = i$ and $\lambda_2 = -i$. What about its eigenvectors? For that we solve*

$$(Q - \lambda_1 I)v_1 = \begin{bmatrix} -i & -1 \\ 1 & -i \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

*and obtain that $a - bi = 0$ or $a = bi$ and thus $v_1 = [i, 1]$. It is a complex eigenvector and satisfies $Qv_1 = \lambda_1 v_1$. Similarly, we have $v_2 = [1, i]$ corresponding to $\lambda_2 = -i$. Therefore, even if all the entries in a matrix are real, its eigenvalues and eigenvectors may have complex numbers.*

## 4.3 *Diagonalizing Square Matrices*

Let $A$ be an $n \times n$ real matrix with $n$ distinct eigenvalues. For such matrices, their corresponding eigenvectors are linearly independent (see exercises). Let $\lambda_1, \ldots, \lambda_n$ be the distinct eigenvalues and let $x_1, \ldots, x_n$ be the corresponding eigenvectors, respectively. Let each

$x_i = [x_{i1}, x_{i2}, \ldots, x_{in}]$. Define an eigenvector matrix $X$, where the $i$th column of $X$ is the eigenvector $x_i$, $1 \leq i \leq n$. Formally,

$$X = \begin{bmatrix} x_{11} & x_{21} & \ldots & x_{n1} \\ \vdots & \vdots & \vdots & \vdots \\ x_{1n} & x_{2n} & \ldots & x_{nn} \end{bmatrix}$$

Since eigenvectors are linearly independent, we know that $X^{-1}$ exists. Define a diagonal $n \times n$ matrix $\Lambda$ whose entries are as follows.

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & 0 & \ldots & 0 \\ 0 & \lambda_2 & 0 & \ldots & 0 \\ 0 & 0 & \lambda_3 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & 0 & \lambda_n \end{bmatrix}$$

Consider the matrix product $AX$,

$$AX = A \begin{bmatrix} x_1 & \ldots & x_n \\ & & \end{bmatrix} = \begin{bmatrix} \lambda_1 x_1 & \ldots & \lambda_n x_n \\ & & \end{bmatrix} = X\Lambda$$

Since $X^{-1}$ exists, we multiply by $X^{-1}$ on both the sides from left and obtain

$$X^{-1}AX = X^{-1}X\Lambda = \Lambda \tag{4.2}$$

and when we multiply on the right we obtain

$$AXX^{-1} = A = X\Lambda X^{-1} \tag{4.3}$$

The above two equations collectively are called the diagonalizing a matrix $A$ whose eigenvectors are linearly independent.

For our example in the last section where

$$A = \begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix}$$

with eigenvalues $\lambda_1 = 5$ and $\lambda_2 = 1$ and the eigenvectors $[1, 3]$ and $[1, -1]$ respectively, we have

$$\begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 3 & -1 \end{bmatrix} = \begin{bmatrix} 5 & 1 \\ 15 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 3 & -1 \end{bmatrix} \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix}$$

and

$$X^{-1}AX = \begin{bmatrix} 1/4 & 1/4 \\ 3/4 & -1/4 \end{bmatrix} \begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 3 & -1 \end{bmatrix} = \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix}$$

Similarly,

$$A = X\Lambda X^{-1} = \begin{bmatrix} 1 & 1 \\ 3 & -1 \end{bmatrix} \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1/4 & 1/4 \\ 3/4 & -1/4 \end{bmatrix}$$

Consider the diagonalization given by equation $A = X\Lambda X^{-1}$. Consider $A^2 = (X\Lambda X^{-1})(X\Lambda X^{-1}) = X\Lambda(X^{-1}X)\Lambda X^{-1} = X\Lambda^2 X^{-1}$. Thus, $A^2$ has the same set of eigenvectors as $A$, but its eigenvalues are squared. In general, for an integer $k > 0$, $A^k = X\Lambda^k X^{-1}$, and its eigenvectors are same as that of $A$ and its eigenvalues are raised to the power of $k$.

Let $u_0, u_1, u_2, \cdots \in \mathbb{R}^n$ are vectors and $u_{k+1} = Au_k$ for $k \geq 0$. Any vector in $\mathbb{R}^n$ can be expressed as a linear combination of the eigenvectors $x_1, x_2, \ldots x_n$. Thus, for constants $c_1, c_2, \ldots, c_n$,

$$\begin{aligned}
u_0 &= c_1x_1 + c_2x_2 + \cdots + c_nx_n \\
u_1 &= Au_0 = A(c_1x_1 + c_2x_2 + \cdots + c_nx_n) \\
u_1 &= c_1Ax_1 + c_2Ax_2 + \cdots + c_nAx_n \\
u_1 &= c_1\lambda_1x_1 + c_2\lambda_2x_2 + \cdots + c_n\lambda_nx_n \\
u_2 &= Au_1 = c_1\lambda_1Ax_1 + c_2\lambda_2Ax_2 + \cdots + c_n\lambda_nAx_n \\
u_2 &= c_1\lambda_1^2x_1 + c_2\lambda_2^2x_2 + \cdots + c_n\lambda_n^2x_n \\
&\vdots \\
u_{k+1} &= Au_k = c_1\lambda_1^kx_1 + c_2\lambda_2^kx_2 + \cdots + c_n\lambda_n^kx_n
\end{aligned}$$

Note that for large values of $k$, if any of $|\lambda_i| < 1$, $|\lambda_i|^k \to 0$.

Here is an interesting exercise from the text book of Gilbert Strang to illustrate the above concept. Define $G(n)$ for integers $n \geq 0$ as follows:

$$G(n) = \begin{cases} 0, & \text{for } n = 0 \\ 1, & \text{for } n = 1 \\ \frac{G(n-1)+G(n-2)}{2}, & \text{otherwise.} \end{cases}$$

Now, define

$$g_k = \begin{bmatrix} G_{k+1} \\ G_k \end{bmatrix}$$

We obtain

$$g_{k+1} = \begin{bmatrix} G_{k+2} \\ G_{k+1} \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 \\ 1 & 0 \end{bmatrix} g_k$$

Define,

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

The eigenvalues and eigenvectors of $A$ are $\lambda_1 = 1$ and $\lambda_2 = 1/2$ and $x_1 = (1, 1)$ and $x_2 = (1, -2)$, respectively. Thus,

$$g_k = c_1\lambda_1^k x_1 + c_2\lambda_2^k x_2 = c_1 x_1 + c_2(-1/2)^k x_2$$

For large values of $k$,

$$\lim_{k\to\infty} g_k \approx c_1 x_1$$

To find the value of $c_1$, we use that $g_k = c_1\lambda_1^k x_1 + c_2\lambda_2^k x_2$ for $k = 0$, and obtain

$$g_0 = \begin{bmatrix}1\\0\end{bmatrix} = c_1\begin{bmatrix}1\\1\end{bmatrix} + c_2\begin{bmatrix}1\\-2\end{bmatrix}$$

This gives $c_1 = 2/3$ and $c_2 = 1/3$. Thus, for large values of $k$,

$$g_k = 2/3\begin{bmatrix}1\\1\end{bmatrix}$$

and $G_k$ approaches $2/3$ for large values of $k$.

## 4.4 *Symmetric and Positive Definite Matrices*

Let $S$ be an $n \times n$ real symmetric matrix where its $ij$-th entry is identical to the $ji$-th entry for all $1 \leq i \leq n$ and $1 \leq j \leq n$, i.e. $S = S^T$. We will show that $S$ has $n$ real eigenvalues and it consists of a $n$ orthonormal eigenvectors $Q = q_1, q_2, \ldots, q_n$. Moreover, the diagonalization of $S$ is given by $S = Q\Lambda Q^T$, where $\Lambda$ is the diagonal matrix consisting of real eigenvalues on its principal diagonal and $Q$ consists of orthonormal eigenvectors as columns. First we present an example.

**Example 4.4.1** *Let $S = \left[\begin{smallmatrix}3 & 1\\ 1 & 3\end{smallmatrix}\right]$. Its characteristic equation is $\lambda^2 - 6\lambda + 8 = 0$ and the eigenvalues are $\lambda_1 = 4$ and $\lambda_2 = 2$ and the corresponding eigenvectors are $q_1 = (1/\sqrt{2}, 1/\sqrt{2})$ and $q_2 = (1/\sqrt{2}, -1/\sqrt{2})$, respectively. Note that eigenvalues are real and the eigenvectors are orthonormal. Furthermore,*

$$S = \begin{bmatrix}3 & 1\\ 1 & 3\end{bmatrix} = \begin{bmatrix}1/\sqrt{2} & 1/\sqrt{2}\\ 1/\sqrt{2} & -1/\sqrt{2}\end{bmatrix}\begin{bmatrix}4 & 0\\ 0 & 2\end{bmatrix}\begin{bmatrix}1/\sqrt{2} & 1/\sqrt{2}\\ 1/\sqrt{2} & -1/\sqrt{2}\end{bmatrix}$$

**Lemma 4.4.2** *All the eigenvalues of a real symmetric matrix $S$ are real.*

**Proof.** By definition of eigenvalues,

$$Sq = \lambda q \tag{4.4}$$

By taking the complex conjugate, we have that $\bar{S}\bar{q} = \bar{\lambda}\bar{q}$. Since we are given that $S$ is real, $S = \bar{S}$. (Note that for a complex number $a + bi$, its

complex conjugate is $a - bi$.) Using the fact that $S = S^T$, the transpose of $S\bar{q} = \bar{\lambda}\bar{q}$ is given by

$$\bar{q}^T S = \bar{q}^T \bar{\lambda} \tag{4.5}$$

We multiply by $\bar{q}^T$ on the left in Equation 4.4 and obtain $\bar{q}^T S q = \bar{q}^T \lambda q$. Similarly, we multiply by $q$ on the right in Equation 4.5 and obtain $\bar{q}^T S q = \bar{q}^T \bar{\lambda} q$. Thus, $\bar{q}^T \lambda q = \bar{q}^T \bar{\lambda} q$. This implies that $\lambda = \bar{\lambda}$ and this can only happen if $\lambda$'s are real. ∎

**Lemma 4.4.3** *The eigenvectors of a real symmetric matrix $S$ are real.*

**Proof.** Each eigenvector $q$ is a solution of the equation $(S - \lambda I)q = \lambda q$, where all elements of $S$ are real and all $\lambda$'s are real. Thus all entries in $q$ are real. ∎

**Lemma 4.4.4** *Any pair of eigenvectors of a real symmetric matrix $S$ corresponding to two different eigenvalues are orthogonal.*

**Proof.** Let $q_1$ and $q_2$ be two eigenvectors corresponding to $\lambda_1 \neq \lambda_2$, respectively. Thus, $Sq_1 = \lambda_1 q_1$ and $Sq_2 = \lambda_2 q_2$. Since $S$ is symmetric, $q_1^T S = \lambda_1 q_1^T$. Multiply by $q_2$ on the right and we obtain $\lambda_1 q_1^T q_2 = q_1^T S q_2 = q_1^T \lambda_2 q_2$. Since $\lambda_1 \neq \lambda_2$ and $\lambda_1 q_1^T q_2 = q_1^T \lambda_2 q_2$, this implies that $q_1^T q_2 = 0$ and thus the eigenvectors $q_1$ and $q_2$ are orthogonal. ∎

Using the diagonalization discussed in the previous section, the matrix $S$ can be expressed as $S = Q\Lambda Q^{-1}$, where $Q$ is the matrix of orthonormal eigenvectors (see Equation 4.3). Equivalently,

$$S = Q\Lambda Q^{-1} = \lambda_1 q_1 q_1^T + \lambda_2 q_2 q_2^T + \cdots + \lambda_n q_n q_n^T.$$

For our example we will have

$$S = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} = 4 \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} + 2 \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}$$

The above representation enables us to express any symmetric matrix as the sum of rank one matrices $\lambda_i q_i q_i^T$. In the next section, we will have a similar way of expressing any rectangular matrix.

A symmetric matrix $S$ is said to be *positive definite* if all its eigenvalues $> 0$. It is called *positive semi-definite* if all eigenvalues are $\geq 0$. An alternative way to define a positive definite matrix is as follows. A symmetric matrix $S \in \mathbb{R}^n \times \mathbb{R}^n$ is positive definite if for for any non-zero vector $x \in \mathbb{R}^n$, $x^T S x > 0$. If $x^T S x \geq 0$, then $S$ is positive semi-definite.

**Lemma 4.4.5** *Show that for a symmetric matrix $S \in \mathbb{R}^n \times \mathbb{R}^n$ if for for any non-zero vector $x \in \mathbb{R}^n$, $x^T S x > 0$ implies that all its eigenvalues are $> 0$.*

**Proof.** Let $\lambda_i$ be an eigenvalue of $S$ and its corresponding unit eigenvector is $q_i$. Note that $q_i^T q_i = 1$. Since $S$ is symmetric, we know that $\lambda_i$ is real. Now we have, $\lambda_i = \lambda_i q_i^T q_i = q_i^T \lambda_i q_i = q_i^T S v_i$. But $q_i^T S q_i > 0$, hence $\lambda_i > 0$. ∎

## 4.5 *Singular Value Decomposition*

Let $A$ be a $m \times n$ matrix of rank $r$. By rank, we mean the number of linearly independent rows (or columns) of $A$. Note that $A$ may not be a square matrix. The singular value decomposition (SVD) enables us to diagonalize any type of matrices. In previous section we have seen the diagonalization of a square matrix $A = X^{-1}AX$, where $X$ is the column of $n$-independent eigenvectors. There we needed $A$ to be square, (usually) all its eigenvalues to be distinct so that the eigenvectors are independent. Here we will not make any such assumptions on $A$ but we will be able to diagonalize it. This diagonalization is slightly more complex, and will use four orthonormal bases associated with column and row subspaces of $A$.[2] These are:

1. Let $u_1, \ldots, u_r$ be an orthonormal bases of columns of $A$.
2. Let $u_{r+1}, \ldots, u_m$ be an orthonormal basis for the nullspace of $A^T$.
3. Let $v_1, \ldots, v_r$ be an orthonormal bases of rows of $A$.
4. Let $v_{r+1}, \ldots, v_n$ be an orthonormal basis for the nullspace of $A$.

Moreover, these vectors will satisfy the following $Au_1 = \sigma_1 v_1, \ldots, Au_r = \sigma_1 v_r$, where $\sigma_i > 0$ for $i = 1, \ldots, r$. This can equivalently be expressed in matrix notation as

$$A \begin{bmatrix} v_1 & v_2 & \ldots & v_r \end{bmatrix} = \begin{bmatrix} u_1 & u_2 & \ldots & u_r \end{bmatrix} \begin{bmatrix} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & . & & \\ & & & . & \\ & & & & \sigma_r \end{bmatrix}$$

We further expand this to include the whole set of orthonormal vectors $U = [u_1, \ldots, u_r, u_{r+1}, \ldots, u_m]$ and $V = [v_1, \ldots, v_r, v_{r+1}, \ldots, v_n]$ as follows

$$A \begin{bmatrix} v_1 \ldots v_r v_{r+1} \ldots v_n \end{bmatrix} = \begin{bmatrix} u_1 \ldots u_r u_{r+1} \ldots u_m \end{bmatrix} \begin{bmatrix} \sigma_1 & & & & & \\ & . & & & & \\ & & \sigma_r & & & \\ & & & 0 & & \\ & & & & 0 & \\ & & & & & 0 \end{bmatrix}$$

[2] To get some insight in these subspaces, consider $Ax = b$, where $A$ is an $m \times n$ matrix and assume all elements in $A, b, x$ are real. This system has solution when $b$ is equal to some linear combination of the columns of $A$ (specified by $x$). In that case we say that $b$ is in the column space of $A$. The column space of $A$ consists of all linear combinations of the columns of $A$ and $Ax = b$ is solvable if and only if $b$ is in the column space of $A$. Note that $A$ has $n$ columns, but each column only consists of $m$ entries. Each column is in $\mathbb{R}^m$ and the column space is a subspace of $\mathbb{R}^m$. The rank of a matrix is the number of independent columns (or rows) of $A$. Equivalently, when we do a row reduction (e.g. in Gaussian elimination) the rank is the number of non-zero rows in $A$. Furthermore, we define the null space of $A$ to consist of all solutions $x$ to the equation $Ax = 0$. Note that these are the vectors in $\mathbb{R}^n$.

and more compactly we can write this as $AV = U\Sigma$ or equivalently $A = U\Sigma V^{-1} = U\Sigma V^T$, where $\Sigma$ is the diagonal matrix consisting $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r, 0, 0, \ldots$ and $V^T = V^{-1}$ (as $V$ is a matrix of orthonormal vectors). Let us try to see what are these $\sigma_i$'s are.

First we assume that $m \geq n$ and $rank(A) = n$, and later on we will consider the case that $rank(A) < n$. Consider the matrix product $A^TA$. This is an $n \times n$ matrix. Moreover it is symmetric as $(A^TA)^T = A^T(A^T)^T = A^TA$.

**Lemma 4.5.1** *The matrix $A^TA$ is positive definite.*

**Proof.** Take any non-zero vector $x \in \mathbb{R}^n$. Consider $x^TA^TAx = (Ax)^T(Ax) = ||Ax||^2 \geq 0$. We have assumed that $rank(A) = n$. This implies that the rank of the null-space of $A$ is $n - rank(A) = 0$. Hence, the only vector $x$ for which $Ax = 0$ is $x = 0$. This implies that $x^TA^TAx = ||Ax||^2 > 0$ and hence $A^TA$ is positive definite. ∎

From the positive definiteness of $A^TA$, we know that its eigenvalues are positive and the corresponding eigenvectors are orthonormal. Let its eigenvalues be $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n > 0$ and let the corresponding orthonormal eigenvectors be $v_1, v_2, \ldots, v_n$, respectively. We know that $A^TAv_i = \lambda_i v_i$, or equivalently $v_i^TA^TAv_i = \lambda_i$. [3] Define $\sigma_i = ||Av_i||$. Then $\sigma_i^2 = ||Av_i||^2 = v_i^TA^TAv_i = \lambda_i$. Thus, $\sigma_i = ||Av_i|| = \sqrt{\lambda_i}$. Note that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n > 0$. We define the vectors $u_1, \ldots, u_n \in \mathbb{R}^m$ as follows. Each $u_i = Av_i/\sigma_i$. Note that $u_i$'s are well defined as $\sigma_i \neq 0$.

[3] $v_i^Tv_i = 1$ as $v_i$ is orthonormal.

**Lemma 4.5.2** *The set of vectors $u_i = Av_i/\sigma_i$, for $i = 1, \ldots, n$, are orthonormal.*

**Proof.** First, let us consider

$$||u_i|| = ||Av_i||/\sigma_i = \sigma_i/\sigma_i = 1.$$

Now we show that the dot product of any two vectors $u_i$ and $u_j$ for $1 \leq i \neq j \leq n$ is zero.

$$u_i^Tu_j = (Av_i/\sigma_i)^T(Av_j/\sigma_j) = \frac{1}{\sigma_i\sigma_j}v_i^TA^TAv_j = \frac{1}{\sigma_i\sigma_j}v_i^T\lambda_j v_j = \frac{\lambda_j}{\sigma_i\sigma_j}v_i^Tv_j = 0.$$

[4] ∎

[4] As $A^TAv_j = \lambda_j v_j$ and $v_i$ and $v_j$ are orthonormal.

Now we have that $v_1, \ldots, v_n$ are orthonormal and $u_1, \ldots, u_n$ are orthonormal and for $i = 1, \ldots, n$, $Av_i = \sigma_i u_i$. In the matrix notation we can write this as $AV' = U'\Sigma'$, where $A$ is the given $rank(A) = n$ $m \times n$ matrix, $V'$ is $n \times n$ matrix consisting of orthonormal vectors $v_1, \ldots, v_n$, $U$ is $m \times n$ matrix consisting of orthonormal vectors $u_1, \ldots, u_n$, and $\Sigma'$ is $n \times n$ diagonal matrix where each $(i,i) - th$ entry

is $\sigma_i$ for $i = 1, \ldots, n$ and $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_n > 0$. Since $V'$ is square and orthonormal, $V'^{-1} = V'^T$. Multiply by $V'^T$ on both the sides on the right of $AV' = U'\Sigma'$ and we obtain $A = U'\Sigma'V'^T$.

The matrix $U'$ is not a square matrix. It consists of orthonormal vectors $u_1, \ldots, u_n$ but it has more rows than columns (as $m \geq n$). Hence the null space of $U'$ is non-empty. Let $u_{r+1}, \ldots, u_m$ be orthonormal vectors to $u_1, \ldots, u_r$ (i.e., in the null space of $A^T$). Therefore, together with $u_1, \ldots, u_n$ we form the matrix $U$ of dimension $m \times m$ consisting of $m$ orthonormal vectors $u_1, \ldots, u_m$. Thus $U^TU = I$ and $U^{-1} = U^T$.

Lastly, we transform $\Sigma'$ to $\Sigma$. It is $m \times n$ matrix, where each of its entry is 0 except the $(i,i)$-th entry equals $\sigma_i$ for $i = 1, \ldots, n$.

Observe that under the assumption that $rank(A) = n$ and $m \geq n$, now we have that $A = U\Sigma V'^T$, where $U^TU = I$ and $V^TV = I$. Note that we can also express $A = \Sigma_{i=1}^n \sigma_i u_i v_i^T$. This will be the rank one decomposition of $A$. Since $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_n$, we can see that the sum of the (initial) values of $\sigma_i u_i v_i^T$ corresponding to the large values of $\sigma_i$ essentially approximate $A$.

**Example 4.5.3** *Let* $A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 1 \end{bmatrix}$. *$A$ is $3 \times 2$ matrix, where $m = 3$ and $n = 2$. Rank of $A$ is $n = 2$.* $A^TA = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 2 \\ 2 & 2 \end{bmatrix}$. *Eigenvalues of $A^TA$ are given by det* $\begin{bmatrix} 5-\lambda & 2 \\ 2 & 2-\lambda \end{bmatrix} = 0$. *This results in* $\lambda^2 - 7\lambda + 6 = (\lambda - 6)(\lambda - 1) = 0$, *or* $\lambda_1 = 6$ *and* $\lambda_2 = 1$. *Note that* $\lambda_1 \geq \lambda_2 > 0$. *The corresponding eigenvectors are*

$A^TAv_1 = \lambda_1 v_1$ *is* $v_1 = \begin{bmatrix} 2/\sqrt{5} \\ 1/\sqrt{5} \end{bmatrix}$, *and*

$A^TAv_2 = \lambda_2 v_2$ *is* $v_2 = \begin{bmatrix} -1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix}$.

*Note that $v_1$ and $v_2$ are orthonormal vectors as $v_1^Tv_2 = 0$ and $||v_1|| = 1$ and $||v_2|| = 1$. Next we compute the vectors $u_1$ and $u_2$. First $\sigma_1 = \sqrt{\lambda_1} = \sqrt{6}$ and $\sigma_2 = \sqrt{\lambda_2} = \sqrt{1} = 1$. Then,*

$u_1 = \frac{1}{\sigma_1} Av_1 = \frac{1}{\sqrt{6}} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 2/\sqrt{5} \\ 1/\sqrt{5} \end{bmatrix} = \begin{bmatrix} 2/\sqrt{30} \\ 1/\sqrt{30} \\ 5/\sqrt{30} \end{bmatrix}$.

*Similarly,*

$u_2 = \frac{1}{\sigma_2} Av_2 = \frac{1}{\sqrt{1}} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} -1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix} = \begin{bmatrix} -1/\sqrt{5} \\ 2/\sqrt{5} \\ 0 \end{bmatrix}$.

*Note that $u_1$ and $u_2$ are orthonormal vectors.*

*Set* $U' = \begin{bmatrix} 2/\sqrt{30} & -1/\sqrt{5} \\ 1/\sqrt{30} & 2/\sqrt{5} \\ 5/\sqrt{30} & 0 \end{bmatrix}$, $\Sigma' = \begin{bmatrix} \sqrt{6} & 0 \\ 0 & 1 \end{bmatrix}$, *and* $V' = \begin{bmatrix} 2/\sqrt{5} & -1/\sqrt{5} \\ 1/\sqrt{5} & 2/\sqrt{5} \end{bmatrix}$.

*Observe that $A = U'\Sigma'V'^T$, i.e.*

$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 2/\sqrt{30} & -1/\sqrt{5} \\ 1/\sqrt{30} & 2/\sqrt{5} \\ 5/\sqrt{30} & 0 \end{bmatrix} \begin{bmatrix} \sqrt{6} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 2/\sqrt{5} & 1/\sqrt{5} \\ -1/\sqrt{5} & 2/\sqrt{5} \end{bmatrix}$.

*Next, we extend $U'$ to $U$ by taking a unit vector in the null space of $U'$ that is orthonormal to $u_1$ and $u_2$. Let $u_3 = (a, b, c)$. Then we set $u_1^Tu_3 = 0$ and $u_2^Tu_3 = 0$ and $||u_3|| = \sqrt{a^2 + b^2 + c^2} = 1$ and obtain*

*that* $u_3 = (2/\sqrt{6}, 1/\sqrt{6}, -1/\sqrt{6})$. *We derive* $\Sigma$ *from* $\Sigma'$ *by setting* $\Sigma = \begin{bmatrix} \sqrt{6} & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$.

*Now we have that* $A = U\Sigma V'^T$, *i.e.*

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 2/\sqrt{30} & -1/\sqrt{5} & 2/\sqrt{6} \\ 1/\sqrt{30} & 2/\sqrt{5} & 1/\sqrt{6} \\ 5/\sqrt{30} & 0 & -1/\sqrt{6} \end{bmatrix} \begin{bmatrix} \sqrt{6} & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 2/\sqrt{5} & 1/\sqrt{5} \\ -1/\sqrt{5} & 2/\sqrt{5} \end{bmatrix}.$$

*Lastly, in terms of rank one decomposition of* $A$, *observe that*

$$A = \sqrt{6} \begin{bmatrix} 2/\sqrt{30} \\ 1/\sqrt{30} \\ 5/\sqrt{30} \end{bmatrix} \begin{bmatrix} 2/\sqrt{5} \\ 1/\sqrt{5} \end{bmatrix}^T + \begin{bmatrix} -1/\sqrt{5} \\ 2/\sqrt{5} \\ 0 \end{bmatrix} \begin{bmatrix} -1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix}^T.$$

Now we handle the case when $m \geq n$ and $rank(A) < n$. Let $rank(A) = r < n$. Thus the rank of the null space of $A$ is $n - r$. We know that $n - r$ eigenvalues of $A$ are 0. We claim the following.

**Lemma 4.5.4** *The* $n - r$ *eigenvalues of* $A^T A$ *equals* 0.

**Proof.**

Consider a basis of the null space of $A$. Let $x_1, \ldots, x_{n-r}$ be a basis of the null space of $A$. This implies that $Ax_j = 0$ for $j = 1, \ldots, n - r$. Now, $A^T Ax_j = 0 = 0x_j$. Thus, 0 is an eigenvalue of $A^T A$ corresponding to each $x_i$'s. Thus $n - r$ eigenvalues of $A^T A$ equals 0. ∎

As in the case of $rank(A) = n$, we first compute the eigenvalues and eigenvectors of $A^T A$. Let $\lambda_1 \geq \lambda_2 \geq \lambda_n \geq 0$ be the eigenvalues of $A^T A$. Since $n - r$ of them are 0, we know that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r > 0$ and $\lambda_{r+1} = \cdots = \lambda_n = 0$. Let $v_1, \ldots, v_n$ be the corresponding orthonormal vectors, where $v_{r+1}, \ldots, v_n$ are in the null space of $A^T A$, i.e. $A^T Av = 0$ for each $v \in \{v_{r+1}, \ldots, v_n\}$. The vectors $v_1, \ldots, v_n$ defines the orthonormal matrix $V$. Next we define $\sigma_i = ||Av_i|| = \sqrt{\lambda_i}$. Note that $\sigma_1 \geq \ldots \sigma_r > 0$ and $\sigma_{r+1} = \cdots = \sigma_n = 0$. We have to take some care in defining the vectors $u_1, \ldots, u_n$ as some of the $\sigma_i$'s are 0. For $i = 1, \ldots, r$, we define $u_i = \frac{1}{\sigma_i} Av_i$. As before $u_1, \ldots, u_r$ are orthonormal and $Av_i = \sigma_i u_i$. For $i = r + 1, \ldots, n$ we construct $u_i$'s that are orthonormal to all the vectors $u_1, \ldots, u_{i-1}$. Note that each of these $u_i$'s satisfy $\sigma_i u_i = Av_i = 0$ as $\sigma_i = 0$, for $i = r + 1, \ldots, n$. This will result in a matrix of orthonormal vectors $u_1, \ldots, u_n$ of dimension $m \times n$. We will further extend it to form a matrix $U$ of orthonormal vectors $u_1, \ldots, u_m$. We also construct the matrix $\Sigma$ of dimension $m \times n$, where all its entries are 0 except the $(i, i)$-th entry equals $\sigma_i$, for $i = 1, \ldots, r$. Given that $Av_i = \sigma_i u_i$ for $i = 1, \ldots, n$, we have that $A = U\Sigma V^T$. Furthermore, the rank one decomposition of $A = \Sigma_{i=1}^r \sigma_i u_i v_i^T$.

If $m < n$, then we can work with $A^T$ in place of $A$, and the details are similar and hence we skip them. Given that $A = U\Sigma V^T$, consider $A^T A = (U\Sigma V^T)^T (U\Sigma V^T) = (V\Sigma U^T)(U\Sigma V^T) = VU^T \Sigma^2 UV^T = V\Sigma^2 UU^T V^T = V\Sigma^2 V^T$. Note that $A^T A$ is a symmetric matrix, and since it is expressed in the diagonalized form $A^T A = V\Sigma^2 V^T$, $\sigma_i^2$'s are

its eigenvalues and $V$ is its eigenvectors matrix. Similarly, consider $AA^T$ and we obtain that $AA^T = (U\Sigma V^T)(U\Sigma V^T)^T = U\Sigma V^T V\Sigma U^T = U\Sigma^2 U^T$. Thus $U$ is the eigenvector matrix for the symmetric matrix $AA^T$ with the same eigenvalues as $A^TA$. Overall, the singular value decomposition of a $m \times n$ matrix $A$ can be captured in the following theorem.

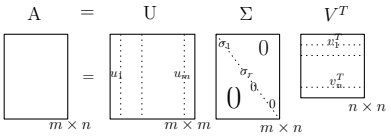


Figure 4.1: Illustration of SVD of a $m \times n$ matrix $A$ expressed as $U\Sigma V^T$.

**Theorem 4.5.5** *Let $A$ be a $m \times n$ matrix of real numbers of rank $r \leq \min(m, n)$. Then $A = U\Sigma V^T$, where*

*$U$ is a orthonormal $m \times m$ matrix and $U^TU = I$,*

*$V$ is a orthonormal $n \times n$ matrix and $V^TV = I$, and*

*$\Sigma$ is an $m \times n$ matrix where each of its entries is 0, except the $(i, i)$-th entry is $\sigma_i$ for $i = 1, \ldots, r$. Note that $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_r > 0$ and $\sigma_i = \sqrt{\lambda_i}$ where $\lambda_i$ are the eigenvalues of $A^TA$. The set of orthonormal vectors $v_1, \ldots, v_r$ and $u_1, \ldots, u_r$ are eigenvectors of $A^TA$ and $AA^T$, respectively. Furthermore, the orthonormal basis $v_{r+1}, \ldots, v_n$ for the null space of $A$ together with $v_1, \ldots, v_r$ forms the set $V$. Analogously, the orthonormal basis $u_{r+1}, \ldots, u_m$ for the null space of $A^T$ together with $u_1, \ldots, u_r$ forms the set $U$. Alternatively, we can express $A = \Sigma_{i=1}^{r} \sigma_i u_i v_i^T$ in terms of its rank one components.*

## 4.6 Markov Matrices

Let $X_0, X_1, \ldots$ be a sequence of random variables that are evolving over time. At time 0, we have the r.v. $X_0$, followed by r.v. $X_1$ at time 1 and so on. Assume that each $X_i$ takes value from the set $\{1, \ldots, n\}$ that represents the set of states. This sequence is a *Markov chain* if the probability that $X_{m+1}$ equals a particular state $\alpha_{m+1} \in \{1, \ldots, n\}$ only depends on what is the state of $X_m$ and is completely independent of the states of $X_0, \ldots, X_{m-1}$, i.e $P[X_{m+1} = \alpha_{m+1} | X_m = \alpha_m, X_{m-1} = \alpha_{m-1}, \ldots, X_0 = \alpha_0] = P[X_{m+1} = \alpha_{m+1} | X_m = \alpha_m]$, where $\alpha_0, \ldots, \alpha_{m+1}, \cdots \in \{1, \ldots, n\}$. This is called the memoryless property of Markov chains as the most recent r.v. $X_m$ determines the value of $X_{m+1}$, and it doesn't matter how $X_m$ acquired its state value. To understand the transition of Markov chain from one state to another, we define *state transition probabilities*, i.e. what is the probability to go from state $i$ to state $j$ for all $1 \leq i, j, \leq n$. See Figure 4.3 for an example of a Markov chain with three states. This is usually represented in a transition matrix defined as follows. A square matrix $A$ of dimension $n \times n$ is a *Markov matrix* if all its entries are non-negative and the entries within each column sums to 1. This matrix represents a system with $n$ states where the $(ij)$-th entry is the transition probability from state $j$ to state $i$. Suppose that $A[i, j] = 0.3$. This means that if $X_m = j$, then there is a 30% chance that $X_{m+1} = i$. If $A[k, j] = 0$

then a direct transition from the state $j$ to the state $i$ is impossible. If $A[j,j] = 0.4$ then there is a 40% chance that the next state transition will be to the same state. See Figure 4.4 to see the Markov matrix corresponding to the Markov chain of Figure 4.3.

It is fairly common to represent a Markov chain as a directed graph $G = (V, E)$. Its node set $V$ consists of all the states $\{1, \ldots, n\}$. There is a directed edge from a node $j$ to node $i$, if there is a non-zero probability to make a transition from state $j$ to state $i$ in one step, i.e. $A[i,j] > 0$. We typically start with an initial state in Markov chain given by the value of r.v. $X_0$ (representing a node in $G$), and in each successive step we make a state transition from the current state given by $X_m = i$ to the next state given by $X_{m+1} = j$ (i.e., follow a directed edge from node $i$ to node $j$) that respects the the probability of the state transition from the node $j$ to $i$ in $G$. One of the questions that is commonly asked is what is the probability of reaching the state $j$ after taking $n$ steps starting from the state $i$? In general, given an initial probability vector representing the probabilities of starting in various states, we are interested to know what is the steady state, i.e. after traversing the chain for a large number of steps, what is the probability of landing in various states. This concept will become clear in the next section after we establish some more (graph-theoretic) properties of Markov chains.

We say that a state $i$ is *recurrent* if starting from the state $i$, with probability 1, we can return to the state $i$ after making finitely many transitions. Otherwise, it is transient, i.e. there is a non-zero probability of not returning to the state $i$. See Figure 4.2 for an example.

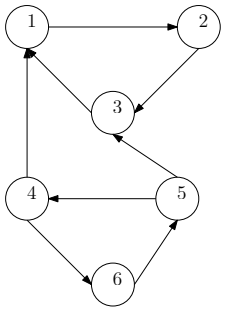


Figure 4.2: Recurrent States={1,2,3}. Transient States={4,5,6}

We say a Markov chain is *irreducible* if it is possible to go between any pair of states in a finite number of steps. Otherwise it is called reducible. Note that the chain in Figure 4.2 is reducible as state 4 cannot be reached from state 1. In terms of graph theory, if $G$ is strongly connected then it is irreducible. Exercise 4.10 asks you to show that if a Markov chain is irreducible, then all its states are recurrent. Observe that if all states of a Markov chain are recurrent, then it doesn't mean that it is irreducible. For example, consider a Markov chain with two states, and the only transition that is allowed is to stay in the same state with probability 1. Clearly both the states are recurrent, but there is no way to reach one state from the other. *Period* of a state $i$ is the greatest common divisor (GCD) of all possible number of steps it takes the chain to return to the state $i$ starting from $i$. Formally, the period of the state $i$ equals $GCD\{m > 0 : P[X_m = i | X_0 = i] > 0\}$. If there is no way to return to $i$ starting from $i$, then its period is undefined. We say a Markov chain is *aperiodic* if the periods of each of its states is 1.

### 4.6.1 *Eigenvalues of a Markov matrix*

Recall that a Markov matrix (or a stochastic matrix or a transition matrix or a probability matrix) $A$ is a square matrix that captures the probability of transition from one state to another in a Markov process. Each $A[i,j]$, $1 \leq i \leq n$ and $1 \leq j \leq n$, is non-negative and its value is equal to the probability of transition from the state $j$ to state $i$. Observe that the sum of the values within any column is 1 (as that equals the probability of leaving from a state to any of the possible states). Assume that $A$ is a $n \times n$ Markov matrix. Its eigenvalues are the roots of the equation given by the determinant of the matrix $A - \lambda I$, i.e. $det(A - \lambda I) = 0$, where $I$ denotes an identity matrix. We are interested in establishing some properties of the eigenvalues of $A$. First an example.

The Markov matrix $A$ corresponding to the Markov chain from Figure 4.3 is given in Figure 4.4.

Let us compute the eigenvalues of $A$ by determining the determinant of the following matrix and equating it to zero:

$$A - \lambda I = \begin{bmatrix} 0-\lambda & 1/3 & 1/3 \\ 1/2 & 0-\lambda & 2/3 \\ 1/2 & 2/3 & 0-\lambda \end{bmatrix}$$

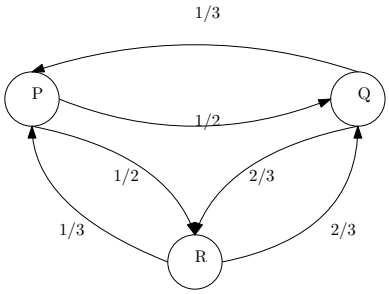



Figure 4.3: Markov chain with three states and transition probabilities.

$$A = \begin{array}{c} \\ P \\ Q \\ R \end{array} \begin{array}{c} \begin{array}{ccc} P & Q & R \end{array} \\ \begin{bmatrix} 0 & 1/3 & 1/3 \\ 1/2 & 0 & 2/3 \\ 1/2 & 2/3 & 0 \end{bmatrix} \end{array}$$

Figure 4.4: Markov matrix $A$ corresponding to Figure 4.3.

Now $\det(A - \lambda I) = 0$ gives us the equation $9\lambda^3 - 7\lambda - 2 = 0$. The roots of this equation are $\lambda_1 = 1, \lambda_2 = -2/3$, and $\lambda_3 = -1/3$ and the corresponding eigenvectors are $v_1 = (2/3, 1, 1)$, $v_2 = (0, -1, 1)$, and $v_3 = (-2, 1, 1)$, respectively. We observe that the largest (principal) eigenvalue is 1 and the corresponding (principal) eigenvector is $(2/3, 1, 1)$. Note that $Av_i = \lambda_i v_i$, for $i = 1, \ldots, 3$. If required, we can convert any eigenvector to a unit vector $(\frac{v_i}{||v_i||})$ or take its scalar multiple, without altering its corresponding eigenvalue. Moreover, it is straightforward to see that $A$ will have an eigenvalue of 1, as in $A^T$ all the elements in each row add to 1 and the determinant (and the eigenvalues) of a matrix is same as that of its transpose (see Exercise 4.1). The next theorem states that the largest eigenvalue of a Markov matrix is 1.

**Theorem 4.6.1** *The largest eigenvalue of any Markov matrix is* 1.

**Proof.** Let $A$ be a Markov matrix of dimension $n$ and let $B = A^T$. Let $\mathbf{1} = (1, 1, \ldots, 1)$ be a vector of dimension $n$. All the elements within each row of $B$ sums to 1, thus $\sum_{j=1}^{n}(b_{ij} \cdot 1) = 1$ and $B\mathbf{1} = 1 \cdot \mathbf{1}$. Hence 1 is an eigenvalue of $B$ (and therefore for $A$ as $A = B^T$). Now we show that the largest eigenvalue of $B$ is 1. We prove this by contradiction. Suppose there exists an eigenvalue $\lambda > 1$ and a

non-zero vector $\mathbf{x}$ such that $B\mathbf{x} = \lambda\mathbf{x}$. Let $x_i$ be among the largest element(s) of $\mathbf{x}$. We can assume that $x_i > 0$, otherwise we multiply by a scalar and that also satisfies this equation. Entries in $B$ are non-negative (they are probabilities), and the sum of all the elements in any row of $B$ is 1. We can interpret each entry in $\lambda\mathbf{x}$ as a convex combination of the elements of $\mathbf{x}$. For example, the $i$-th row yields $b_{i1}x_1 + b_{i2}x_2 + \cdots + b_{in}x_n = \lambda x_i$. But $b_{i1}x_1 + b_{i2}x_2 + \cdots + b_{in}x_n \leq b_{i1}x_i + b_{i2}x_i + \cdots + b_{in}x_i = (b_{i1} + b_{i2} + \cdots + b_{in})x_i = x_i$. Because of the convex combination, no entry in $\lambda\mathbf{x}$ can be larger than $x_i$. But since $\lambda > 1$, and in particular $\lambda x_i > x_i$, the $i$th row sum $\sum_{j=1}^{n}(b_{ij}x_j) = \lambda x_i > x_i$ leads to a contradiction. Therefore, the largest eigenvalue of $B$, and hence of $A$, is 1. ∎

Consider the Markov matrix $A$ corresponding to Figure 4.3.

$$A = \begin{bmatrix} 0 & 1/3 & 1/3 \\ 1/2 & 0 & 2/3 \\ 1/2 & 2/3 & 0 \end{bmatrix}$$

Note that all the entries in $A^2$ are $> 0$ and all the entries within a column still adds to 1.

$$A^2 = \begin{bmatrix} 1/3 & 2/9 & 2/9 \\ 1/3 & 11/17 & 1/6 \\ 1/3 & 1/6 & 11/17 \end{bmatrix}$$

In fact, we can observe that if the entries within each column of $A$ adds to 1, then entries within each column of $A^k$, for any integer $k > 0$, will add to 1.

Next consider multiplying $A$ by a vector $u_0$, where each coordinate of $u_0 \geq 0$ and the sum of the coordinates of $u_0$ is 1. Think of $u_0$ as a initial probability distribution of a random surfer, i.e. the probability of the surfer in any given state. Consider $u_1 = Au_0$. Observe that each of the coordinates of $u_1 \geq 0$ and they also add to 1. The vector $u_1$ is the probability of various states in which the surfer is after making a state transition conditioned on that the starting probability distribution is $u_0$. For example, for $u_0 = (1/3, 1/3, 1/3)$,

$$u_1 = Au_0 = \begin{bmatrix} 0 & 1/3 & 1/3 \\ 1/2 & 0 & 2/3 \\ 1/2 & 2/3 & 0 \end{bmatrix} \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} = \begin{bmatrix} 4/18 \\ 7/18 \\ 7/18 \end{bmatrix}$$

Similarly, define $u_2 = Au_1 = A^2u_0$, and for our example it is

$$u_2 = Au_1 = \begin{bmatrix} 0 & 1/3 & 1/3 \\ 1/2 & 0 & 2/3 \\ 1/2 & 2/3 & 0 \end{bmatrix} \begin{bmatrix} 4/18 \\ 7/18 \\ 7/18 \end{bmatrix} = \begin{bmatrix} 7/27 \\ 10/27 \\ 10/27 \end{bmatrix}$$

Likewise, we compute $u_3 = Au_2 = [20/81, 61/162, 61/162]$, $u_4 = Au_3 = [61/243, 91/243, 91/243]$, $u_5 = Au_4 = [182/729, 547/1458, 547/1458]$, $\ldots$, $u_\infty = [0.25, 0.375, 0.375] = [2/8, 3/8, 3/8]$. Note that by taking the repeated powers, $\lim_{k\to\infty} A^k u_0 = [2/8, 3/8, 3/8]$ and that corresponds to the steady state. In fact the steady state corresponds to the principal eigenvector. We formalize this notion next.

We can express the vector $u_0$ as a linear combination of eigenvectors. For our example, let $u_0 = c_1v_1 + c_2v_2 + c_3v_3$, where $c_1, c_2, c_3$ are constants. In particular,

$$\begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} = c_1 \begin{bmatrix} 2/3 \\ 1 \\ 1 \end{bmatrix} + c_2 \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} + c_3 \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix}$$

Now $u_1$ can be expressed as $u_1 = Au_0 = c_1Av_1 + c_2Av_2 + c_3Av_3 = c_1\lambda_1v_1 + c_2\lambda_2v_2 + c_3\lambda_3v_3$, as $Av_i = \lambda_iv_i$, i.e.,

$$u_1 = A \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} = c_1\lambda_1 \begin{bmatrix} 2/3 \\ 1 \\ 1 \end{bmatrix} + c_2\lambda_2 \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} + c_3\lambda_3 \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix}.$$

Thus, in general, for integer $k > 0$, $u_k = A^k u_0 = c_1\lambda_1^k v_1 + c_2\lambda_2^k v_2 + c_3\lambda_3^k v_3$, i.e.

$$u_k = A^k \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} = c_1\lambda_1^k \begin{bmatrix} 2/3 \\ 1 \\ 1 \end{bmatrix} + c_2\lambda_2^k \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} + c_3\lambda_3^k \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix}.$$

and that equals

$$u_k = c_1 1^k \begin{bmatrix} 2/3 \\ 1 \\ 1 \end{bmatrix} + c_2(-\frac{2}{3})^k \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} + c_3(-\frac{1}{3})^k \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix}.$$

For large values of $k$, $(\frac{2}{3})^k \to 0$ and $(\frac{1}{3})^k \to 0$. The above expression reduces to

$$u_k \approx c_1 \begin{bmatrix} 2/3 \\ 1 \\ 1 \end{bmatrix} = \frac{3}{8} \begin{bmatrix} 2/3 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2/8 \\ 3/8 \\ 3/8 \end{bmatrix}$$

Note that the value of $c_1$ is derived by solving the equation for $u_0 = c_1v_1 + c_2v_2 + c_3v_3$ for $u_0 = [1/3, 1/3, 1/3]$.

Let us see what happens if the initial vector $u_0 = [1/4, 1/4, 1/2]$. Then $u_1 = Au_0 = [1/4, 11/24, 7/24]$, $u_2 = Au_1 = [1/4, 23/72, 31/72]$, $u_3 = Au_2 = [1/4, 89/216, 73/216]$, $\ldots$, $u_\infty = [2/8, 3/8, 3/8]$. Hence a different initial value $u_0 = [1/4, 1/4, 1/2]$ still leads to the same steady state corresponding to the principal eigenvector. The reasoning is same as before. Express $u_0 = d_1v_1 + d_2v_2 + d_3v_3$ for constants $d_1, d_2$ and $d_3$. Now, for any $k > 0$, $u_k = A^k u_0 = d_1\lambda_1^k v_1 + d_2\lambda_2^k v_2 + d_3\lambda_3^k v_3 = d_1v_1 + d_2(-\frac{2}{3})^k v_2 + d_3(-\frac{1}{3})^k v_3$. Now take $\lim_{k\to\infty} u_k = d_1v_1$. We solve for $d_1$ using the initial condition

$$\begin{bmatrix} 1/4 \\ 1/4 \\ 1/2 \end{bmatrix} = d_1 \begin{bmatrix} 2/3 \\ 1 \\ 1 \end{bmatrix} + d_2 \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} + d_3 \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix}$$

and obtain that $d_1 = 3/8$. Thus $u_\infty = [2/8, 3/8, 3/8]$.

What can we say about the matrix product $A^k$ for large values of $k$? Observe that by the same reasoning each column of $A^k$ will be a steady state, i.e. the vector $u_\infty = [2/8, 3/8, 3/8]$.

Consider the following example that illustrates the number of fans of the Senators and the Leafs NHL Teams at the end of the season. Assume that there are 3,000,000 hockey fans all over Canada for these two teams and at the end of the season, depending on the performance of the two teams, certain fraction of the fans switch loyalties. Assume that the following transition matrix captures the change in loyalties:

$$A = \begin{array}{c} \begin{matrix} \textit{Sens} & \textit{Leafs} \end{matrix} \\ \begin{bmatrix} 0.9 & 0.3 \\ 0.1 & 0.7 \end{bmatrix} \end{array} \begin{array}{l} \textit{Sens} \\ \textit{Leafs} \end{array}$$

Note that only 10% of Sens fans switch loyalties as opposed to 30% of the Leafs fans. We assume that this trend stays forever. Let us assume that there are $50,000$ Sens fans and $2,500,000$ leafs fans at the start (say 20 years ago) and we want to know what will be the steady state of the fans distribution. Let us first find the eigenvalues and eigenvectors of $A$. It is easy to see that $\lambda_1 = 1$ and $\lambda_2 = 0.6$ and the corresponding eigenvectors are $v_1 = (3, 1)$ and $v_2 = (1, -1)$, respectively. We know that $u_0 = (50000, 2500000)$ and we want to find $u_k$ for large values of $k$, where $u_k = Au_{k-1}$. From the theory developed above we know that for constants $c_1$ and $c_2$, $u_k = c_1(\lambda_1)^k v_1 + c_2(\lambda_2)^k v_2 = c_1 1^k v_1 + c_2(0.6)^k v_2$. The initial condition $u_0 = [500000, 2500000] = c_1[3, 1] + c_2[1, -1]$ results in $c_1 = 750,000$ and $c_2 = -1,750,000$. As $k \to \infty$, $0.6^k \to 0$. Thus, $\lim_{k\to\infty} u_k = 750,000[3, 1]$ or there are $2,250,000$ Sens fans and $750,000$ Leafs fans in the steady state.

The above examples leads to the following abstraction. Assume that all the entries of a Markov matrix $A$, or of some finite power of $A$, i.e. $A^k$ for some fixed integer $k > 0$, are strictly $> 0$. These conditions imply that $A$ corresponds to an irreducible aperiodic Markov chain $M$. (Recall that in an irreducible chain $M$, for any pair of states $i$ and $j$, it is always possible to go from state $i$ to state $j$ in finite number of steps with positive probability. Informally, period of a state $i$ is the greatest common divisor of all possible number of steps it takes the chain to return to the state $i$ starting from $i$. $M$ is aperiodic if the GCD is 1 for period of each of the states in $M$.) As a consequence of the Perron-Frobenius theorem from linear algebra it will turn out that almost always (a) the largest eigenvalue 1 of $A$ will be unique, (b) all other eigenvalues of $A$ have magnitude strictly smaller than 1, (c) all the coordinates of the eigenvector $v_1$ corresponding to the eigenvalue 1 are $> 0$, and $(d)$ the steady state corresponds to the eigenvector $v_1$.

### 4.6.2 *PageRank*

Now a days we cannot imagine a life without the www. Google, Yahoo, Safari, ... are among the various search engines that search the internet to answer our web queries on a wide range of topics. In this section, we sketch how the ranking of the web-pages is done by the page rank algorithm from the founders of Google [5] (see also [6]). Ranking assigns a real number to each web-page. The higher the number, the more important the page is. Since the web is extremely large, the ranking cannot be done manually. First we will provide a very simple model of the web and see how the Markov matrices can help us to rank the web pages.

Consider the web as a directed graph $G = (V, E)$ defined as follows. Each web-page is a vertex of $G$. If a web-page $u$ points (links) to the web-page $v$, there is a directed edge from $u$ to $v$ in $E$. The weight of an edge $uv$ is $\frac{1}{\text{out-degree}(u)}$. Assume $V = \{v_1, \ldots, v_n\}$. Define an $n \times n$ adjacency matrix $M$ as follows.

$$\text{For } 1 \leq i, j \leq n,\ M(i,j) = \begin{cases} \frac{1}{\text{out-degree}(v_j)}, & \text{if} v_j v_i \in E \\ 0 & \textit{otherwise}. \end{cases}$$

For example consider the following simple web-graph (Figure 4.5) and its associated matrix (Figure 4.6).

In the above example the node $v_4$ has out-degree 3 and hence the weight on each of its outgoing edges $\{v_4v_1, v_4v_4, v_4v_5\}$ is $1/3$. Assume that a web-surfer starts the surfing at the web-page $v_1$. It has two outgoing edges and with equal probability $(1/2)$ the surfer decides to follow one of them, say $v_3$. Now at $v_3$, he/she decides between two possible outgoing edges and picks one of them with

[5] Sergey Brin and Lawrence Page. Reprint of: The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 56(18):3825–3833, 2012

[6] Sergey Brin, Rajeev Motwani, Lawrence Page, and Terry Winograd. What can you do with a web in your pocket? *IEEE Data Eng. Bull.*, 21(2):37–47, 1998

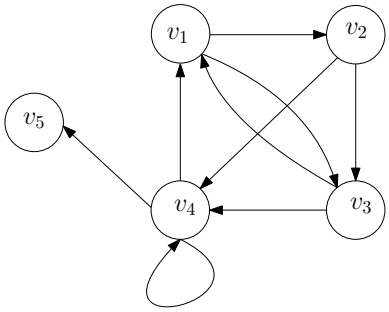



Figure 4.5: Web graph with 5 nodes.

$$M = \begin{array}{c} \begin{matrix} v_1 & v_2 & v_3 & v_4 & v_5 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 1/2 & 1/3 & 0 \\ 1/2 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 1/3 & 0 \\ 0 & 0 & 0 & 1/3 & 0 \end{bmatrix} \end{array}$$

Figure 4.6: Matrix $M$ corresponding to Figure 4.5.

equal probability, say $v_4$. At $v_4$ there are three possibilities and with equal probability (to return to $v_1$, to stay at $v_4$, or to advance to $v_5$) the surfer chooses to go to $v_5$. Node $v_5$ has no outgoing edges (its column is zero) and hence the web surfer is stuck. To overcome this, we say that the web-surfer jumps to a random page and restarts the whole process. This can be reflected by modifying the above matrix to the following:

$$Q = \begin{array}{c} \begin{matrix} v_1 & v_2 & v_3 & v_4 & v_5 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 1/2 & 1/3 & 1/5 \\ 1/2 & 0 & 0 & 0 & 1/5 \\ 1/2 & 1/2 & 0 & 0 & 1/5 \\ 0 & 1/2 & 1/2 & 1/3 & 1/5 \\ 0 & 0 & 0 & 1/3 & 1/5 \end{bmatrix} \end{array}$$

Note that $M$ corresponding to Figure 4.6 is updated to reflect that from node $v_5$, with equal probability, the surfer will go to any of the 5 web-pages.

The creators of Google, in addition to the above modifications, suggested the following. During the surfing, the web-surfer at each of the node (i.e. a web-page) flips a coin. If the outcome is Heads, it follows the outlined approach using the matrix $Q$. But if the outcome is tails, it 'teleports' to a page, chosen uniformly at random among all the webpages, and continues the surfing from there. Let the probability of heads be $\alpha$, then we can express the transition matrix followed by Google as $K = \alpha Q + \frac{1-\alpha}{n} E$, where $E$ is an $n \times n$ matrix and all of its entries are 1. We make a few remarks about $K$. Each of its entries $> 0$, and the entries within each column sums to 1. Hence $K$ is Markov matrix corresponding to a aperiodic irreducible Markov chain. Its largest eignenvalue is 1 and its corresponding eigenvector has positive entries, they add to 1, and corresponds to the steady state of $K$. Thus, the values in this eigenvector corresponds to the page rank of the web-pages.

Note that for the purpose of the computation of the page ranks, since $K$ is an extremely large matrix, it is not advisable to compute its eigenvector corresponding to its principal eigenvalue directly. This requires executing Gaussian elimination and it has relatively large computational complexity. The computational issues are addressed by exploiting the fact that $Q$ is extremely sparse and $E$ is a special matrix. For a vector $v = (1/n, \ldots, 1/n)$ corresponding to the uniform probability distribution of a random web surfer initially, the computation of $Kv = \alpha Qv + \frac{1-\alpha}{n} Ev$ can exploit the sparsity of $Q$ and the properties of $E$. Similarly $K^2 v = K(Kv) = Kv'$ has a similar computational flavour as we are again multiplying $K$ by a vector $v'$. Thus,

we can repeatedly compute this product and stop when we think the successive vectors are very close to each other and the computation has converged. Hopefully, the resulting vector represents the steady state and we can deduce the page rank of each of the web page.

## 4.7 Exercises

**4.1** *Let $q_1, \ldots, q_n$ be a set of $n$ independent orthonormal vectors in $\mathbb{R}^m$. Let $Q$ be a $m \times n$ matrix, where its $i$-th column is the vector $q_i$, for $i = 1, \ldots, n$. Show that $Q^T Q = I$, where $I$ is $n \times n$ identity matrix. Show that if $m = n$, then $Q^{-1} = Q^T$. That is, the inverse of a square matrix of orthonormal vectors is its transpose.*

**4.2** *Let $A$ be a $m \times n$ matrix where all its columns are linearly independent. Show that $A^T A$ is invertible. (Hint: To show that a matrix $B$ is invertible, we can show that $Bx = 0$ only for $x = 0$.)*

**4.3** *Let $A$ be a square matrix, where each of its diagonal entry is strictly larger than the sum of all other entries within that row. Show that $A$ is invertible.*

**4.4** *Show that the eigenvalues of a square matrix $A$ is same as that of $A^T$. Do they have the same eigenvectors?*

**4.5** *What are eigenvalues and eigenvectors of an identity matrix.*

**4.6** *For a square matrix $A$ show that the product of its eigenvalues equals the determinant of $A$. (Hint: Consider the characteristic polynomial and set $\lambda = 0$.)*

**4.7** *For a square matrix $A$ show that the product of its eigenvalues equals the determinant of $A$. (Hint: Consider the characteristic polynomial and set $\lambda = 0$.)*

**4.8** *For a square matrix $A$ show that the sum of its eigenvalues equals the sum of the diagonal entries of $A$ (called the* trace *of $A$).*

**4.9** *Show that for a real symmetric matrix $S = \left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right]$, its eigenvectors are $x_1 = \left[\begin{smallmatrix} b \\ \lambda_1 - a \end{smallmatrix}\right]$ and $x_2 = \left[\begin{smallmatrix} \lambda_2 - c \\ b \end{smallmatrix}\right]$ where $\lambda_1$ and $\lambda_2$ are the eigenvalues, respectively.*

**4.10** *For a square matrix $A$ , show that if all its eigenvalues are distinct, then its corresponding eigenvectors are linearly independent. First show that for a pair of distinct eigenvalues $\lambda_1 \neq \lambda_2$, their corresponding eigenvectors $v_1$ and $v_2$ are linearly independent. (Note that $v_1$ and $v_2$ are linearly independent if $c_1 v_1 + c_2 v_2 = 0$ then $c_1 = c_2 = 0$.)*

**4.11** *Show that for a Markov matrix $A$, and an identity matrix $I$, $A - I$ is singular, i.e. the determinant of $A - I$ is 0. Show that the rows of $A - I$ are not independent and hence $det(A - I) = 0$. This implies that 1 is an eigenvalue of a Markov matrix.*

**4.12** *Let $A$ be a square matrix and let $p(\lambda) = det(A - \lambda I) = 0$ be its characteristic polynomial. We are interested to find out whether all the roots (eigenvalues) of $p(\lambda)$ are distinct or not without actually computing its roots. Show that this can be achieved by finding the GCD of $p(x)$ and its derivative $p'(x)$.*

**4.13** *Show that in an irreducible Markov chain all the states are recurrent.*

**4.14** *In a Markov chain we say that a pair of states $(i, j)$* communicates *with each other if it is possible to reach from the state $i$ to the state $j$ with non-zero probability after a finite number of steps, and similarly it is possible to reach from the state $j$ to the state $i$ in a finite number of steps with non-zero probability. Show that 'communicates' is an equivalence relation. Show that an irreducible Markov chain has a unique equivalence class.*

**4.15** *Suppose that the underlying graph of a Markov chain is a bipartite graph. Can this chain be aperiodic?*

# 5
# *Minimum Spanning Trees*

## *5.1 Minimum Spanning Trees*

Let $G = (V, E)$ be an undirected connected graph with a cost function $w$ mapping edges to positive real numbers. A spanning tree is an undirected tree connecting all vertices of $G$. The *cost* of a spanning tree is equal to the sum of the costs of the edges in the tree. A *minimum* spanning tree (MST) is a spanning tree whose cost is minimum over all possible spanning trees of $G$. It is easy to see that a graph may have many MSTs with the same cost (e.g., consider a cycle on 4 vertices where each edge has a cost of 1; deleting any edge results in a MST, each with a cost of 3).

As in the CLRS book[1], we will describe the two main algorithms for building MSTs, Kruskal's and Prim's. Both of these algorithms are greedy algorithms and are based on the following generic algorithm (Algorithm 5.1). The algorithm maintains a subset of edges $A$, which is a subset of some MST of $G$.

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009

**Algorithm 5.1:** Generic-MST
**Input:** Graph $G$, cost function $w$
**Output:** A minimum spanning tree of $G$

```
1 A ← 0
2 while A ≠ MST do
3 |   find a safe edge {u, v} for A
4 |   A ← A ∪ {u, v}
5 end
6 return A
```

Intuitively this algorithm is straight-forward except for two pressing questions: What is a safe edge, and how do we find one? To answer these questions, we first need a few definitions.

**Cut** A cut $(S, V \setminus S)$ of $G = (V, E)$ is a partition of

vertices of $V$.

**Edge crossing a cut**
: An edge $(u, v) \in E$ crosses the $cut(S, V \setminus S)$ if one of its end point is in the set $S$ and the other one in the set $(V \setminus S)$.

**Cut respecting $A$**
: A cut $(S, V \setminus S)$ respects the set $A$ if none of the edges of $A$ crosses the cut.

**Light edge**
: An edge which crosses the cut and which has the minimum cost of all such edges.

**Theorem 5.1.1** *Let $A$ be a subset of the edges of $E$ which is included in some MST, and let $(S, V \setminus S)$ be a cut which respects $A$. Let $(u, v)$ be a light edge crossing the cut $(S, V \setminus S)$, then $(u, v)$ is safe for $A$.*

**Proof.** Assume that $T$ is a MST that includes $A$ (similarly, you may think of $A$ as being a subset of the edges of $T$, or being "the makings of" a MST). If $T$ includes the edge $(u, v)$ then $(u, v)$ is safe for $A$. If $T$ does not include $(u, v)$, then we will show that there is another MST, $T'$, that includes $A \cup \{(u, v)\}$, and this will prove that $\{(u, v)\}$ is safe for $A$. Since $T$ is a spanning tree, there is a path, say $P_T(u, v)$, from the vertex $u$ to vertex $v$ in $T$. By inserting the edge $(u, v)$ in $T$ we create a cycle. Since $u$ and $v$ are on different sides of the cut, there is at least one edge $(x, y) \in P_T(u, v)$ that crosses the $cut(S, T \setminus S)$. Moreover $(x, y) \notin A$, since the cut respects $A$. But the cost of the edge $(x, y)$ is at least the cost of the edge $(u, v)$, since edge $(u, v)$ is a light edge crossing the cut. Construct a new tree $T'$ from $T$ by deleting the edge $(x, y)$ in $T$ and inserting the edge $(u, v)$. Observe that the cost of the tree $T'$ is at most the cost of the tree $T$ since the cost of $(x, y)$ is at least the cost of $(u, v)$. Moreover $A \cup \{(u, v)\} \subset T'$ and $(x, y) \notin A$, hence edge $(u, v)$ is safe for $A$. ∎

The above theorem leads to the following corollary, where we fix a particular cut (i.e. the $cut(C, V \setminus C)$).

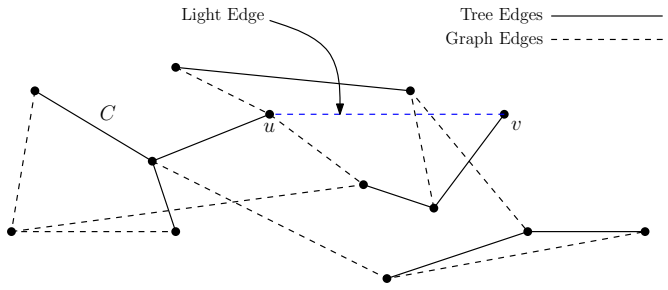



Figure 5.1: An example of Corollary 5.1.2. The edge $(u, v)$ connects $C$ to some other component of $G_A$ and is a light edge; it is therefore safe to add to the MST.

**Corollary 5.1.2** *Let $A \subset E$ be included in some MST. Consider the forest consisting of $G_A = (V, A)$, i.e., the graph with the same vertex set as $G$ but restricted to the edges in $A$. Let $C = (V_C, E_C)$ be a connected component of $G_A$. Let $(u, v)$ be a light edge connecting $C$ to another connected component in $G_A$, then $(u, v)$ is safe for $A$ (See Figure 5.1).*

## 5.2 *Kruskal's Algorithm for MST*

Proposed by Kruskal in 1956, this algorithm follows directly from Corollary 5.1.2. Here are the main steps. To begin with the set $A$ consists of only isolated vertices, and no edges (so, $|V|$ "connected" components in all).

1. Sort the edges of $E$ in non-decreasing order with respect to their cost.
2. Examine the edges in order; if the edge joins two components then add that edge (a safe edge) to $A$.

To implement Step 2, we do the following. Let $e_i$ be the edge under consideration, implying that all edges with a lesser cost than $e_i = (a, b)$ have already been considered. We need to check whether the endpoints $a$ and $b$ are within the same component or whether they join two different components. If the endpoints are within the same component, then we discard the edge $e_i$. Otherwise, since it is the next lightest edge overall, it must be the lightest edge between some pair of connected components, and so we know from Corollary 5.1.2 that it is safe to add to $A$. We will need to merge these two components to form a bigger component.

To accomplish all of this, we will need some data structure which supports the following operations:

- MAKE-SET$(v)$ - create a new set containing only the vertex $v$.
- FIND$(v)$ - Find the set which presently contains the vertex $v$.
- MERGE$(V_x, V_y)$ - Merge the two sets $V_x$ and $V_y$ together such that FIND will work correctly for all vertices in merged set.

We can implement this data structure as follows. For each vertex we keep track of which component it lies in using a label associated with the vertex. Initially each vertex belongs to its own component, which is done with MAKE-SET. During the algorithm the components will be merged, and the labels of the vertices will be updated. Assume that we need to merge the two components $V_a$ and $V_b$ corresponding to the end points $a$ and $b$ of the edge $e_i = (a, b)$. We use FIND$(a)$ and FIND$(b)$ to get the sets $V_a$ and $V_b$ respectively. We

then call MERGE which will relabel all of the vertices in one of the components to have the same labels as the vertices of the other. The component which we relabel will be the one which is smaller in size. Given such a data structure, we can implement Kruskal's algorithm as in Algorithm 5.2.

**Algorithm 5.2:** Kruskal-MST
**Input:** Graph $G = (V, E)$, cost function $w$
**Output:** A minimum spanning tree of $G$

```
1  A ← ∅
2  foreach v ∈ V do
3  |  MakeSet(v)
4  end
5  sort the edges of E in non-decreasing order w.r.t. w
6  foreach e = {a, b} ∈ E, where e is taken in sorted order do
7  |  V_a ← Find(a)
8  |  V_b ← Find(b)
9  |  if V_a ≠ V_b then
10 |  |  A ← A ∪ {e}
11 |  |  Merge(V_a, V_b)
12 |  end
13 end
14 return A
```

Let us analyze the complexity of Kruskal's algorithm. Sorting the edges takes $O(|E| \log |E|)$ time. The test for an edge, whether it joins two connected components or not, can be done in constant time. (In all $O(E)$ time for all edges.) What remains is to analyze the complexity of merging the components which can be bounded by the total complexity of relabeling the vertices. Consider a particular vertex $v$, and let us estimate the maximum number of times this will be relabeled. Notice that the vertex gets relabeled only if it is in a smaller component and its component is merged with a larger one. Hence after merging, the size of the component containing $v$ becomes at least double. Since the maximum size of a component is $|V|$, this implies that $v$ can be relabeled at most $\log_2 |V|$ times. Therefore, the total complexity of the Step 2 of the algorithm is $O(|E| + |V| \log |V|)$ time. These results are summarized in the following theorem.

**Theorem 5.2.1 (Kruskal)** *A minimum (cost) spanning tree of an undirected connected graph $G = (V, E)$ can be computed in $O(|V| \log |V| + |E| \log |E|)$ time.*

## 5.3 *Prim's MST algorithm*

Prim's algorithm is very similar to Dijkstra's single source shortest path algorithm[2], and, in fact, their complexity analysis will be the same. Here the set $A$ at any stage of the algorithm forms a tree, rather than a forest of connected components as in Kruskal's. Initially the set $A$ consists of just one vertex. In each stage, a light edge is added to the tree connecting $A$ to a vertex in $V \setminus A$.

The key to Prim's algorithm is in selecting that next light edge efficiently at each iteration. For each $v \in V \setminus A$, we keep track of the least cost edge which connects $v$ to $A$, and the cost of this edge is used as the "key" value of $v$. These key values are then used to build a priority queue $Q$. See Figure 5.2 for an example of these sort of light edges.

[2] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959

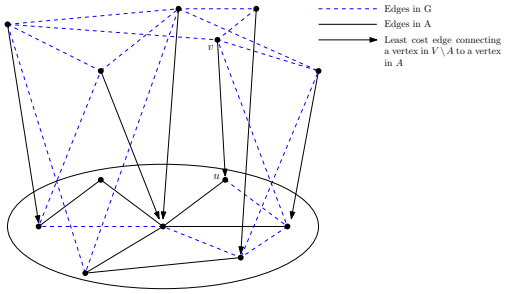



Figure 5.2: Example of Prim's Algorithm showing the set $A$ (encircled) and the least cost edges associated with each vertex in $V \setminus A$.

In each step of the algorithm, the vertex $v$ with the least priority is extracted out of $Q$. Suppose that corresponds to the edge $e = \{u, v\}$, where $u \in A$, then observe that $e$ is a safe edge since it is the light edge for $cut(A, V \setminus A)$. We update $A := A \cup \{e\}$. Finally, after extracting $v$ out of $Q$, we need to update $Q$. The details are explained in Algorithm 5.3.

The vertices that are in the set $A$ at any stage of the algorithm are the vertices in $V \setminus Q$, i.e., the ones that are not in $Q$. $kev(v)$ is the weight of the light edge $\{v, \pi(v)\}$ connecting $v$ to some vertex in the MST $A$. Notice that the *key* value for any vertex starts at infinity, when it is not adjacent to $A$ via any edge, and then keeps decreasing.

Let us analyze the complexity of the algorithm. The main steps are the priority queue operations, namely *decrease-key* and *extract-min*. We perform $|V|$ extract-min operations in all, one for each vertex. We also perform $O(|E|)$ decrease-key operations, one for each edge. The following table shows the complexity of these operations depending on the type of priority queue you choose. These complexities are per operation, although the complexities of Fibonacci Heaps are *amortized* (kind of an average over the worst possible scenario! - more on that later).

**Algorithm 5.3:** PRIM-MST
**Input:** Graph $G = (V, E)$, cost function $w$, root vertex $r$
**Output:** A minimum spanning tree of $G$

```
1  foreach v ∈ V do
2  |  key(v) ← ∞
3  |  π(v) ← nil /* π keeps track of the parent of a vertex
   |     in the tree.                                    */
4  end
5  key(r) ← 0
6  Q ← V /* Priority queue consists of vertices with their
      key values                                         */
7  while Q ≠ ∅ do
8  |  u ← Extract-Min(Q)
9  |  foreach v adjacent to u do
10 |  |  if v ∈ Q and w(u, v) < key(v) then
11 |  |  |  π(v) ← u
12 |  |  |  key(v) ← w(u, v)
13 |  |  end
14 |  end
15 end
```

| | Binary Heaps | Fibonacci Heaps |
|---|---|---|
| Extract-min | $O(\log n)$ | $O(\log n)$ |
| Decrease-key | $O(\log n)$ | $O(1)$ |

## 5.4 *Randomized Algorithms for Minimum Spanning Trees*

Here we discuss some results related to randomized algorithms for computing minimum spanning trees. These results are based on Section 10.3 of Raghavan and Motwani's book on Randomized Algorithms [3] and T. Chan's simplified analysis from [4]. Assume that all edge weights are distinct and hence there is a unique MST in the given graph $G = (V, E)$. The randomized algorithm uses a few concepts which are discussed in the following subsections followed by the actual algorithm itself in Section 5.4.3. The first concept is an algorithm due to Boruvka from 1926[5] which helps us to reduce the number of vertices in the graph. The second is about heavy and light edges with respect to a spanning tree.

[3] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, New York, NY, USA, 1995

[4] Timothy M. Chan. Backwards analysis of the Karger-Klein-Tarjan algorithm for minimum spanning. *Inf. Process. Lett.*, 67(6):303–304, 1998

[5] Otakar BorÅŕvka. O jistém problému minimálním. *PrÃące mor. pÅŹÃ■rodovÄŻd. spol. v BrnÄŻ III*, 3:37–58, 1926; and Otakar BorÅŕvka. Příspěvek k řešení otázky ekonomické stavby elektrovodních sítí. *Elektrotechnickỳ obzor*, 15:153–154, 1926

### 5.4.1 *Boruvka's Algorithm*

Observe that for any vertex $v \in V$, the edge, say $\{v, w\}$, with the minimum weight incident to that vertex will be included in the MST, as per Corollary 5.1.2. This leads to a simple way to compute the MST as given in Algorithm 5.4.

**Algorithm 5.4:** Boruvka-MST
**Input:** Graph $G$, cost function $w$ (all costs distinct)
**Output:** A minimum spanning tree $T$ of $G$

```
1 T ← ∅
  // Each iteration of this loop is called a Phase
2 while G contains more than a single vertex do
3 |   Mark the edges with the minimum weight incident to each
  |    vertex. Add these edges to T
4 |   Identify the connected components of the marked edges
5 |   Replace each connected component by a vertex
6 |   Eliminate all self loops. Eliminate all multiple edges between
  |    a pair of vertices, except the edge with the minimum weight
7 end
8 return T
```

A few observations about this algorithm:

- Let $G'$ be the graph obtained from $G$ after contracting the edges in a single phase of the algorithm. Then the MST of $G$ is the union of the contracted edges from that phase and the edges in the MST of $G'$.
- In each contraction phase, the number of vertices in the graph is reduced by at least a half. Hence there will only be $O(\log |V|)$ phases in all.
- Each phase can be implemented in $O(|V| + |E|)$ time and so we obtain yet another MST finding algorithm. Total running time for this algorithm is $O(|E| \log |V|)$.

### 5.4.2 *Heavy Edges*

We've already seen the definition of a light edge. Now we examine edges which are not light.

Let $F$ be any spanning tree of $G$ (in particular, $F$ may or may not be a minimum spanning tree). Consider any two vertices, say $u$ and $v$, of $G$ and there is a unique path $P(u, v)$ between them in $F$. Let $w_F(u, v)$ be the edge with the maximum weight along this path.

We say an edge $\{u, v\}$ is $F$-heavy if the weight of this edge is larger than the weight of each of the edges on the unique path between $u$ and $v$ in $F$. More formally, we define an edge $\{u, v\} \in E$ to be $F$-heavy if $w(u, v) > w_F(u, v)$, otherwise it is $F$-light. Observe that by this definition, all edges of $F$ are $F$-light (every edge of a path in $F$ is at most as heavy as the heaviest edge in that path of $F$).

**Lemma 5.4.1** *Let $F$ be a spanning tree of $G$ which is not necessarily minimum. If an edge of $G$ is $F$-heavy then it does not lie in the Minimum Spanning Tree of $G$.*

**Proof.** It is left for you to prove it formally. The proof proceeds by contradiction. Assume that the edge $e = \{u, v\} \in E$ is $F$-heavy, that $T$ is an MST of $G$, and that $e \in T$ (so, we are talking about two trees here, $T$, which is known to be an MST, and $F$, which may be one as well). Consider the edges on the path $P(u, v)$ in $F$. Add all these edges to $T$ and remove $e$ from $T$, to obtain a graph $G'$ (that is, $G' = T \cup P(u, v) \setminus \{e\}$, but since it may have some cycles, we cannot call it a tree). $G'$ is still connected. Remove edges from $P(u, v)$ one-by-one from $G'$ until $G'$ is once again a tree. Observe that this new tree is still a spanning tree, but it must have weight lower than that of $T$, contradicting the minimality of $T$. ∎

It is very important to note that because of the way $F$-light is defined, an edge which is $F$-light may or may not be in the MST. For example, if we construct $F$ such that the heaviest edge of $E$ is in $F$, that edge will be counted as $F$-light, even though it may not be present in any minimum spanning tree.

### 5.4.3 *Randomized Algorithm*

Algorithm 5.5 gives the main steps of the randomized algorithm we have been discussing. The analysis is based on Timothy Chan's paper[6].

[6] Timothy M. Chan. Backwards analysis of the Karger-Klein-Tarjan algorithm for minimum spanning. *Inf. Process. Lett.*, 67(6):303–304, 1998

**Theorem 5.4.2** *Algorithm 5.5 correctly computes the MST of $G$ in $O(|V| + |E|)$ time.*

**Proof.** The correctness of the algorithm is straightforward. To estimate the complexity, the crux is in estimating the size of the set $E_3$, i.e., the size of the set of $F_2$-light edges in $E_1$. We will prove in the Sampling Lemma (Lemma 5.4.4) that for a random subset $R \subset E$, the expected number of edges that are light with respect to MST$(R)$ is at most $(|E| \cdot |V_1|)/|R|$. In our case, the expected value of $|R| = |E_1|/2 \leq |E|/2$, and hence the expected number of $F_2$-light edges will be at most $2|V_1| \leq |V|/4$. Hence the running time of this algorithm is given by the recurrence

**Algorithm 5.5:** Randomized-MST
**Input:** Connected Graph $G = (V, E)$ with distinct edge weights
**Output:** A minimum spanning tree $T$

1 Execute 3 phases of Boruvka's algorithm (reduces number of vertices). Let the resulting graph be $G_1 = (V_1, E_1)$, where $|V_1| \leq |V|/8$ and $|E_1| \leq |E|$. Let $C$ be the set of contracted edges. (Running time: $O(|V| + |E|)$)
2 Random Sampling: Choose each edge in $E_1$ with probability $p = 1/2$ to form the set $E_2$ and obtain the sampled graph $G_2 = (V_2 = V_1, E_2)$.
3 Compute Recursively the Minimum Spanning Tree of $G_2$, and let it be $F_2$. ($T(|V|/8, |E|/2)$)
4 Verification: Compute the set of $F_2$-light edges in $E_1$, and let this set be $E_3$. ($O(|V_1| + |E_1|)$ time)
5 Final MST: Compute MST, $F_3$, of the graph $G_3 = (V_3 = V_1, E_3)$. ($T(|V|/8, |V|/4)$)
6 **return** MST of $G$ as $C \cup F_3$.

$$T(|V|, |E|) = O(|V| + |E|) + T(|V|/8, |E|/2) + T(|V|/8, |V|/4),$$

which magically solves to $O(|V| + |E|)$. ∎

Before we describe the Sampling Lemma here are some technicalities. Consider that we are sampling the edges of the graph $G = (V, E)$, and the sampled edges form the subgraph $R$.

We use the notation $R$ for both the set of edges as well as the sampled graph. Since we are sampling the edges, it is possible that the sampled graph $R$ of the graph $G$ is not connected, and hence there will not be any minimum spanning tree. To ensure connectedness, we will fix any spanning tree $T_0$ of $G$, consisting of $|V| - 1$ edges and we will consider the minimum spanning tree of $R \cup T_0$, denoted as $MST(R)$.

**Lemma 5.4.3 (Observation about light edges)** *An edge $e \in E$ is light with respect to $MST(R)$ if and only if $e \in MST(R \cup \{e\})$.*

**Proof.** If $e = (u, v)$ is light then there is some edge $e'$ on the unique path between $u$ and $v$ in $MST(R)$ such that its weight, $w(e') = w_{MST(R)}(u, v)$, and hence $e$ can be added to $MST(R)$ and $e'$ can be removed to obtain MST of $R \cup \{e\}$. Therefore $e \in MST(R \cup \{e\})$.

Now suppose $e \in MST(R \cup \{e\})$. We need to show that $e$ is light with respect to $MST(R)$. Since $e$ is part of the MST, by definition it is light with respect to that MST. ∎

**Lemma 5.4.4 (Sampling Lemma)** *For a graph $G = (V, E)$ and a random subset $R \subset E$, of edges, the expected number of edges that are light with respect to $MST(R)$ is at most $(|E| \cdot |V|)/|R|$.*

**Proof.** Pick a random edge $e \in E$ (this choice is independent of the edges in $R$). We will prove that $e$ is light with respect to $MST(R)$ with probability at most $|V|/|R|$. From Lemma 5.4.3 we see that this is equivalent to finding the bound on the probability that $e \in MST(R \cup \{e\})$. Let $R' = R \cup \{e\}$. We will use a technique called *backward analysis*. First we analyze the probability on a fixed set $R'$, and then we will show that the expression obtained is not dependent on the elements of $R'$, but just the cardinality, and hence the probability holds unconditionally as well.

Instead of adding a random edge to $R$, we will think of deleting a random edge from $R'$. This is an easier proposition since we know the elements of $R'$, having just fixed it. $MST(R')$ has $|V| - 1$ edges, and $e$ is a random edge of $R'$, hence the probability that $e$ is an edge from $MST(R')$, given a fixed choice of $R'$, is $(|V| - 1)/|R'| \leq |V|/|R|$. This bound is independent upon the choice of the set $R'$, and holds unconditionally as well. ∎

## 5.5 MST Verification

This section is contributed by Gregory Bint. If I give you any tree $F$ derived from a graph $G = (V, E)$, can you identify whether $F$ is a minimum spanning tree of $G$?

A trivial method for determining this would be to run a known-correct algorithm such as Kruskal's or Prim's on $G$ and compare the output to $F$, however there are two main drawbacks to this approach:

1. It is too slow
2. The MST may not be unique, making a direct comparison difficult.

What we would like is to be able to calculate this in linear time with respect to the graph. The following lemma has been shown to be very useful in this respect, and it is used by virtually every MST verification algorithm.

**Lemma 5.5.1** *Let $F$ be a spanning tree of $G$, then $F$ is a minimum spanning tree of $G$ if and only if every edge in $E \setminus F$ is $F$-heavy.*

**Proof.** Let $P(u, v)$ be the unique tree path between $u$ and $v$ in $F$, and let $w_F(u, v)$ be the weight of the heaviest edge along that path. $w(e)$ or $w(u, v)$ is the weight of the edge $e$ having endpoints $u$ and $v$.

We first show that if $F$ is a MST, then every edge in $E \setminus F$ is $F$-heavy. Let $e$ be any edge in $E \setminus F$ with endpoints $u$ and $v$ and assume that $e$ is $F$-light. Note that $P(u,v) \cup \{e\}$ is a cycle. Let $e' = \{x,y\}$ be the edge corresponding to $w_F(u,v)$. Since $e$ is $F$-light, $w(e') > w(e)$, meaning we could replace $e'$ by $e$ in $F$ to obtain a lighter tree overall, contradicting the minimality of $F$.

For the other half of the proof, we show that if every edge in $E \setminus F$ is $F$-heavy, then $F$ is a MST of $G$. Again, we proceed by contradiction. Suppose that $F$ is not a MST of $G$, then we should be able to lower the weight of the tree by replacing some edges in $F$ with those from $E \setminus F$. But, for every $einE \setminus F$ with endpoints $u$ and $v$, we have that $w(u,v) > w_F(u,v)$, so exchanging $e$ for any other edge in $P(u,v)$ will increase the weight of $F$. ∎

Given the above lemma, a natural idea for an algorithm would be to try to classify every edge in the graph, and then check if each non-tree edge is in fact $F$-heavy. This turns out to be something which *is* possible: given a graph $G = (V,E)$ and a tree $F$, we can partition the edges of $G$ in two sets, the set of heavy edges and the set of light edges, with respect to $F$ in $O(|V| + |E|)$ time.

We will see that many of the algorithms for doing so fairly complex, although there has been recent progress in simplifying it somewhat.

### 5.5.1 *Overview of verification algorithms*

Here we look at a brief history of the literature on MST Verification. As hinted at above, every single one of the following methods uses Lemma 5.5.1 as its underpinning. This problem can also be restated as the following:

**Problem 5.5.2 (The Tree Path Maxima Problem)** *Let $F$ be a spanning tree of $G$, then we want to identify the cost of the heaviest edge along each tree path $P(u,v)$.*

Given an answer to Problem 5.5.2, we can perform a simple linear scan through the the edges of $G \setminus F$. For each edge $e \in G \setminus F$, we compare the cost of $e$ with the tree path of its endpoints. If every edge $e$ is heavier than its corresponding tree path maxima, then $F$ is a MST of $G$. We look at this sort of translation of the problem in more detail in Section 5.5.6.

Here is a timeline of some results:

- In 1979, Tarjan introduced a method which uses path compression[7] of trees to achieve a near-linear time of $O(m\alpha(m,n))$ where $\alpha$

[7] Robert Endre Tarjan. Applications of path compression on balanced trees. *J. ACM*, 26(4):690–715, October 1979

is the Inverse Ackermann function.

- In 1984, Komlós's provided an algorithm[8] which showed that only a linear number of *comparisons* of the edge costs would be sufficient to solve the problem, however the algorithm itself has significantly more than linear overhead.
- In 1992, Dixon *et al.*[9] combine methods from Tarjan's 1979 algorithm and Komlós's 1984 algorithm to produce the first linear time MST verification algorithm. The problem is divided into one large problem and several small problems, with the larger being attacked with path compression, and the smaller with a lookup scheme which is bounded in size by Komlós's algorithm.
- In 1994, Karger *et al.* present an algorithm for computing the MST of a graph in *expected* linear time.[10] While not a verification algorithm in itself, its output could be useful to help verify another tree, or to take the place of the other tree altogether (e.g., why bother verifying a potential MST in linear time when you can just *create* one!)
- In 1995, King produced another linear time MST verification method[11] which is a great deal simpler than that of Dixon *et al.*. In this method, Boruvka's algorithm is used to reduce a general tree down to one which can be handled entirely by the full branching tree base case of Komlós's algorithm, which is simpler than his algorithm for general trees.
- In 2010, Hagerup simplifies King's method[12] even further and provides an implementation in the *D* programming language. Like King's method, Hagerup continues to use Komlós's *full branching tree* case, but eschews complex edge encoding schemes in favour of a richer logical data type.

We will walk through parts of Komlós's algorithm, Dixon *et al.*'s algorithm, King's algorithm, and finally Hagerup's algorithm as we piece together the tools needed for a reasonably simple approach to solving this problem.

[8] J. Komlos. Linear verification for spanning trees. In *Foundations of Computer Science, 1984. 25th Annual Symposium on*, pages 201–206, 1984; and J. KomlÃşs. Linear verification for spanning trees. *Combinatorica*, 5(1):57–65, 1985

[9] B. Dixon, M. Rauch, and R. Tarjan. Verification and sensitivity analysis of minimum spanning trees in linear time. *SIAM Journal on Computing*, 21(6):1184–1192, 1992

[10] David R. Karger, Philip N. Klein, and Robert E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *J. ACM*, 42(2):321–328, March 1995; and Philip N. Klein and Robert E. Tarjan. A randomized linear-time algorithm for finding minimum spanning trees. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, STOC '94, pages 9–15, New York, NY, USA, 1994. ACM

[11] Valerie King. A simpler minimum spanning tree verification algorithm. In SelimG. Akl, Frank Dehne, JÃűrg-RÃijdiger Sack, and Nicola Santoro, editors, *Algorithms and Data Structures*, volume 955 of *Lecture Notes in Computer Science*, pages 440–448. Springer Berlin Heidelberg, 1995; and V. King. A simpler minimum spanning tree verification algorithm. *Algorithmica*, 18(2):263–270, 1997

[12] Torben Hagerup. An even simpler linear-time algorithm for verifying minimum spanning trees. In Christophe Paul and Michel Habib, editors, *Graph-Theoretic Concepts in Computer Science*, volume 5911 of *Lecture Notes in Computer Science*, pages 178–189. Springer Berlin Heidelberg, 2010

### 5.5.2 *Komlós's Algorithm*

In 1984, Komlós[13] gives an algorithm of sorts which can solve Problem 5.5.2 in $O(n + m)$ *comparisons*. He does not provide an implementable algorithm, however, and there are other factors of overhead in his method which would drive up the actual cost of a straight implementation. Nevertheless, this method of breaking down the problem is built upon by later papers, notably Dixon *et al.* in 1992, and King, which we cover in Section 5.5.4.

[13] J. Komlos. Linear verification for spanning trees. In *Foundations of Computer Science, 1984. 25th Annual Symposium on*, pages 201–206, 1984; and J. KomlÃşs. Linear verification for spanning trees. *Combinatorica*, 5(1):57–65, 1985

Komlós begins by considering two special cases of spanning trees. In each case, we consider $F$ to be a directed tree with edges oriented away from the root. Additionally, we shift the edge costs down to their lower endpoint vertices, as this simplifies the conceptual model.

The first case occurs when the tree is a path. For the path, we construct a *symmetric order heap*, $H$, which a tree with both the binary search property on the ordering determined by the path, and the (maximum) heap property determined by the vertex costs. The root of $H$ represents the heaviest vertex, and the heaviest vertex of any path from $u$ to $v$ is found at $LCA(u, v)$. Determining the LCA of two vertices in a tree can be accomplished in several ways and is covered elsewhere in these notes, but Komlós cites Harel[14] specifically.

[14] Dov Harel. A linear time algorithm for the lowest common ancestors problem. In *Foundations of Computer Science, 1980., 21st Annual Symposium on*, pages 308–319, 1980

The second case is somewhat more interesting and is concerned with processing *full branching trees*. Not to be confused with full *binary* trees, a full branching tree is defined as one where every leaf is at the same level, and every internal vertex has *at least* 2 children. Let $F$ be our full branching tree with root $r$ and all edges directed away from $r$.

We need to calculate the maximum cost edge of every path through $F$. Given a vertex $y$, let $A(y)$ be the set of all paths through $F$ which contain $y$. That is $A(y) = \{P(x, z) | x \geqq y \geqq z\}$ where $u \geqq v$ denotes that $u$ is a predecessor of $v$ (or, $u$ may equal $v$). Since $F$ is directed away from the root, this means that $u$ is at least as close to the root than $v$. Note that if $F$ was not directed, then it might be possible for $x \geqq y \geqq z$ to hold even though $y \notin P(x, y)$. Given $A(y)$, let $A^*(y)$ be the set of all paths through $y$, but restricted to just the interval $[r, y]$; that is, just the subpath from the root down to $y$.

We process $F$ one level at a time, starting from the root, finding the maximum weights of all paths in the sets $A^*(y)$. To do this, assume that we have calculated the maximum costs in all such paths up to level $i$, and that we are now trying to process some vertex $y$ on level $i + 1$. Let $\bar{y}$ be the parent of $y$. Since $\bar{y}$ resides on level $i$, we know the maximum cost of all paths in $A^*(\bar{y})$.

The key observation to make here is the following.

**Property 5.5.3** *Consider two paths $P(x, \bar{y})$ and $P(x', \bar{y})$. If $x$ is a predecessor of $x'$ then the maximum cost in $P(x, \bar{y})$ is at least as large as the maximum cost in $P(x', \bar{y})$ (since, under these conditions, $P(x', \bar{y})$ is a subpath of $P(x, \bar{y})$).*

In $A^*(\bar{y})$, the shortest path is $P(\bar{y}, \bar{y})$ while the longest is $P(r, \bar{y})$. By the above property, the maximum costs form a non-decreasing sequence with respect to the length of the path. That is, we can order the maximum costs by considering the path length. This observation about the ordering helps us while building $A^*(y)$, as we can use a

binary search insertion of $f(y)$ to compare $f(y)$ against all path cost maximums in $A^*(\bar{y})$ simultaneously.

By now you should be asking "How are all these sets like $A(y)$ and $A^*(\bar{y})$ created, copied, and updated?" As far as Komlós's paper is concerned, the answer is "slowly". Essentially what Komlós shows us is that a linear number of *comparisons* are sufficient to determine maximum path costs, however finding those comparisons is left open.

The remainder of Komlós's paper details how these two primitive cases can be applied to any general tree, however this method is fairly complex, and as he states, results in too much overhead. No implementable algorithm is given in this paper.

### 5.5.3 *Dixon* et al.*'s Technique*

This technique has the distinction of being the first to achieve a linear running time, requiring $O(m)$ time on a graph with $n$ vertices and $m$ edges. The underlying process is fairly complex, however, and involves first preprocessing the graph into a suitable form.

The preprocessing itself is interesting as it shows a method of massaging a graph into a more attractive form for the problem at hand without affecting anything about the spanning tree that we wish to verify. The preprocessing involves the following steps.

For a given graph $G = (V, E)$ and spanning tree $F$, not necessarily minimum, we choose an arbitrary vertex $r$ to be the root of $F$. Now consider any non-tree edge $\{v, w\}$ with cost $c(v, w)$ and lowest common ancestor $u$. If $u$ is not one of $v$ or $w$, then this implies that $v$ and $w$ are not *related*; that is, the $v$ is neither ancestor nor descendant of $w$. In such a case, we replace the edge $\{v, w\}$ by $\{v, u\}$ and $\{u, w\}$, each with cost $c(v, w)$. $F$ is unchanged by this process and, more importantly, the maximum weight along $P(v, w)$ is also unchanged, which preserves the current minimality of $F$. There are several linear time algorithms for finding lowest common ancestors in a tree (e.g., Harel & Tarjan, 1984[15] or Schieber & Vishkin, 1988[16]).

Taken over the entire graph, this will at most double the number of non-tree edges. When completed, every non-tree edge in $F$ is a backedge.

In the second stage of preprocessing, we will mark several vertices. We can imagine these marks as subdividing the tree into edge-disjoint subtrees where a marked vertex represents a "root", and any marked descendants are ignored.

The choice of which vertices to mark is based on subtree size. Using a post-order traversal, for each vertex $v$ we calculate $h = 1 + \sum\{s(w) | w \text{ is a child of } v\}$. Let $g$ be a small integer, then if $h \leq g$ we assign $s(v) := h$, otherwise $s(v) := 1$ and $v$ becomes marked. We

[15] D. Harel and R. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984

[16] Baruch Schieber and Uzi Vishkin. On finding lowest common ancestors: Simplification and parallelization. In JohnH. Reif, editor, *VLSI Algorithms and Architectures*, volume 319 of *Lecture Notes in Computer Science*, pages 111–123. Springer New York, 1988

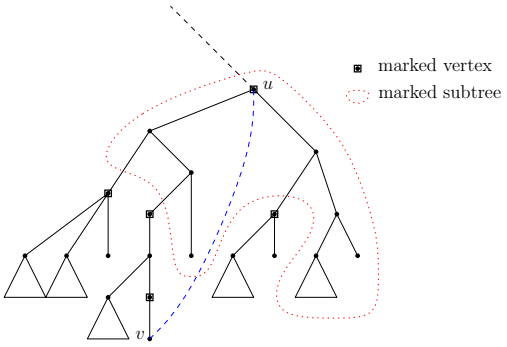



Figure 5.3: An example tree with marked vertices. The marked subtree defined by $u$ is enclosed in red.

will look more at the specific choice of $g$ later. Note the following important properties resulting from this process:

1. The number of marked vertices, and hence the number of subtrees, is at most $(n-1)/g+1$.
2. Considering any subtree, if its root (marked vertex) is deleted, along with incident edges, we get a collection of disjoint trees, each with size at most $g$. We call each of these a *microtree*.

Following that process, $r$ is also marked, although it will probably not have Property 2.

A final phase of edge replacements will ensure that all backedges either span two vertices belonging to the same microtree, or span between microtrees and marked vertices only (i.e., the edge $\{u,v\}$ in Figure 5.3 will be replaced).

To accomplish this, we first build the tree $F'$ whose vertex set consists of all of the marked vertices of $F$, and where, for two vertices $s$ and $t$ in $F'$, $s$ is the parent of $t$ (i.e, there is a tree edge between them) if $s$ is the first marked vertex that we encounter when walking from $t$ to $r$. We call $T'$ the *macrotree*.

We can now eliminate the "long" edges, like $\{u,v\}$, by doing the following. Let $p(v)$ be the nearest marked vertex to $v$ which is a proper ancestor of $v$. Note that if $v$ is marked then $p(v) \neq v$. We also assume that $p(r)$ is undefined (but it won't be needed anyway). We can calculate $p(v)$ for the entire tree using a depth-first search. For every non-tree edge $\{u,v\}$, assume w.l.o.g. that $u$ is an ancestor of $v$ and find $p(u)$ and $p(v)$. If $p(u) = p(v)$, then $u$ and $v$ are part of the same microtree (recall that the root of a microtree is *not* marked).

Otherwise, if $p(u) \neq p(v)$, then we know that there is at least one marked vertex between them. Let $r_1 = u$ if $u$ is marked, or $r_1 = p(u)$ if $u$ is not marked. Similarly, let $r_3 = v$ if $v$ is marked, or $r_3 = p(v)$ if $v$ is not marked. Let $r_2$ be the child of $r_1$ in $F'$ (note: $F'$, not $F$). We then replace $\{u,v\}$ by $\{u,r_2\}$, $\{r_2,r_3\}$, and $\{r_3,v\}$, skipping any edge

that creates either a self-loop or which duplicates a tree edge. For edges which we did not skip, assign the cost $c(u, v)$. As in the first phase of edge replacements, assigning this cost preserves the current minimality of $F$.

With this preprocessing finished, we have now divided the problem into one large tree rooted at $r$ with several microtrees around the periphery. The authors complete the process by using Tarjan's Path Compression on the large tree.

The microtrees are processed in a very different way. Essentially, the authors precalculate all possible minimum spanning trees on graphs containing at most $g$ vertices. Leveraging Komlós's result, they show that for any such input, the corresponding decision tree for comparing edges and determining minimality is not too big. The choice of $g$ is such that the total size of these precalculations is only $O(n)$, which places $g$ in the neighbourhood of $O(\log \log n)$ [17].

[17] V. King. A simpler minimum spanning tree verification algorithm. *Algorithmica*, 18(2):263–270, 1997

### 5.5.4 *King's Method*

Presented by King[18] in 1995, this method is not the first MST verification algorithm to achieve linear time (that falls to Dixon *et al.*[19]), however it is quite a bit simpler. King's method uses Boruvka's algorithm in a clever way to change any input tree into a full binary tree, which can then be entirely processed by the appropriate case presented by Komlós. This method requires linear time and space in the unit-cost RAM model with $\Theta(\log n)$ word size.

[18] Valerie King. A simpler minimum spanning tree verification algorithm. In SelimG. Akl, Frank Dehne, JÃűrg-RÃijdiger Sack, and Nicola Santoro, editors, *Algorithms and Data Structures*, volume 955 of *Lecture Notes in Computer Science*, pages 440–448. Springer Berlin Heidelberg, 1995; and V. King. A simpler minimum spanning tree verification algorithm. *Algorithmica*, 18(2):263–270, 1997

[19] B. Dixon, M. Rauch, and R. Tarjan. Verification and sensitivity analysis of minimum spanning trees in linear time. *SIAM Journal on Computing*, 21(6):1184–1192, 1992

#### *Boruvka Tree Property*

The first step is to take our input tree $F$ and convert it to a full binary tree. This is accomplished by running Boruvka's algorithm on the tree $F$ (we usually would run Boruvka's on an entire graph, but not in this case). As Boruvka's runs on $F$, we can build a new tree $B$ which represents the *execution* of the algorithm on $F$, rather than a modification of $F$ itself.

Algorithm 5.6 details the construction of $B$. In the first step, a leaf is added to $B$ for each vertex of $F$, so we already know that $|B| \geq |F|$. In fact, $B$ will have at most twice as many vertices as $F$ when we are finished. The algorithm proceeds by colouring the vertices and edges to represent subtrees within $F$, such that any vertices connected along a coloured (blue) path is considered part of the same subtree.

Refer to Figure 5.4 for an example of the algorithm's execution. Note the following important properties which ensure that $B$ is a full branching tree.

1. In each step of Loop 1, an edge joins two blue trees into one.

**Algorithm 5.6:** FullBranchingTree
**Input:** A spanning tree $F = (V, E)$ with distinct edge weights
**Output:** A full branching tree $B$ satisfying Lemma 5.5.4

```
1  Initialize B as an empty tree
2  foreach vertex v of V do
3  |  Colour v blue, considering it as a singleton tree
4  |  Add the leaf f(v) to B
5  end
6  while there is more than one blue tree do
   |  // Loop "1", joins blue trees together
7  |  foreach blue tree a do
8  |  |  Select a minimum cost edge e incident to a and colour it
   |  |   blue
9  |  end
   |  // Loop "2", updates B
10 |  foreach new blue tree t do
11 |  |  Add f(t) to B
12 |  |  Let A be the set of trees joined into t in Loop 1
13 |  |  Add an edge {f(t), f(a)} for each a ∈ A
14 |  |  Set the cost of {f(t), f(a)} to that of edge selected by a in
   |  |   Loop 1 (i.e., e)
15 |  end
16 end
17 return B
```

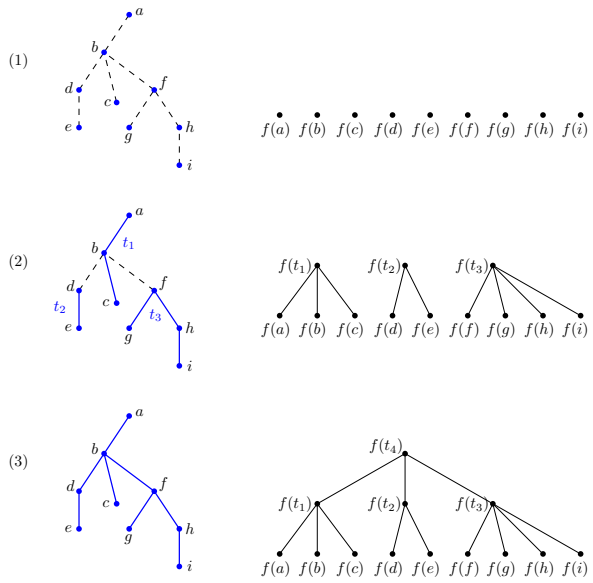



Figure 5.4: An example of the Full Binary Tree construction given by Algorithm 5.6. $F$ is shown on the left and $B$ on the right. Edge weights are not shown, so imagine that each tree chooses its minimum weight edge at each step.

---

2. In each phase of the while loop, every blue tree is combined by some edge with another blue tree. Thus, from every level of $B$, every vertex has a parent in the next level.

For every $v$ in $F$ there is a vertex $f(v)$ in $B$, and by construction we also have that for every path $F(x,y)$ there is a path $B(f(x),f(y))$. However, to show that there is any meaningful correspondence between these paths beyond their existence, we need the following lemma, presented as Theorem 1 in King's paper.

**Lemma 5.5.4** *Let $F$ be any spanning tree and let $B$ be the tree constructed by Algorithm 5.6. For any pair of vertices $x,y \in F$, the cost of the heaviest edge in $F(x,y)$ equals the cost of the heaviest edge in $B(f(x),f(y))$.*

**Proof.** Let the cost of an edge $e$ be denoted by $w(e)$. For every edge $e \in B(f(x),f(y))$, we will show that there is an edge $e' \in F(x,y)$ such that $w(e') \geq w(e)$.

Suppose that $e = \{a,b\}$ such that $a$ is the endpoint of $e$ which is farthest from the root. As $a$ is in $B$, $a = f(t)$ for some blue tree $t$, and $t$ must contain either $x$ or $y$, but not both. Similarly, $b = f(t')$ which is new blue tree consisting of $f(t)$ and others from the previous phase of the algorithm. Since $e \in B$, $e$ was selected by $t$.

Let $e'$ be the edge in $F(x,y)$ with exactly one endpoint in $t$. Since $e'$ is adjacent to $t$, $t$ would have considered $e'$. Since $t$ ultimately chose $e$, it must be that $w(e') \geq w(e)$ since $t$ chooses the edge with minimum cost.

To finish the proof we also need to show the following: The cost of the heaviest edge in $F(x,y)$ is the cost of the heaviest edge in

$B(f(x), f(y))$. Let $e$ be the heaviest edge in $F(x, y)$ (for simplicity, assume that there is a unique such edge). If $e$ is ever selected by a blue tree which contains either $x$ or $y$, then $B(f(x), f(y))$ contains an edge with the same weight.

Otherwise, assume that $e$ is selected by some other blue tree $t'$ not containing $x$ or $y$. We know that $e$ is on the path from $x$ to $y$ in $F$, so $t'$ contained at least one intermediate vertex on that path. But since $F$ is a tree, if it contains an intermediate vertex of $F(x, y)$, it must be incident to at least two edges of $F(x, y)$. By our assumption, $e$ is the heaviest edge on this path, so $t'$ would have selected the other edge, giving a contradiction. ∎

The intuition with the last part of the above proof is that, since $e$ is the heaviest edge along $F(x, y)$, any blue tree which includes part of that path, but which does not yet include $x$ or $y$ always has another edge to select which brings it "closer" to $x$ or $y$.

King's algorithm now continues with $B$ rather than $F$, which maintains the path maximum cost property for each path in $F$, implying that if Lemma 5.5.1 holds for $B$ it will also hold for $F$.

The remainder of King's paper shows a bit-wise labeling scheme from for the vertices and edges of $B$ which exploits Property 5.5.3 of the full binary tree case presented by Komlós. We will now jump to Hagerup's method to conclude our verification method, which he wrote specifically to simplify away from this labeling scheme, but which otherwise picks up at exactly this point of the algorithm.

### 5.5.5 *Hagerup's Method*

Hagerup presents an algorithm for solving the Tree Path Maxima problem (TPM) rather than MST Verification, *per se*, but as we have mentioned, a solution to TPM implies a solution to MST Verification. A sketch of such a translation is given in the next section.

The input to Hagerup's method assumes that we are given a tree on $n$ vertices and a list of pairs $(u_1, v_1), \ldots, (u_m, v_m)$ such that in each pair $u_i$ is a proper ancestor of $v_i$. At most, this list would describe the endpoints of every root to leaf path in $B$, and every subpath of such a root to leaf path. Any subset is also permissible. In practice, we choose a subset equivalent to the non-tree edges of $G$, the graph containing the spanning tree $F$ we are trying to verify.

The basic algorithm involves collecting several types of information about each vertex. For every vertex $u$ in $B$, we store the depth $d(u)$, and, if $u$ is not the root $r$, we let $w(u)$ represent the cost of the edge from $u$ to its parent. For each $u$ we also build the following set:

$$D_u = \{d(u_i) | u_i \text{ is a proper ancestor of } u \text{ and } v_i \text{ is a descendant of } u\}$$

Simply put, $D_u$ stores the set of depths corresponding to proper ancestors of $u$ such that $u$ is in the subpath represented by some pair $(u_i, v_i)$ from the input.

We would also like to create the set $M_u$ for each $u$, which stores a subset of the ancestors of $u$ indicated by $D_u$. The choice of which ones are stored again exploits Property 5.5.3.

Consider any two successive ancestors $d$ and $d'$ of $u$ which are indicated by $D_u$ such that $d$ is closer to the root, and $d'$ is closer to $u$. Then $d \in M_u$ if the path maximum cost of the path $d \to u$ is greater than that of $d' \to u$. Put another way, we store only those ancestors of $u$ where there is an actual increase in path maximum cost between it and the previous (closer) ancestor.

This can still work out to be a lot of entries, however, and a lot of copying between vertices, which breaks linear time. Fortunately, Hagerup was able to find an alternate, yet equivalent set representation which does satisfy our needs, and our desired running time, using the *set infix* operator. The details of this operator and its equivalence to $M_u$ take a few pages to discuss and can be found in his paper.

### 5.5.6 *Putting it all together*

One way of applying all of the tools we have seen so far to build a complete MST Verification algorithm is as follows.

Taking a graph $G = (V, E)$ with spanning tree $F$, let $U = E \setminus F$ be the set of non-tree edges. We use King's method of using Boruvka's method to convert $F$ to the full branching tree $B$. Translate $U$ onto $B$ so that $\forall e = \{x, y\} \in U$ we create $e' = \{f(x), f(y)\}$ and call the resulting graph $G'$. We next apply Dixon *et al.*'s first preprocessing step to $G'$ to replace all cross edges with back edges. Let $U'$ be the set of non-tree edges in $G'$ after all of this.

The set $U'$ corresponds to the pairs $(u_i, v_i)$ that we need to input into Hagerup's algorithm. After that algorithm has run, MST Verification is completed by examining every non-tree edge in $G$, translating it to the equivalent one or two edges in $U'$, querying $B$, and determining whether the non-tree edge is costlier than the tree path maximum.

The extra steps required to find $U$, translate it to $B$, and then find $U'$ all take time linear in the number of edges.

## 5.6 *Bibliographic Notes*

Kruskal's algorithm, presented in Section 5.2 makes use of a data structure known as UNION-FIND or DISJOINT-SET. A near-linear time

implementation was first described by Tarjan[20].

Boruvka's algorithm is quite old[21], and not originally published in English. NeÅąetÅŹil *et al.* published a translation from the original Czech in 2001 along with some comments.[22]

Komlós mentions that his method of using symmetric order heaps for processing paths is something of a well-known method by the time he covers it in his own paper. However, he was unable to find a reference to it in any other literature, which is why he took the time to write about it.

The way that King uses Boruvka's algorithm is first described by Tarjan in 1983[23].

## 5.7 Exercises

**5.1** *Let S=(V,T) be a minimum cost spanning tree, where $|V| = n + 1$. Let $c_1 \leq c_2 \leq ... \leq c_n$ be the costs of the edges in T. Let S′ be an arbitrary spanning tree with edge costs $d_1 \leq d_2 \leq .... \leq d_n$. Show that $c_i \leq d_i$, for $1 \leq i \leq n$.*

**5.2** *Assume all edges in a graph G have distinct cost. Show that the edge with the maximum cost in any cycle in G cannot be in the Minimum Spanning Tree of G. Can you use this to design an algorithm for computing MST of G by deletion of edges, and what will be its complexity?*

**5.3** *Recall that Dijkstra's SSSP algorithm was for directed (or undirected) graphs where the weights of the edges are positive and we need to compute shortest paths from the source vertex to all other vertices in the graph. What happens when some of the edges have negative weights. Try to consider the cases where the algorithm will fail and where the algorithm will still work.*

**5.4** *Design an efficient algorithm to find a spanning tree of a connected, (positive) weighted, undirected graph $G = (V, E)$, such that the weight of the maximum-weight edge in the spanning tree is minimized (Justify your answer).*

**5.5** *Let $G = (V, E)$ be a weighted directed graph, where the weight of each edge is a positive integer and is bounded by a number X. Show how shortest paths from a given source vertex s to all vertices of G can be computed in $O(X|V| + |E|)$ time (Justify your answer).*

**5.6** *Prove that if all edge weights are distinct then the minimum spanning tree of a simple undirected graph is unique.*

**5.7** *Provide a formal proof of Lemma 5.4.1.*

**5.8** *Suppose all edge weights are positive integers in the range $1..|V|$ in a connected graph $G = (V, E)$. Devise an algorithm for computing Minimum*

[20] Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, April 1975

[21] Otakar BorÅŕvka. O jistém problému minimálním. *PrÃące mor. pÅŹÃ■rodovÄŽd. spol. v BrnÄŽ III*, 3:37–58, 1926; and Otakar BorÅŕvka. Příspěvek k řešení otázky ekonomické stavby elektrovodních sítí. *Elektrotechnickỳ obzor*, 15:153–154, 1926

[22] Jaroslav NeÅąetÅŹil, Eva MilkovÃą, and Helena NeÅąetÅŹilovÃą. Otakar borÅŕvka on minimum spanning tree problem translation of both the 1926 papers, comments, history. *Discrete Mathematics*, 233(1âĂŞ3):3 – 36, 2001. <ce:title>Czech and Slovak 2</ce:title>

[23] Robert Tarjan. *Data Structures and Network Algorithms*, volume 44, chapter 6. Minimum Spanning Trees, pages 71–83. SIAM, 1983

*Spanning Tree of G whose running time is better than that of Kruskal's or Prim's algorithm.*

**5.9** *Consider a connected graph $G = (V, E)$ where each edge has a non-zero weight. Furthermore assume that all edge weights are distinct. Show that for each vertex $v \in V$, the edge incident to $v$ with minimum weight belongs to a Minimum Spanning Tree.*

*(Bonus Problem: Can you use this to devise an algorithm for MST - the above step identifies at least $|V|/2$ edges in MST - you can collapse these edges (by identifying the vertices and then recursively apply the same technique - the graph in the next step has at most half of the vertices that you started with - and so on!)*

**5.10** *Prove that the distance values extracted from the priority queue over the entire execution of Dijkstra's single source shortest path algorithm, in a directed connected graph with positive edge weights, is a NON-Decreasing sequence. Where is this fact used in the correctness of the algorithm?*

**5.11** *Can you devise a faster algorithm for computing single source shortest path distances when all edge weights are 1? (Think of an algorithm that runs in $O(|V| + |E|)$ time on a graph $G = (V, E)$.)*

**5.12** *Execute Dijkstra's SSSP algorithm on the following graph on 7 vertices and 18 edges starting at the source vertex s. The edges and their weights are listed in the following (the entry $(xy, 10)$ means the edge directed from the vertex x to the vertex y with edge weight 10):*

$(sb, 5)$, $(sa, 10)$, $(sf, 5)$, $(bf, 6)$, $(ba, 3)$, $(be, 5)$, $(bc, 5)$, $(fs, 2)$, $(fe, 4)$, $(ca, 3)$, $(ce, 2)$, $(cd, 5)$, $(df, 1)$, $(de, 1)$, $(ef, 1)$, $(ec, 1)$, $(af, 1)$, $(ae, 2)$.

**5.13** *Recall that Dijkstra's SSSP algorithm only computes distances from source vertex to all the vertices. What modifications we should make to the algorithm so that it reports the shortest paths as well (in fact the collection of all these paths can be represented in a directed tree rooted at the source vertex).*

**5.14** *Suppose in place of computing shortest path distance from a vertex to every other vertex, we are interested in finding the shortest path distances between every pair of vertices. Then one way to do this is to run Dijkstra's algorithm $|V|$ times, where each vertex in the graph $G = (V, E)$ is considered as a source vertex once. Can you devise an algorithm that is asymptotically faster than just running Dijkstra's algorithm $O(|V|)$ times?*

**5.15** *Which of the following algorithms result in a minimum spanning tree? Justify your answer. Assume that the graph $G = (V, E)$ is connected.*

1. *Sort the edges with respect to decreasing weight.*
   *Set $T := E$.*

*For each edge $e$ taken in the order of decreasing weight do, if $T - \{e\}$ is connected, then discard $e$ from $T$.*
*Set $MST(G) = T$.*

2. *Set $T := \emptyset$.*
*For each edge $e$, taken in arbitrary order do, if $T \cup \{e\}$ has no cycles then $T := T \cup \{e\}$.*
*Set $MST(G) = T$.*

3. *Set $T := \emptyset$.*
*For each edge $e$, taken in arbitrary order do*
**begin**
*$T := T \cup \{e\}$.*
*If $T$ has a cycle $c$ then let $e'$ be a maximum weight edge on $c$.*
*Set $T := T - \{e'\}$.*
**end**
*Set $MST(G) = T$.*

**5.16** *A spanning tree $T$ of a undirected (positively) weighted graph $G$ is called a minimum bottleneck spanning tree (MBST) if the edge with the maximum cost is minimum among all possible spanning trees. Show that a MST is always a MBST. What about the converse?*

**5.17** *Design a linear time algorithm to compute MBST. (Note that an edge with medium weight can be found in linear time. Consider the set of edges whose weight is smaller than the weight of the 'median edge'. What happens if this graph is connected? disconnected?*

**5.18** *Consider an undirected (positively) weighted graph $G = (V, E)$ with a MST $T$ and a shortest path $\pi(s, t)$ between two vertices $s, t \in V$. Will $T$ still be an MST and $\pi(s, t)$ be a shortest path if*
*a) Weight of each edge is multiplied by a fixed constant $c > 0$.*
*b) Weight of each edge is incremented by a fixed constant $c > 0$.*

**5.19** *Let $G = (V, E)$ be a weighted simple connected graph, and assume that all edge weights are distinct. Define the weight of a spanning tree to be the sum total of the weights of edges in that tree. By definition, a minimum spanning tree $T$ of $G$ has the smallest sum total of the weight among all possible spanning trees of $G$. Suppose we are not interested in minimizing the sum total of the weights, but just the weight of the heaviest edge in a spanning tree. Call such a tree a* light spanning tree *(LST). First show that any MST of $G$ is also a LST. Next show that a LST may not always be a MST. To compute LST, we can use an algorithm to compute MST and report that MST as a LST. You are asked to think of an alternate algorithm, running in $O(|V| + |E|)$ time, to find a LST. (Hint: Let $e_m$ be the edge with the median weight among edges in $G = (V, E)$. Consider the subgraph $G'$*

*formed by all edges in $E$, whose weight is at most the weight of $e_m$. Can you deduce something about LST from the connectivity of $G'$.)*

**5.20** *Suppose you are given $n$-points in the plane. We can define a complete graph $G$ on these points, where the weight of an edge $e = (u, v)$, is Euclidean distance between $u$ and $v$. We need to partition these points into $k$ non-empty clusters, for some $n > k > 0$. The property that this clustering should satisfy is that the minimum distance between any two clusters is maximized. (The distance between two clusters $A$ and $B$ is defined to be the minimum among the distances between pair of points, where one point is from cluster $A$ and the other from cluster $B$.) Show that the connected components obtained after running Kruskal's algorithm till it finds all but the last $k - 1$ (most expensive) edges of MST of $G$ produces an optimal clustering.*

# 6
# *Lowest Common Ancestor*

Given a rooted binary tree $T$ on $n$ nodes, we are asked to preprocess it in $O(n)$ time so that the following type of queries can be answered in $O(1)$ time. Given any two nodes $u$ and $v$ of $T$, report their Lowest Common Ancestor $LCA(a, b)$, i.e., among all the common ancestors of nodes $a$ and $b$, find the one which is furthest from the root of $T$. This subproblem arises in many graph applications. Orginal algorithm is due to Harel and Tarjan [1984]. Many years later, Schieber and Vishkin [1993] proposed a new algorithm for the same problem while studying parallel algorithms. Both of these algorithms are fairly complex and are considered to be far from being implementable. Recently, Bender and Farach-Colton [2000] proposed a fairly simple algorithm for the LCA problem, and thats what we present in this chapter.

It is well known that the following Range Minima Problem (RMQ) is related to the LCA problem. Given an array $A[1...n]$ consisting of $n$ numbers, preprocess it so that given any two indices $i$ and $j$, where $1 \leq i \leq j \leq n$, report the minimum element (or its index in $A$) in the subarray $A[i...j]$. Next we will show the reduction of the LCA problem to RMQ problem, and then provide a solution for the RMQ problem.

## *6.1 LCA $\rightarrow$ RMQ*

Let $T$ be the given binary rooted tree. Consider the depth first search traversal of $T$. Observe that the shallowest node encountered in the depth first traversal of $T$ between $u$ and $v$ is the node corresponding to $LCA(u, v)$. (Recall that the main property of dfs traversal is that once it enters a subtree, then it completely visits all the nodes in the subtree - this sort of corresponds to a nice bracketing sequence.) Our aim is to find this node using the RMQs.

Corresponding to the dfs traversal of $T$, let $E$ be the Euler tour. Recall that $E$ stores the nodes of $T$ in the same order as they are vis-

ited during the dfs traversal. The tour $E$ consists of $2n-1$ entries. Let the level of a node in $T$ be its distance from the root. Corresponding to $E$, define a level array $L[1...2n-1]$ which stores the level of the node $E[i]$ in $L[i]$. Furthermore, observe that a node may appear several times in Euler tour. For each node $x \in T$, we maintain an index $R(x)$ that stores the index of the first appearance of $x$ in $E$. Given our notation, the nodes between $E[R(u), ..., R(v)]$ are nodes in Euler tour between the first visits of $u$ and $v$. What is the shallowest node among the nodes in $E[R(u), ..., R(v)]$? For this we will look at the corresponding entries in the level array $L$. More precisely, we need to report what is the minimum element in the subarray $L[R(u)...R(v)]$; this returns us the index of the shallowest node (one with the smallest level) and denote this by $RMQ_L[R(u)...R(v)]$. Hence, $LCA(u, v) = E[RMQ_L[R(u)...R(v)]]$.

**Lemma 6.1.1** *LCA problem on a rooted binary tree $T$ of $n$ nodes can be converted to the range minima query problem on an array $L$ of size $2n-1$ elements. The reduction takes $O(n)$ time. Moreover, LCA queries can be answered within $O(1)$ time in addition to the time required to answer the range minima queries on $L$.*

**Proof.** Notice that the depth first traversal and the construction of Euler tour of $T$ can be done in $O(n)$ time. Within the same time bounds we can maintain the level array as well as keep track of the first appearance of each node in Euler tour. Hence the conversion can be done in linear time. Given the query, $LCA(u, v)$, we need to find the representatives $R(u)$ and $R(v)$ in $E$, then need to answer the query $RMQ_L[R(u)...R(v)]$ followed by one more look up in the array $E$ to report the node corresponding to $LCA(u, v)$. This computation only requires a few pointer manipulation and hence requires $O(1)$ time in addition to answering the range minima query. ∎

## 6.2 Range Minima Queries

Let $A$ be the array of length $n$ consisting of numbers. Our task is to preprocess $A$ so that the range minima queries $RMQ(i, j)$, $1 \leq i \leq j \leq n$, can be answered in $O(1)$ time.

### 6.2.1 A naive $O(n^2)$ algorithm

A simple way to achieve a constant query time is to precompute and store minima for each possible query. In all there are $O(n^2)$ possible queries of type $RMQ(i, j)$, where $1 \leq i \leq j \leq n$, and for each of them we can compute and store the minima in the range $A[i, ..., j]$. It

is easy to see that this computation can be done in $O(n^2)$ time and then given a query it can be answered in $O(1)$ time.

### 6.2.2 An $O(n \log n)$ algorithm

In place of precomputing minima for each possible query, now we precompute minima's for only $O(n \log n)$ selected types of queries. For every $i$ between 1 and $n$ and for every $j$ between 1 and $\log n$, we find minimum element in the subarray $A[i, ..., i + 2^j]$ (we are sloppy with boundary conditions here to keep it simple) and store it in a table in location $M[i, j]$. Next we show that using dynamic programming the table $M$ can be computed in $O(n \log n)$ time. Minima in a subarray of size $2^j$ is computed by looking at the minima of two constituent blocks of size $2^{j-1}$. Either $M[i, j] = M[i, j - 1]$ or $M[i, j] = M[i + 2^{j-1} - 1, j - 1]$.

How do we answer a range minimum query in $O(1)$ time? Let the query be $RMQ(i.j)$, where $1 \leq i \leq j \leq n$. First compute $k = \lfloor \log_2 j - i \rfloor$. Now observe that $2^k$ is the largest interval, that is a power of 2, that fits in the range from $i$ to $j$. Compute $RMQ(i, j)$ be finding out the minimum of two entries in the table, namely $M[i, k]$ and $M[j - 2^k + 1, k]$. Notice that these two table values have been precomputed and hence query can be answered in $O(1)$ time.

**Lemma 6.2.1** *An array $A$ consiting of $n$ numbers can be preprocessed in $O(n \log n)$ time so that the range minima queries can be answered in $O(1)$ time.*

### 6.2.3 An $O(n)$ algorithm with $\pm 1$ property

Consider the following special case of the array $A$ where each element differs from its previous element either by a $+1$ or a $-1$ (this is especially true for the LCA problem as levels of consecutive nodes in Euler tour differs by 1). We will show that in this case $A$ can be preprocessed in $O(n)$ time and RMQs can be answered in $O(1)$ time.

The strategy is pretty simple. First we partition array $A$ into subarrays, where each subarray is of size $\frac{\log n}{2}$ (we are assuming that $n$ is a nice power of 2, otherwise we have to use floors and ceilings and that will not add anything more in terms of understanding.) Within each subarray we find the minimum value and then store all these minimas in an array $A'$. Notice that the size of the array $A'$ is $\frac{2n}{\log n}$ and hence it can be preprocessed in $O(n)$ time by using Lemma 6.2.1.

Consider a range minima query $RMQ(i, j)$ in array $A$, where $i \leq j$. It is answered as follows: Indices $i$ and $j$ may fall within the same subarray, therefore we need to preprocess each subarray

for answering RMQs. If $i$ and $j$ fall in different subarrays then we compute the following three quantities:

1. Minimum value starting at index $i$ up to the end of the subarray containing $i$.

2. Minimum value among the subarrays between the subarray containing $i$ and $j$. This is computed using the preprocessing done for $A'$ in constant time.

3. Minimum value from the beginning up to the index $j$ within the subarry containing $j$.

Now our subproblem is reduced to solving the RMQ problem in subarrays of size $\frac{\log n}{2}$ with $\pm 1$ property. The key observation here is that we do not have too many different kinds of these subarrays.

**Claim 6.2.2** *Given two arrays of same size where each element in the first array is constant value more than the corresponding element in the second array, then the answer to RMQ queries (i.e. the index) is identical in both the arrays.*

Essentially the preprocessing and the RMQ queries work with relative order of elements in these arrays, and they do not need actual values of the elements. Hence for the two subarrays within the above claim, same preprocessing is sufficient to answer RMQ queries. We normalize each of the subarrays by first subtracting the initial value from each of the elements. Next we show that there are only $O(\sqrt{n})$ normal subarrays.

**Claim 6.2.3** *There are at most $O(\sqrt{n})$ normalized subarrays. Each subarrays has length $\frac{\log n}{2}$, where the first element is a 0, and the elements in the array satisfy $\pm 1$ property.*

**Proof.** Each normalized subarray can be specified by a $\pm 1$ vector. Therefore, there are only $2^{\frac{1}{2}\log n - 1} = O(\sqrt{n})$ different types of subarrays of length $\frac{1}{2}\log n$. ∎ We preprocess

each of these subarrays in $O(\log^2 n)$ time to answer RMQ queries in $O(1)$ time using the naive algorithm. The preprocessing requires in all $O(\sqrt{n}\log^2 n)$ time. We summarize the results in the following.

**Lemma 6.2.4** *An array $A$ consisting of $n$-numbers satisfying the $\pm 1$ property can be preprocessed in $O(n)$ time so that the range minima queries can be answered in $O(1)$ time.*

**Corollary 6.2.5** *A binary tree on $n$-nodes can be preprocessed in $O(n)$ time so that the lowest common ancestor queries can be answered in $O(1)$ time.*

## 6.3 $RMQ \rightarrow LCA$

Next we show that an instance of the RMQ problem can be converted to an instance of the LCA problem. For a linear array $A$ of size $n$, the tree $T$ for the LCA problem consists of $n$ nodes and given a RMQ query, we perform an equivalent LCA query on $T$, and whose answer in turn provides the answer for the original range minima query. This will imply that the general RMQ problem (i.e., even without the $\pm 1$ property) can be answered in $O(1)$ time by performing an $O(n)$ time preprocessing. The key to this conversion is the concept of Cartesian Tree.

Let $A[1...n]$ be the input array on which we need to perform RMQ queries. Cartesian tree $T$ for $A$ is defined as follows. It is a rooted binary tree and the root of $T$ stores the index of the smallest element in $A$. Deleting the minimum element from $A$ splits it into two subarrays. Left and right children of the root are recursively defined Cartesion trees for left and right subarrays of $A$, respectively.

**Claim 6.3.1** *Cartesian tree $T$ for an array $A$ of n-numbers can be constructed in $O(n)$ time.*

**Proof.** We scan the array $A$ from left to right and incrementally build the Cartesian tree $T = T_n$ as follows. Suppose so far we have built the tree $T_i$ with respect to elements $A[1..i]$ and we want to extend it for $A[1...i+1]$ to obtain $T_{i+1}$, where $i < n$. Main observation is that the node storing the index $i+1$ in $T_{i+1}$ is on the rightmost path of $T_{i+1}$. We start at the rightmost node of $T_i$ and follow the parent pointers till we find the location to insert $i+1$ in Cartesian tree. Note that each comparison will either add a node or removes one from the rightmost path. Since each node can only join the rightmost path once (if it leaves it then it can't be back to the rightmost path), therefore the total time in constructing $T$ is $O(n)$. ∎

**Claim 6.3.2** *Let $A$ be the array on n-numbers and $T$ be the corresponding Cartesian tree storing the indices of elements in $A$ in its node. Then $RMQ(i,j) = LCA(i,j)$.*

**Proof.** This follows from the recursive definition of Cartesian tree $T$. Let $k = LCA(i,j)$ in $T$. Observe that the node labeled $k$ is the first node that separates $i$ with $j$. In other words, the element $A[k]$ is the smallest element between $A[i]$ and $A[j]$, i.e. $RMQ[i,j] = k$. ∎

## 6.4 Summary

In this chapter, we have shown that the lowest common ancestor query in a rooted binary tree on $n$-nodes can be answered by solving the range minima query in an array consisting of $2n - 1$ numbers satisfying the $\pm 1$ property. Moreover, the general RMQ problem in an array can be reduced to solving LCA queries on the corresponding Cartesian tree. All our preprocessing algorithms require linear time and the queries can be answered in constant time.

## 6.5 Exercises

**6.1** *Prove that in the LCA algorithm of Bender and Farach-Colton, why does the reduction from the LCA problem to the range-minima query works, i.e., show that in place of finding the LCA of nodes u and v in the binary tree, why does it suffice to compute the smallest level number in the level array in an interval defined by the first occurrence of the node u an v in the level array.*

**6.2** *This problem is to show that an arbitrary range minima query (RMQ) problem can be solved within the same complexity as the one with the $\pm 1$ RMQ problem. Recall that the $\pm 1$ RMQ problem for an array of size n required $O(n)$ time to preprocess and then the queries were answered in $O(1)$ time. The idea is to reduce an arbitrary RMQ problem to the LCA problem. This reduction uses Cartesian Tree. Let A be an array consiting of n numbers (need not satisfy the $\pm 1$ property). The Cartesian Tree C for A is defined as follows: The root of C is the minimum element of A, and it stores the position of this element in the array. Removing the root element splits the array into left and right subarrays. The left and right children of the root are recursively constructed Cartesian trees of the left and right subarrays, respectively. Prove the following:*

1. *Cartesian tree C of an array A of size n can be computed in $O(n)$ time (use incremental construction).*
2. *Show that $RMQ_A(i, j) = LCA_C(i, j)$ (Recall that in C we store the indices i and j.)*

# 7
# Network Flow

## 7.1 What is a Flow Network

A *flow network* consists of the following:

1. A simple finite directed graph $G = (V, E)$.
2. Two specified vertices, namely source $s$ and target $t$.
3. For each edge $e \in E$, a non-negative number $c(e)$ called the capacity. If a pair of vertices $u$ and $v$ are not joined by an edge, then $c(u, v) = 0$.

**Flow:** A flow function $f$ in $G$ is a real-valued function

$$f : V \times V \to \Re$$

that satisfies the following three properties.

1. **Capacity Constraint**: For all $u, v \in V$, $f(u, v) \leq c(u, v)$.
2. **Skew Symmetry** (A tough constraint to see!): For all $u, v \in V$, $f(u, v) = -f(v, u)$. This is for notational purposes, and basically says that flow from a vertex $u$ to vertex $v$ is the negative of the flow in the reverse direction.
3. **Flow conservation**: For all $u \in V - \{s, t\}$, $\sum_{v \in V} f(u, v) = 0$. This uses the skew symmetry property, otherwise we have to sum up the flow values coming into a vertex and that should be equal to the sum of the outgoing flow values from that vertex. This is same as the Kirchoff law for current in an electrical circuit, i.e. no node can hold the current, or no node can hold the flow, or whatever comes in goes out. There is no reservoir at a node.

The value of the flow is defined to be the flow out of the source $s$ or the flow into the target $t$, i.e.

$$|f| = \sum_{v \in V} f(s, v) = \sum_{v \in V} f(v, t).$$

The **Maximum Flow Problem** to find the flow of maximum value in a given flow network.

See Figure 7.1 for an example of a network flow.

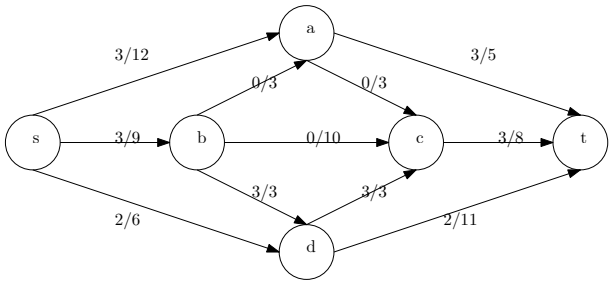



Figure 7.1: An example of a network flow with a flow of 8 from $s$. Each edge shows the amount of flow on that edge (numerator term) and the total capacity (denominator term).

## 7.2 *Ford and Fulkerson's Algorithm*

This is an iterative method for computing the flow.

Ford-Fulkerson-Method $(G, s, t)$

1. Initialize the flow $f$ to 0.
2. While there exists an augmenting path $p$, augment the flow $f$ along $p$.
3. Return $f$.

An augmenting path is a path from $s$ to $t$ along which additional flow can be sent. This path is found using the concept of residual networks. The residual network consists of those edges which can admit more flow. The residual capacity $c_f(u, v)$ of an edge $(u, v)$ in a flow network is given by

$$c_f(u, v) = c(u, v) - f(u, v).$$

In our example $c_f(s, a) = 12 - 3 = 9$, $c_f(a, s) = 0 - (-3) = 3$, $c_f(b, a) = c(b, a) - f(b, a) = 3 - 0 = 3$. Given a flow network $G$ and the flow function $f$, the residual network $G_f = (V, E_f)$ consists of the same vertex set and the edges $E_f$ are defined as follows:

$$E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}.$$

The residual network of our example is given in Figure 7.2.

As we can see that there is an augmenting path in this network (the red path), and the flow can be augmented along this path, by a value of 3. Hence we get a new flow network with the total flow value equals to $8 + 3 = 11$ given in Figure 7.3. Note that edge $dc$ has

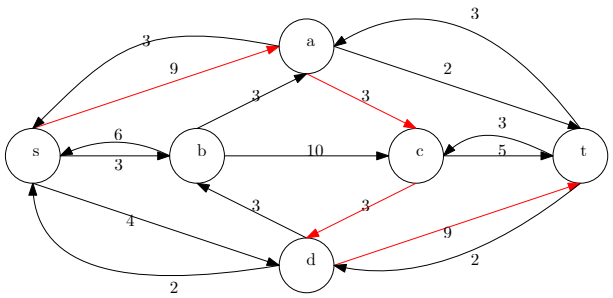



Figure 7.2: Residual network corresponding to the flow in Figure 7.1. The red-path from $s$ to $t$ is an augmenting path, with a residual capacity of 3.

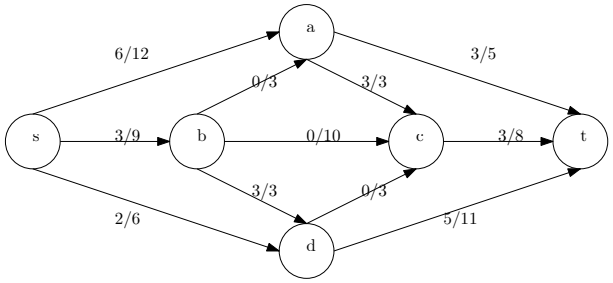



Figure 7.3: Flow network after augmenting the flow from Figures 7.1 and 7.2.

a flow of 0 after the augmentation, whereas it had a flow of 3 units before the augmentation.

The new residual network that we obtain for the flow corresponding to the flow network in Figure 7.3 is given in Figure 7.4. Note that there exists an augmenting path that can further increases the flow value by 4.

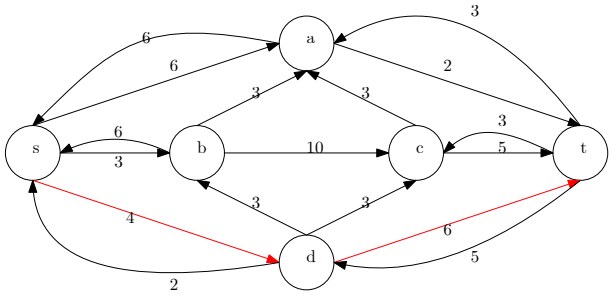



Figure 7.4: Residual network corresponding to the flow in Figure 7.3.

The new flow graph is shown in Figure 7.5 and the corresponding residual network is shown in Figure 7.6.

This will continue for a few more iterations and after that there is no path between $s$ and $t$ in the residual network. The corresponding figures are Figure 7.7 and Figure 7.8.

Now consider the residual network in Figure 7.8, where there are no paths joining $s$ and $t$. As can be seen from the figure, there is a path from $s$ to every vertex in the set $\{s, a, b, c\}$, and there are

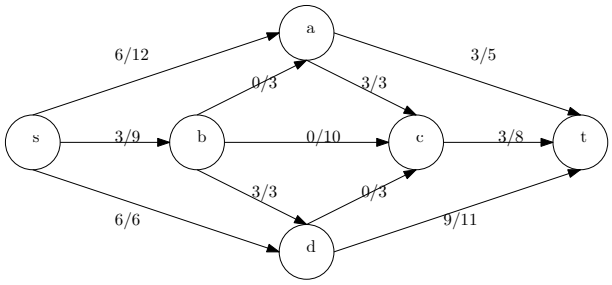



Figure 7.5: Flow network after augmenting the flow from Figures 7.3 and 7.4.

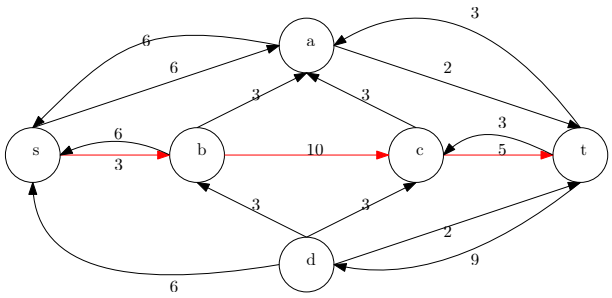



Figure 7.6: Residual network corresponding to the flow in Figure 7.5.

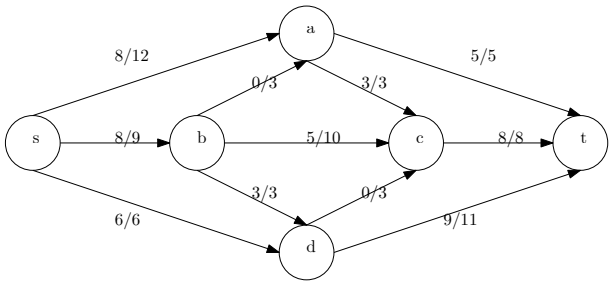



Figure 7.7: Resulting Flow network.

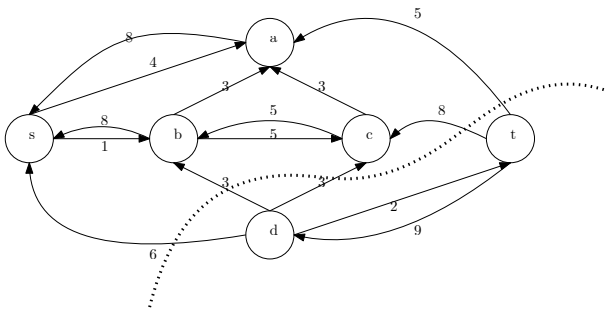



Figure 7.8: Residual network corresponding to the flow in Figure 7.7.

paths from vertices $\{d, t\}$ to $t$. This automatically partitions the set of vertices into two, call it a $s - t$ cut $\{S, T\}$, where $s \in S$ and $t \in T$ (in our example, $S = \{s, a, b, c\}$ and $T = \{d, t\}$). See Figure 7.9. Define the capacity of a cut as follows

$$c(S, T) = \sum_{u \in S, v \in T} c(u, v).$$

In other words consider the edges crossing the cut, and sum up the capacities of the edges which go from a vertex in the set $S$ to a vertex in the set $T$. Define the net flow across the cut to be

$$f(S, T) = \sum_{u \in S, v \in T} f(u, v).$$

In other words the net flow is the sum of the positive flow on edges going from $S$ to $T$ minus the sum of the positive flows on edges going from $T$ to $S$ (recall the skew symmetry property). Amazingly in our example $f(S, T) = c(S, T)$. Is it always true or just a luck! Before we get to this, a few observations.

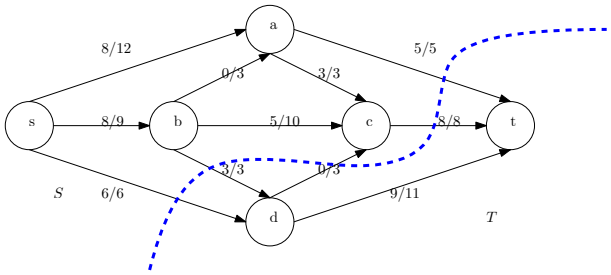



Figure 7.9: An illustration of a $s - t$ cut. Note that $f(S, T) = \sum_{u \in S, v \in T} f(u, v) = f(s, d) + f(b, d) - f(d, c) + f(c, t) + f(a, t) = 22$ and $c(S, T) = \sum_{u \in S, v \in T} c(u, v) = c(s, d) + c(b, d) + c(c, t) + c(a, t) = 22$.

**Observation 7.2.1** *For any $s - t$ cut $S$, $T$, and flow $f$*

$$|f| \leq c(S, T).$$

This follows from the definition of the flow across the cut. The flow $f(S, T)$ is defined to be the sum of the positive flows along the edges in the forward direction, i.e. the ones going from vertices in $S$ to vertices in $T$ minus the sum of the positive flows along the edges in the reverse direction. If we ignore the reverse direction, then clearly the flow along each edge in the forward direction is bounded by the capacity of the edge. Sum of these capacities is the capacity of the cut and hence the observation.

The following observation explains why the flow $f'$ found using the augmenting paths in the residual graph $G_f$, can be augmented with the flow $f$ in $G$, to obtain a new flow in $G$ of a higher value $|f + f'| \geq |f|$.

**Observation 7.2.2** *Let $G$ be the flow network with flow $f$ and $G_f$ be the corresponding residual network and let $f'$ be the flow in $G_f$. Then the flow sum $f + f'$ is a flow in $G$ and its value is $|f + f'| = |f| + |f'|$.*

**Proof.** To prove that $f + f'$ is a flow in $G$, we need to prove that the three conditions are satisfied. We show the capacity constraint, and others are left as an exercise. The capacity constraint follows from

$$(f + f')(u, v) = f(u, v) + f'(u, v) \leq f(u, v) + (c(u, v) - f(u, v)) = c(u, v).$$

Observe that

$$|f + f'| = \sum_{v \in V} (f + f')(s, v) = \sum_{v \in V} f(s, v) + \sum_{v \in V} f'(s, v) = |f| + |f'|.$$

∎

**Theorem 7.2.3** *Let $f$ be a valid flow in the flow network $G = (V, E)$ from the source $s$ to the target $t$, then the following statements are equivalent.*

1. *Flow $f$ is a maximum flow.*
2. *Residual network $G_f$ does not contain an augmenting path.*
3. *There exists some cut $c(S, T)$ such that $|f| = c(S, T)$.*

*This is the famous* max-flow min-cut theorem.

**Proof.** Recall that to prove that the three statements are equivalent we need to show that $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 1$.
$1 \Rightarrow 2$: Let $f$ be a maximum flow and, for contradiction, assume that there exists an augmenting path in $G_f$. Then we can increase the flow along the path using Observation 7.2.2 and contradicting that $f$ is a maximum flow.
$2 \Rightarrow 3$: Define the set

$$S = \{v \in V | \text{there is a path from } s \text{ to } v \text{ in } G_f\}$$

and

$$T = V \setminus S.$$

Also observe that $s \in S$ and $t \in T$, so it is a valid $s - t$ cut. Moreover for all edges $(u, v)$ crossing the cut, where $u \in S$ and $v \in T$, $f(u, v) = c(u, v)$, otherwise $(u, v) \in E_f$ and $v \in S$, which is not possible. Net flow across the cut $(S, T)$ is $|f|$! Why? (Think about this yourself!) So we have shown a cut where 3 holds.
$3 \Rightarrow 1$: We know that the capacity of any cut is an upper bound to the value of the flow. If for a cut we obtain the equality, then we have attained the max-flow. (In other words the capacity of minimum cut is the value of the maximum flow!). ∎

This proves the correctness of the Ford-Fulkerson algorithm. The algorithm iteratively increases the value of the flow using augmenting paths and returns the value of the flow, the maximum flow, when it is not able to find an augmenting path in the residual graph. How do we analyze the complexity of this algorithm?

First a special case where all capacities are integers. Observe that value of all flows computed during the algorithm are integers. In each iteration of the algorithm, the value of flow increases by at least 1. If $f^*$ is a maximum flow, then the number of iterations in the algorithm are bounded by $|f^*|$. It is easy to see that each iteration requires $O(|E|)$ time; this involves computing residual graph (i.e. capacities on at most $2|E|$ edges), and computing a path between $s$ and $t$ (directed dfs or bfs). Hence the algorithm runs in $O(|f^*||E|)$ time - this is a strange complexity since the running time depends upon the value of the output! Is there a better way to analyze this algorithm!

## 7.3 *Edmonds-Karp Algorithm*

In this algorithm, in the Ford-Fulkerson method, a particular path is chosen to be an augmenting path in the residual graph. A BFS tree rooted at $s$ is computed in the residual graph and an unweighted shortest path from $s$ to $t$ is chosen to be an augmenting path. It turns out that this variation leads to an algorithm that runs in $O(|V||E|^2)$ time. Here is the main lemma - let $\delta_f(s, v)$ denote the shortest path distance between $s$ and $v$ in the unweighted residual graph $G_f$, corresponding to the flow network $G$ with flow function $f$.

**Lemma 7.3.1** *Shortest path distance for all vertices $v \in V - \{s, t\}$ in $G_f$ increases monotonically with each flow augmentation.*

**Proof.**

Caution: This is a little bit strange proof, and the proof in generic terms goes as follows. To prove the statement $P$, the contradictory proof assumes that $\neg P$ is true. Inside the proof we need to prove a claim $C$, which in turn is proved using the contradiction. Say $\neg C$ is true. The contradiction is arrived by showing that $\neg C$ is true only if $P$ is true, but since $\neg P$ is assumed to be true, implying that $C$ is true. Once we show that $C$ is true, the contradiction to the original assumption is arrived at.

Assume that for a vertex $v \in V - \{s, t\}$, the shortest path decreases after a flow augmentation. Let $f$ be the flow before the augmentation and $f'$ be the flow after the augmentation. Let $v$ be the vertex with minimum $\delta_{f'}(s, v)$ whose distance was decreased by the augmentation (i.e. $\delta_{f'}(s, v) < \delta_f(s, v)$). Let $u$ be the vertex just

before $v$ in the shortest path from $s$ to $v$ in $G_{f'}$, i.e. $(u,v) \in E_{f'}$. Then $\delta_{f'}(s,u) = \delta_{f'}(s,v) - 1$. Moreover, $\delta_{f'}(s,u) \geq \delta_f(s,u)$ (by choice of $v$).

Now we will show that $(u,v) \notin E_f$, and as a consequence of that we will arrive to contradiction (somehow!).

First why $(u,v) \notin E_f$? Suppose $(u,v) \in E_f$, then $\delta_f(s,v) \leq \delta_f(s,u) + 1$ (triangle inequality - sum of two sides of the triangle is at least as big as the third side). But, $\delta_f(s,u) + 1 \leq \delta_{f'}(s,u) + 1 = \delta_{f'}(s,v)$. This implies that $\delta_f(s,v) \leq \delta_{f'}(s,v)$, contradicts our assumption!

Now consider the scenario that $(u,v) \notin E_f$ and $(u,v) \in E_{f'}$. The flow from $v$ to $u$ must have been increased in Edmonds - Karp algorithm and this edge must be on a shortest path. This implies that $\delta_f(s,v) = \delta_f(s,u) - 1 \leq \delta_{f'}(s,u) - 1 = \delta_{f'}(s,v) - 2$, and this contradicts the assumption that $\delta_{f'}(s,v) < \delta_f(s,v)$. ∎

In each iteration of the augmenting path algorithm, at least one edge becomes critical, i.e. flow value becomes equal to its capacity. The critical edge disappears from the residual network. Of course the flow along this edge may be decreased in the future, and this edge may reappear again in the residual network, but this cannot happen more than $|V|/2$ times. Why?

Say $(u,v)$ became critical, then $\delta_f(s,v) = \delta_f(s,u) + 1$. Flow along $(u,v)$ is decreased only if $(v,u)$ appears on an augmenting path, let $f'$ be the flow and note that $\delta_{f'}(u) = \delta_{f'}(v) + 1$. Since the shortest path distances are monotone, this implies that

$$\delta_{f'}(s,u) = \delta_{f'}(v) + 1 \geq \delta_f(s,v) + 1 = \delta_f(s,u) + 2.$$

Therefore the distance to $u$ from the source has increased by at least 2 between two consecutive times that $(u,v)$ became critical. The maximum distance is at most $|V|$ and hence an edge can become critical at most $|V|/2$ times. There are $O(|E|)$ edges in all in the residual graph, and hence the number of augmentations (or iterations) are bounded by $O(|V||E|)$ times. Each augmentation can be implemented in $O(|E|)$ time, and hence flow between $s$ and $t$ in the graph $G = (V,E)$ can be computed in $O(|V||E|^2)$ time.

## 7.4 *Applications of Network Flow*

We can use the flow networks to compute Maximum Matching in a Bipartite Graphs. Recall that a Graph $G = (V = A \cup B, E)$ is bipartite, if the set of vertex $V$ is partitioned into two sets $A$ and $B$, such that all the edges in the graph are between vertices of $A$ to vertices in $B$. A matching in a graph is a collection of edges such that no two edges in the matching are incident to the same vertex. A matching in $G$ is

called a *maximum matching* if the cardinality of the number of edges in it is maximum among all matchings in $G$. Note that there can be a number of maximum matching in a graph. Using flow networks we can compute easily maximum matching in $G$. Here is the simple method. We add two vertices, namely $s$ and $t$, to the set of vertices in $G$. Vertex $s$ is connected to all the vertices in the set $A$ by directed edges from $s$. The capacity of all these edges is set to 1. The capacity of all the edges in the set $E$, i.e., the edges joining vertices in the set $A$ to vertices in the set $B$, is set to 1 and they are directed from vertices in $A$ to vertices in $B$. Lastly vertices in $B$ are joined to $t$ by directed edges with capacity 1. Let $G'$ be the resulting flow network. Compute the maximum $s - t$ flow in $G'$. Observe that the value of the flow is the size of the maximum matching. Why ?

Note that value of flow in each of the edge will be an integral value, since all the capacities are integers (this is one of the exercises in [1]). Since the capacity of all the edges between vertices in $A$ and $B$ is 1, the value of the flow on these edges is either 0 or 1. This implies that no two edges in $E$ are incident on the same vertex will ever have nonzero flow. In other words the edges in $E$ which have nonzero flow are the edges in a matching. Also maximum matching corresponds to a largest set of independent edges in $G$ and each of these edges can admit a flow of value 1 and at the same time satisfy all the three conditions required for a flow network. Hence maximum matching in $G$ corresponds to a valid flow in $G'$.

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009

## 7.5 Exercises

**7.1** *Construct a network flow example with 7 vertices and 11 directed edges, where each edge has a positive capacity and compute the maximum flow and minimum cut in this graph. You should show some of the steps in the algorithm. (Follow Edmonds-Karp shortest path heuristic).*

**7.2** *Assume that we have a network flow graph $G = (V, E)$ with positive capacities on each of the edges and two specified vertices $s$ and $t$. Suggest an efficient algorithm to find an edge in $E$, such that setting its capacity to zero (i.e. deleting this edge) will result in the largest decrease in the maximum flow in the resulting graph.*

**7.3** *Suppose we are given a flow network $G$, where edges have positive integer capacities, and $\mathcal{C} = \sum_{u,v \in V} c(u, v)$, where $c(u, v)$ is the capacity of the edge $e = (u, v) \in E$. Show the following*

1. *The value of the max flow is an integer.*
2. *There is an assignment of non-negative integer flow values on each edge of $G$, satisfying all the flow conservation conditions, so that $G$ achieves*

*max flow.*

3. *Show that the number of iterations required in the Ford-Fulkerson's algorithm (Residual network, find an augmenting path, augment the flow, repeat) is $O(C)$.*

4. *Show that in the worst case, Ford-Fulkerson's algorithm, as stated in Part 3 runs in exponential time.*

5. *Construct an example, where one can realize the worst case as stated in Part 4.*

**7.4** *Let $G = (V, E)$ be the flow network. Let $C = \max_{(u,v)\in E} c(u, v)$ be the maximum capacity. Show the following:*

1. *Minimum cut of $G$ has a capacity of at most $C|E|$.*

2. *For a given number $K > 0$, show how to find an augmenting path of capacity at least $K$ in $O(|E|)$ time, provided that such a path exists.*

3. *Execute the following algorithm:*

   (a) *Initialize the flow $f = 0$;*

   (b) *$K = 2^{\lfloor \log C \rfloor}$;*

   (c) *While $K \geq 1$ do*

      i. *While there exists an augmenting path $p$ of capacity at least $K$ then augment flow $f$ along $p$.*

      ii. *$K := K/2$;*

   (d) *Return $f$.*

   *Show that the above algorithm computes Max-Flow.*

4. *Show that the loop in Step 3c(i) is executed at most $O(|E|)$ times for each value of $K$.*

5. *Show that the algorithm runs in $O(|E|^2 \log C)$ time.*

**7.5** *Let $G = (V, E)$ be a flow network. Recall that $G$ is a complete graph, where some of the edges may have a capacity of zero. Suppose your task in the max flow problem is to increase the flow of a network as much as possible, but you are only allowed to increase the capacity of only one edge, whose capacity is strictly larger than zero. First show that there are networks where such an edge may not exist, i.e. increasing the capacity of a single edge ($> 0$ capacity) will not alter the value of the max-flow. Show that there are networks, where such an edge may exist. Try to design an algorithm which can detect whether flow can be increased.*

**7.6** *A simple undirected graph $G = (V, E)$ is called k-edge connected if removal of any set of k-edges keeps G still connected. (e.g. cycles are 1-edge connected.) Show how to compute edge connectivity of G by invoking at most $|V|$ network flow computations.*

# 8
# *Separators in a Planar Graph*

This chapter is based on Kozen [1] and the famous paper of Lipton and Tarjan [2] on the planar separator theorem. Earlier we have seen that for a binary tree on $n$-nodes, there exists a node such that whose removal leaves no component having more than $2(n+1)/3$ nodes. This can be extended to outerplanar graphs, where we can remove a pair of vertices such that none of the components have more that $2(n+1)/3$ nodes. Usually this phenomenon is referred to as a balanced decomposition using small size separators. This is a 'key idea' in most of the divide and conquer type algorithms on these graphs. As can be seen that the depth of recursion will be $O(\log n)$ and since the size of the separator is small, the "merge" step will be economical as well. First we start with some preliminaries and then we will prove that in a planar graph there exists a separator of size $O(\sqrt{n})$.

[1] D. Kozen. *The design and analysis of algorithms*. Springer, 1992

[2] Richard J. Lipton and Robert E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979

## *8.1 Preliminaries*

**Definition 8.1.1** *A graph is called* planar *if the vertices and edges can be laid out (embedded) in the plane so that no two edges intersect except at their end points. An embedded planar graph is usually referred to as a* plane graph.

**Definition 8.1.2** *In an embedded plane graph, we have vertices, edges and faces. The* dual *of a plane graph $G$ is a planar graph $G^*$ whose vertices correspond to faces of $G$ and two vertices in $G^*$ are joined together if the corresponding faces in $G$ share an edge.*

**Definition 8.1.3** *A plane graph $G$ is* triangulated *if each of its face is a triangle, i.e., it is bounded be three edges. In other words, in the dual each vertex has degree three.*

**Definition 8.1.4** *A set $S \subseteq V$ for a graph $G = (V, E)$ is called a* vertex separator, *if removal of vertices (and incident edges on these vertices) from $G$ results in two disjoint sets of vertices $A, B \subseteq V$ with no edges between*

*them. If the sizes of the sets $A$ and $B$ are a constant fraction of that of the size of $V$, then $S$ is called as a* balanced separator.

**Definition 8.1.5** *A planar graph $G = (V, E)$ consists of at most $|E| = 3|V| - 6$ edges. This follows from Euler's relation, i.e. $|V| - |E| + |F| = 2$. You may like to check the proof at*

`http:\\www.ics.uci.edu/~eppstein/junkyard/euler/`

**Definition 8.1.6** *An* outerplanar graph *is a plane graph such that all its vertices lie on a single face. This face is usually referred to as the* outerface.

**Definition 8.1.7** *The dual of a triangulated outerplanar graph is a binary tree.*

We have seen that a complete graph on five vertices, $K_5$, and a complete bipartite graph on six vertices, $K_{3,3}$, are nonplanar. It is easy to see that a tree is planar, and outerplanar graphs are planar. Both of these graphs admit small size separators. What we will prove in this chapter is that all planar graphs satisfy a similar property.

**Theorem 8.1.8** *[Lipton and Tarjan [3]] Let $G = (V, E)$ be an embedded undirected triangulated planar graph, where $n = |V|$. There exists a partition of $V$ into disjoint sets $A$, $B$, and $S$, such that*

1. $|A|, |B| \leq \frac{2n}{3}$
2. $|S| \leq 4\sqrt{n}$
3. *There is no edge in $E$ that joins a vertex in $A$ with a vertex in $B$.*
4. *Such a set $S$ can be found in linear time.*

[3] Richard J. Lipton and Robert E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979

It will turn out that the way we prove this theorem, it will lead to a linear time algorithm (i.e. $O(|V| + |E|)$) for finding such a separator. Note that if the given graph is not embedded in the plane, then there is a linear time algorithm by Hopcroft and Tarjan that embeds it. In fact that algorithm also figures out in linear time whether the given graph is planar or not, and if it is planar it finds an embedding. Also if a plane graph is not triangulated, then it can be triangulated in linear time, by inserting required number of edges on each face. Other than this essentially we will use breadth first and the concept of fundamental cycles to prove this theorem.

## *8.2 Proof of the Planar Separator Theorem*

Assume that the graph $G = (V, E)$ is undirected, connected, planar, triangulated and embedded. The first step in the proof/algorithm is

to do a breadth-first search starting at an arbitrary vertex, say $s$, in $G$, and assign levels to vertices. Vertex $s$ is at level 0, vertices adjacent to $s$ are at level 1, vertices adjacent to level 1 vertices that have not been assigned any level are level 2 vertices, and so on. Let $l$ be the last level, and pretend that there is a level $l+1$ which consists of no vertex (this is just required for the proof!). Let $L(t)$ denote the set of vertices that are in level $t$, $0 \leq t \leq l$. Recall that in BFS, no edge can span over two or more levels. All edges must connect vertices in the same level or consecutive levels. Observe that each of the level, $L(t)$, for $0 < t < l$, is a separator in its own right, although may not be of small size and may not lead to a balanced decomposition!

Number the vertices according to BFS ordering, where $s$ gets number 1, followed by vertices in level 1, then vertices in level 2, and so on (see Figure 8.1). Let $t_1$ be the middle level, that is the one which contains the vertex number $n/2$ in the BFS numbering. Consider the set $L(t_1)$. Note that $|\cup_{t<t_1} L(t)| < n/2$ and $|\cup_{t\leq t_1} L(t)| \geq n/2$. If $|L(t_1)| \leq 4\sqrt{n}$, then $S = L(t_1)$ and we are done. Note that in that case we can set the set $A$ to be all the vertices in levels 0 up to the level $t_1 - 1$. Similarly the set $B$ can be defined as all the vertices in levels $t_1 + 1$ to $l$. Clearly $|A| < n/2$ and $|B| < n/2$. In general, it is not necessary that $L(t_1)$ may satisfy the requirements on the size of the separator. Here is the lemma which will be very handy in that case.

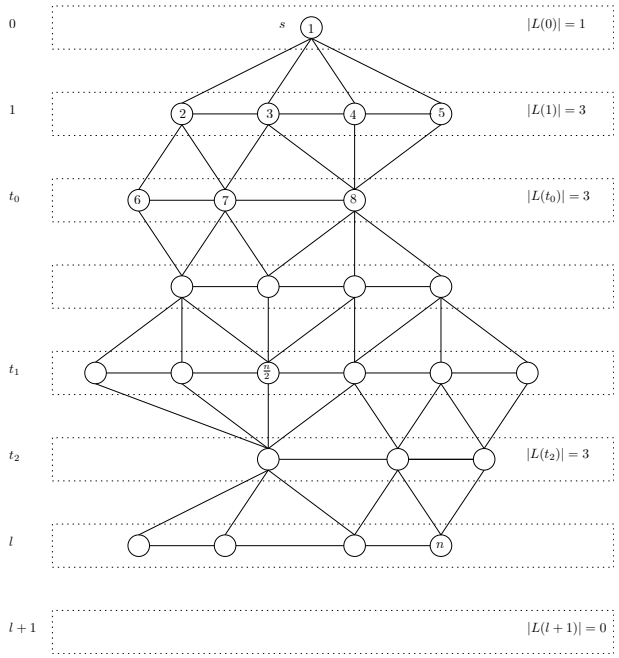



Figure 8.1: BFS and the sets $L(\cdot)$.

**Lemma 8.2.1** *There exists levels $t_0 \leq t_1$ and $t_2 > t_1$ such that, $t_2 - t_0 \leq \sqrt{n}$, $|L(t_0)| \leq \sqrt{n}$ and $|L(t_2)| \leq \sqrt{n}$.*

**Proof.** Note that $|L(0)| = 1$ and $|L(l+1)| = 0$. Let $t_0 \leq t_1$ be the largest number such that $|L(t_0)| \leq \sqrt{n}$. Let $t_2 > t_1$ be the smallest number such that $|L(t_2)| \leq \sqrt{n}$. Note that every level between $t_0$ and $t_2$ contains more than $\sqrt{n}$ vertices, therefore by pigeon hole principle there must be fewer than $\sqrt{n}$ levels between $t_0$ and $t_2$, otherwise $G$ will have more than $n$ vertices! Therefore, $t_2 - t_0 \leq \sqrt{n}$. ∎

Define three sets $C, D$ and $E$ as follows: $C = \cup_{t<t_0} L(t)$, $D = \cup_{t_0<t<t_2} L(t)$ and $E = \cup_{t>t_2} L(t)$. If $|D| \leq 2/3n$, then we have the required separator, by setting $S = L(t_0) \cup L(t_2)$, $A$ the largest of $C$, $D$ or $E$ and $B$ the union of the other two.

What if $|D| > 2/3n$? Then both the sets $C$ and $E$ are small, have less than $1/3n$ vertices. We will find a $\frac{1}{3} - \frac{2}{3}$ separator $S_D$, of $D$, of size at most $2\sqrt{n}$. Let $D$ be split into $D'$ and $D''$ by $S_D$. Then $S$ will include the vertices in $L(t_0), L(t_2)$, and the separator vertices $S_D$. Set $A = \max(C, E) \cup \min(D', D'')$ and $B = \min(C, E) \cup \max(D', D'')$. Observe that $S, A$, and $B$ satisfy the required size criteria.

Next we will present some ideas regarding finding the separator $S_D$ of $D$. First we remove all the vertices that are not in $D$, except the start vertex $s$. We connect $s$ to all the vertices in level $t_0 + 1$. This can be done still preserving the planarity of $D$, since the original graph is planar. Now we construct a spanning tree $T$ in $D$, such that its diameter is at most $2\sqrt{n}$. Start with vertices in level $L(t_2 - 1)$. For each vertex in this level, choose one of the vertex in the previous level $L(t_2 - 2)$, adjacent to it as its parent. Continue this process with vertices in levels $t_2 - 2$, $t_2 - 3$, $\cdots$, to obtain the tree $T$. Next we state two lemmas, that are relatively easy to prove, that will show the critical property relating the tree $T$, the plane graph $D$, its dual $D^*$, and the dual tree $T'$.

**Lemma 8.2.2** *Let $G = (V, E)$ be a connected plane graph and $G^*$ be its dual. For any $E' \subseteq E$, the subgraph $(V, E')$ has a cycle if and only if the subgraph $(V^*, E - E')$ of $G^*$ is disconnected.*

**Lemma 8.2.3** *Let $G = (V, E)$ be a connected plane graph with dual $G^* = (V^*, E)$ and let $E' \subseteq E$. Then $(V, E')$ is a spanning tree of $G$ if and only if $(V^*, E - E')$ is a spanning tree of $G^*$ (see Figure 8.2 for an illustration).*

Let $E_T$ be the edges of the spanning tree $T$, constructed by following the parents in $D$ as stated above. Recall that the diameter of $T$ is at most $2\sqrt{n}$. Also $D$ is triangulated. Consider the dual $D^*$ of $D$, and consider the edges in $E - E_T$. They define a spanning tree $T'$ in $D^*$ (by Lemma 8.2.3). Also we can orient each edge in $T'$ away from the root. Pick a face of $D$ (say its outer face) and choose this as the root

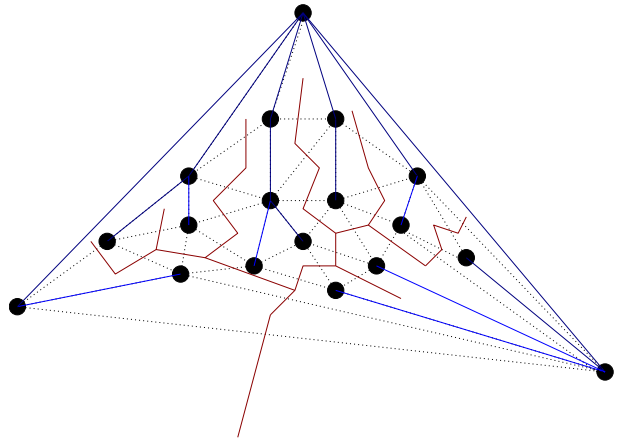


Figure 8.2: The edges of $(V, E')$ are in blue. The edges of $(V^*, E - E')$ are in red.

$T'$. It will turn out that the required separator $S_D$ will be defined by an edge, $e = (u, v)$ in $e \in E - E_T$, and the unique path in the tree $T$ between $u$ and $v$. In other words, $e$ defines a unique cycle, $c(e)$, in $T$. The cycle $c(e)$ is referred to as a *fundamental cycle* in literature. To compute/define $c(e)$ appropriately we first perform a DFS of $T'$ and compute the following three quantities.

1. $I(e)$= number of vertices which are in the interior of the cycle $c(e)$.
2. $|c(e)|$= number of vertices on the cycle $c(e)$.
3. Linked list representation of $c(e)$.

For each step of DFS, one of the following four cases will occur (see Figure 8.3)

Case 1: DFS visits a leaf of $T'$ (i.e. a triangular face of $D$). Then $I(e) = 0$, $|c(e)| = 3$, and $c(e) = \{x, u, v\}$.

Case 2: DFS visits a triangle corresponding to an edge $e = (u, v) \in E - E_T$, its degree is two and the other edge of the triangle is $e' = (u', v) \in E - E_T$ which was visited in the previous step. Moreover $u' \in c(e)$. Then $I(e) = I(e')$, $|c(e)| = |c(e')| + 1$, and $c(e) = uc(e')$.

Case 3: DFS visits a triangle corresponding to an edge $e = (u, v) \in E - E_T$, its degree is two and the other edge of the triangle is $e' = (u', v) \in E - E_T$ which was visited in the previous step. Moreover $u' \notin c(e)$. Then $I(e) = I(e') + 1$, $|c(e)| = |c(e')| - 1$, and $c(e)$ equals to $c(e')$ except that $u'$ is removed from the front of the list.

Case 4: DFS visits a triangle corresponding to edge $e = (u, v) \in E - E_T$ and its degree is three. The other two edges $e' = (u, y) \in E - E_T$

and $e'' = (vy) \in E - E_T$ have been already visited by the DFS. Let $p$ be the common path between the cycles $c(e')$ and $c(e'')$. One of the end points of $p$ is $x$, and the other end point is $y$. Then $I(e) = I(e') + I(e'') + |p| - 1$, $|c(e)| = |c(e')| + |c(e'')| - 2|p| + 1$, and $c(e)$ consists of $c'xc''$, where $c'$ is the cycle $c(e')$ with path $p$ removed, and similarly $c''$ is the cycle $c(e'')$ with path $p$ removed.

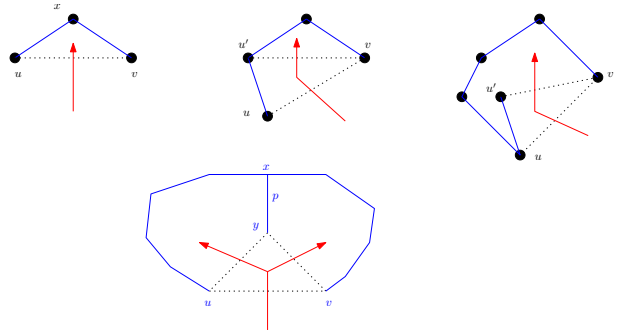



Figure 8.3: Red edges represent the DFS traversal of faces of $D$ corresponding to the tree $T'$. Dashed edges corresponds to edges in $E - E_T$ and blue edges corresponds to edges in $E_T$, i.e. a spanning tree of $D$. Case 1: DFS visits a leaf, i.e. DFS visits a triangle corresponding to $e = (u, v) \in E - E_T$, its degree is one. Case 2: DFS visits a triangle corresponding to $e = (u, v) \in E - E_T$, its degree is two and the other edge of the triangle is $e' = (u', v) \in E - E_T$ which was visited in the previous step and $u' \in c(e)$. Case 3: Same as Case 2 except that $u' \notin c(e)$. Case 4: DFS visits a triangle corresponding to edge $e = (u, v) \in E - E_T$ and its degree is three.

**Lemma 8.2.4** *In the above setting of the graph $D$, there exists an edge $e \in E - E_T$ such that $I(e) \leq 2/3n$ and $n - (I(e) + |c(e)|) \leq 2/3n$.*

**Proof.** Let $e \in E - E_T$ be the first edge in the leaf to root path in $T'$ such that $I(e) + |c(e)| \geq n/3$. Then $n - (I(e) + |c(e)|) \leq 2/3n$. We will prove that $I(e) \leq 2/3n$. The edge $e$ corresponds to one of the four cases encountered in the DFS.

1. In Case 1, $I(e) = 0 \leq 2/3n$.
2. In Case 2, $I(e) + |c(e)| = I(e') + |c(e')| + 1$, and $I(e') + |c(e')| < n/3$, and hence $I(e) + |c(e)| \leq 2/3n$.
3. In Case 3, since $I(e) + |c(e)| = I(e') + |c(e')|$, $e$ cannot be the first edge with this property.
4. In Case 4, $I(e') + |c(e')| < n/3$ and so is $I(e'') + |c(e'')| < n/3$. $I(e) + |c(e)| = I(e') + I(e'') + |p| - 1 + |c(e')| + |c(e'')| - 2|p| + 1 \leq 2/3n - |p| \leq 2/3n$.

∎

## 8.3 Generalizations of the Planar Separator Theorem

In the previous section we saw that the planar separator theorem provides us with a procedure to separate the vertices of a planar graph $G = (V, E)$, $|V| = n$, into three sets $A, B, S$, where $|A|, |B| \leq 2n/3$, $|S| \leq 4\sqrt{n}$, and there exists no edge between $A$ and $B$. In this section we will consider some generalizations of this theorem.

### 8.3.1 Weighted Separators

In our version of the planar separator theorem we considered all vertices to be equal. However, a common variant permits vertices to be weighted such that the sum of all weights is equal to 1 (any other set of non-negative weights can be trivially mapped to one like this). The only difference is that instead of bounding the sizes of the the sets $A$ and $B$ to be $\leq 2n/3$, we bound their weights to be $\leq 2/3$. The separating set $S$ however is still bounded in terms of the number of vertices it contains, and may therefore have arbitrary weight.

To prove the weighted planar separator theorem, our proof from the previous section is sufficient. We need only change certain references to the sizes of sets to refer to the weight of the sets. The rest of the analysis largely follows unchanged.

From now on, we will assume that the planar separator theorem refers to the weighted variant of the planar separator theorem. If no weights are specified, we will assume that all vertices have equal weight, which coincides with our original definition.

### 8.3.2 r-Divisions

In this section we will use the planar separator theorem to construct a more general graph partitioning. The contents of this section are based on a paper by Frederickson [4].

[4] Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16(6):1004–1022, 1987

We define a *region* to be a subset of the vertices of a graph $G = (V, E)$. An *interior vertex* of a region $R$ is contained only in $R$, and adjacent only to other vertices in $R$. A *boundary vertex* is one that is shared between at least two regions. All vertices will be either boundary or interior. Given a parameter $r$, we will divide the graph into $\Theta(n/r)$ regions with $O(r)$ vertices each, and $O(\sqrt{r})$ boundary vertices each. Such a division will be called an *r-division*. Note that the planar separator theorem provides an $n$-division by taking the two sets $A \cup S$ and $B \cup S$.

We begin with a potential naive algorithm. Start with a single region containing all of $V$. While any region $R$ contains more than $r$ vertices, apply the planar separator theorem on $R$ to produce $A, B, S$. Now replace $R$ with $R' = A \cup S$ and $R'' = B \cup S$.

Clearly this procedure produces regions with no more than $r$ vertices, and since it reduces the size of a region by at most $2/3$ until this bound is satisfied, it follows that each region contains $\Theta(r)$ vertices. Consequently, there must be $\Theta(n/r)$ regions. However the number of boundary vertices is more complicated. Initially we have a single region that is made of all interior vertices. Further, $A$ and $B$ consist entirely of interior vertices after constructing $R'$ and $R''$, and $S$ consists entirely of boundary vertices. Therefore we introduce at

most $4\sqrt{n}$ boundary vertices at each recursive step.

To determine the number of boundary vertices, we define $b(v)$ for some vertex $v$ to be one less than the number of regions it is contained in, and $B(n,r)$ to be the sum of $b(v)$ for all $v \in V$. Note that $B(n,r)$ is strictly greater than the number of boundary vertices, as $b(v) \geq 1$ for all boundary vertices, by definition. Our algorithm gives us the following recurrence:

$$\begin{aligned} B(n,r) &\leq 4\sqrt{n} + B(\alpha n + O(\sqrt{n}), r) + B((1-\alpha)n \\ &\quad + O(\sqrt{n}), r) && \text{for } n > r \\ B(n,r) &= 0 && \text{for } n \leq r, \end{aligned}$$

where $1/3 \leq \alpha \leq 2/3$. This recurrence can be solved for $B(n,r) \leq 4n/\sqrt{r} - O(\sqrt{n})$. Therefore, the number of boundary vertices produced by this algorithm is $O(n/\sqrt{r})$. However this tells us nothing about the number of boundary vertices *per region*. Indeed, some regions may have many boundary vertices. To resolve this, we perform further processing on regions with more than $c\sqrt{r}$ boundary vertices, for some constant $c$. Given such a region $R$, we set all $k$ boundary vertices of $R$ to have weight $1/k$, and all interior vertices of $R$ to have weight 0. We then apply the planar separator theorem to $R$ and replace $R$ as before. Since only the boundary vertices have weights, the planar separator theorem will split up the boundary vertices among the two resultant regions. Therefore, after enough iterations all regions will have few enough boundary vertices. Further, since the regions are still strictly shrinking, we cannot have violated the bounds on the size of the region. It remains to be proven that we have not violated the constraint on the maximum number of regions.

If a region has $i > c\sqrt{r}$ boundary vertices, then at most $di/(c\sqrt{r})$ splits will be performed, for some constants $c$ and $d$. This will result in at most $di/(c\sqrt{r})$ new regions. If $t_i$ is the number of regions with $i$ boundary vertices, then the number of new regions will be at most

$$\sum_i (di/c\sqrt{r})t_i = O(n/r)$$

Therefore, our modified algorithm produces an $r$-division.

In our construction, the recursion tree has a depth of $O(\log(n/r))$. Further, for each level we spend $O(n)$ time. Therefore, this algorithm runs in $O(n \log(n/r))$ time.

Further processing on the graph can provide our $r$-division with additional properties such as regions having a constant number of neighbors, and boundary vertices being shared between at most a constant number of regions.

### 8.3.3 Edge Separators

Up until now, we have only considered vertex separators. Edge separators are exactly the same as vertex separators, except that instead of removing vertices, we wish to remove edges. Specifically, given a graph $G = (V, E)$ we wish to find a cut-set $S \subseteq E$ that separates $V$ into two disjoint subsets $A$, $B$. Every edge in $S$ has one endpoint in $A$ and one endpoint in $B$, and every edge in $E \setminus S$ has both of its endpoints in only $A$ or $B$. In general we would like to ensure that $A$ and $B$ are approximately the same size, and $S$ is small.

For graphs with low (e.g. constant) maximum degree, the results for edge separators are generally very similar to those for vertex separators. However on arbitrary planar graphs, edge separators perform much worse.

For instance, consider a graph $G = (V, E)$ in which every vertex has degree 1, except for some vertex $v$ with degree $n - 1$. This graph is a tree, and therefore planar. An excellent vertex separator for $G$ would be $\{v\}$, as it would disconnect the entire graph, allowing us to pick any subsets of $V \setminus \{v\}$ we want for our separated sets. However an edge separator would have to remove a linear number of edges to get balanced sets.

From this example it is clear that not all results for vertex separators hold for edge-separators. In general, vertex separators are more powerful, as for every edge an edge-separator would need to remove, a vertex separator would need to remove at most one vertex, but potentially far fewer. Equivalently, if a vertex separator includes some vertex $v$, an edge separator would need to include every edge of $v$ to achieve the same result. Consequently, results on edge separators often include factors based on the maximum or average degree of the graph [33].

We conclude our look at the planar separator theorem and its generalizations with a table of separator results. There are far too many results on separators with special requirements and for special classes of graphs to adequately report here. As a result, this table is by no means comprehensive. Note that $\Delta(G) = \sum_{v \in V} deg(v)^2$ and $T_{SSSP}(G)$ denotes the time to compute single-source shortest paths in $G$.

| Separator | Graph | # of Sets | Set Sizes | Separator Size | Time | Ref. |
|---|---|---|---|---|---|---|
| Vertex | Tree | 2 | $\leq < 2n/3$ | 1 | $O(n)$ | [74] |
| Vertex | Planar | 2 | $\leq 2n/3$ | $O(\sqrt{n})$ | $O(n)$ | [66] |
| Vertex | Planar | $\Theta(n/r)$ | $O(r)$ | $O(n/\sqrt{r})$ | $O(n \log(n/r))$ | [32] |
| Vertex | Genus $g$ | - | $\leq \epsilon n$ | $O(\sqrt{(g + 1/\epsilon)n})$ | $O(n + g)$ | [2] |
| Vertex | Planar | - | $tw(G)$ | $\leq 4\sqrt{2\sigma(G)/t}$ | $O(n + T_{SSSP}(G))$ | [3] |
| Edge | Planar | - | $tw(G)$ | $\leq 4\sqrt{2\Delta(G)/t}$ | $O(n + T_{SSSP}(G))$ | [3] |

## 8.4 Exercises

**8.1** *Let T=(V,E) be a connected undirected tree such that each vertex has degree at most 3. Let n=|V|. Show that T has an edge whose removal disconnects T into two disjoint subtrees with no more than (2n+1)/3 vertices each. Give a linear time algorithm to find such and edge; prove its correctness.*

**8.2** *Provide an algorithm running in $O(n \log k)$ time to partition the binary tree on n vertices into k ($k \leq n$) subtrees, so that each of the subtree is of size at most $(2/3)^k n$. Try to see whether you can improve the running time of this algorithm (this is not easy!).*

**8.3** *Prove the weighted version of the planar-separator theorem. Let $G = (V, E)$ be an embedded undirected triangulated planar graph, where $n = |V|$. Each vertex $v \in V$ has a positive weight $w(v) \geq 0$ and $\sum_{v \in V} w(v) = 1$. There exists a partition of V into disjoint sets A, B, and S, such that*

1. *$w(A), w(B) \leq \frac{2}{3}$, where $w(A)$ is the sum total of weights of all the vertices in set A*
2. *$|S| \leq 4\sqrt{n}$*
3. *There is no edge in E that joins a vertex in A with a vertex in B.*
4. *Such a set S can be found in linear time.*

**8.4** *Prove Lemma 8.2.2. Let $G = (V, E)$ be a connected planar graph and $G^*$ be its dual. For any $E' \subseteq E$, the subgraph $(V, E')$ has a cycle if and only if the subgraph $(V^*, E - E')$ of $G^*$ is disconnected.*

**8.5** *Let $T = (V, E)$ be a connected undirected tree such that all of its vertices have degree at most 3. Let $n = |V|$. Show that T has an edge whose removal disconnects T into two disjoint subtrees with no more than $\frac{2n+1}{3}$ vertices each.*

**8.6** *Consider the following version of the (weighted) planar-separator theorem. Let $G = (V, E)$ be an embedded undirected triangulated planar graph, where $n = |V|$. Each vertex $v \in V$ has a positive weight $w(v) \geq 0$ and $\sum_{v \in V} w(v) = 1$. There exists a partition of V into disjoint sets A, B, and S, such that*

1. *$w(A), w(B) \leq \frac{2}{3}$, where $w(A)$ is the sum total of weights of all the vertices in set A*
2. *$|S| \leq 4\sqrt{n}$*
3. *There is no edge in E that joins a vertex in A with a vertex in B.*
4. *Such a set S can be found in linear time.*

*Show what changes you need to make in the proof of (unweighted) Planar Separator Theorem to prove the above theorem.*

**8.7** *Prove the following: Let $G = (V, E)$ be a connected planar graph and $G^*$ be its dual. For any $E' \subseteq E$, the subgraph $(V, E')$ has a cycle if and only if the subgraph $(V^*, E - E')$ of $G^*$ is disconnected.*

**8.8** *Prove the following theorem on Geometric Separators. In 2-dimensions assume that you have n squares of arbitrary sizes. Squares are axis aligned. Moreover none of the points in the plane is inside more than k-squares. Prove that there exists either a vertical or a horizontal line which partitions the set of squares in such a way that at least $\lfloor \frac{n+1-k}{4} \rfloor$ of squares interiors lie to each side of the line. How fast you can find such a line?*

# 9
# *Locality-Sensitive Hashing*

The concept of *Locality-Sensitive Hashing* (LSH) is used to determine which items in a given set are similar under some well-defined similarity measure. The key idea is to hash the items using several hash functions. The hash functions have the property that the probability of collision is higher for items that are similar as compared to the items that are dissimilar. Hence the similar items are more likely to hash into the same buckets. Rather than using the naive approach of comparing all pairs of items within a set, LSH technique only compares items within a bucket, thereby reducing the number of comparisons. LSH is often used for finding similar items in very large data sets. LSH provides a method for efficient approximate nearest neighbor search and it has been used in data mining, pattern recognition, computer vision, computational geometry, data compression, spell checking, plagiarism detection, and chemical similarity.

This chapter is organized as follows. We will describe the LSH using an example of finding similar documents in a collection of documents. In Section 9.1 we explain what are shingles in a document, the set comprising of shingles in a document, and the notion of Jaccard similarity to measure the similarity between sets. In Section 9.2 we describe minhashing for summarizing sets. In Section 9.3 we describe the LSH technique for finding sets (documents) having high Jaccard similarity based on minhashing. In Section **??** we discuss finite metric spaces. Section 9.5 discusses the theory of locality sensitive functions. In particular, we define a sensitive family of functions and show how to construct AND and OR families. In Section 9.6 we provide construction of various LSH families including Hamming distance, Cosine distance, Euclidean Distance, Fingerprint similarity, and Image similarities. We also discuss the properties of the similarity measure under which we can apply the theory of LSH. Section **??** consists of some problems for broadening our understanding of LSH technique and its applications. We conclude this chapter by providing some bibliographic remarks.

## 9.1 Similarity of Documents

We will introduce LSH using the problem of finding similar documents. This problem appears, for example, on the web when attempting to find similar, or even duplicate web pages. A search engine would use this technique to allow these similar documents to either be grouped or only shown once on a results page, so that other possible search matches could also be prominently displayed.

**Problem 9.1.1** *Given a collection of web-pages, find the near duplicate web-pages.*

Clearly the content of the page is what matters, so for a preprocessing step any HTML tags are stripped away and main textual content is kept. Typically multiple white spaces are also replaced by a singe or no space during this preprocessing. As a result we are left with a document containing a string of text characters. Keeping in mind that we want to compare similarity of documents opposed to exact equality, the text is split up into a set of smaller strings by a process called *shingling*. This allows the documents to be represented as sets, where fragments of documents can match others. The shingle length $k$ should be large enough so that the probability of any given $k$-shingle appearing in any given document is relatively low. But if the two documents are similar, the probability that a particular shingle will appear in both the documents is higher. For our purposes, it is sufficient to know that choosing $k = 5$ works well for electronic mail, and $k = 9$ is suitable for large text documents. In other words, we look for concatenation of strings made of $k$ words.

**Definition 9.1.2 k-shingle***: A k-shingle of a text document is defined to be any substring of length k which appears in the document.*

**Example 9.1.3** *Let the document $D =$ 'The cow jumped over the moon', and $k = 2$. Then the possible k-shingles for D are:*
$\{Thecow, cowjumped, jumpedover, overthe, themoon\}$.

For simplicity, we will assume that the document contains one long sequence made up of alphabets and we will work with $k$-shingles of the sequence.

**Example 9.1.4** *Let the document $D = \{adbdabadbcdab\}$, and $k = 2$. Then the possible k-shingles for D are: $\{ad, db, bd, da, ab, ba, bc, cd\}$. Note that the set representing the shingles of D consists of unique shingles.*

Note that in the above example, $D$ consists of 13 alphabets, each alphabet requires 1-byte of memory space. For $k = 2$, we need 8 shingles to represent $D$, each requiring 2-bytes of memory space. Thus,

the shingle representation as such increases the memory requirement. However we can work instead with integers by using a hash function to map each $k$-shingle to an integer. For example, consider the following hash function that maps 9-byte shingles to an integer.

**Example 9.1.5** *Let $k = 9$, and let $\mathcal{H}$ be a hash function that maps the set of characters to integers: $\mathcal{H} : |C|^9 \to \mathbb{Z}$. Let $|\mathbb{Z}| \leq 2^{32} - 1$.*

$\mathcal{H}$ potentially reduces the space requirements as a 9-byte shingle is converted to a 4-byte integer. However, this representation of shingles may use 4 times more memory space than the original document. A solution to overcome this is the subject of the next section.

Now that we have documents mapped to sets of $k$-shingles, we can use a similarity measure called *Jaccard similarity* to compare the two sets. The Jaccard similarity is defined with respect to two sets $S$ and $T$, and is the ratio of the size of the intersection of $S$ and $T$ to the size of their union. See Figure 9.1 for an illustration.

**Definition 9.1.6 Jaccard Similarity**: *Let $S$ and $T$ be two sets. Define the Jaccard Similarity of $S$ and $T$ as $\mathsf{SIM}(S, T) = \frac{|S \cap T|}{|S \cup T|}$.*

As a result, we redefine our problem statement as follows:

**Problem 9.1.7** *Given a constant $0 \leq s \leq 1$ and a collection of sets $\mathcal{S}$, find the pairs of sets in $\mathcal{S}$ whose Jaccard similarity is greater than $s$.*

Refer to Example 9.1.3. Suppose we have two documents $D$ and $E$, where $D =$ "The cow jumped over the moon" and $E =$ "The dog jumped over the moon", and let $k = 2$. The shingles corresponding to $D$ and $E$ are

$\{Thecow, cowjumped, jumpedover, overthe, themoon\}$ and

$\{Thedog, dogjumped, jumpedover, overthe, themoon\}$, respectively.

Their Jaccard similarity is $3/7$. We will report that $D$ and $E$ are similar documents for any value of $s \leq 3/7$.

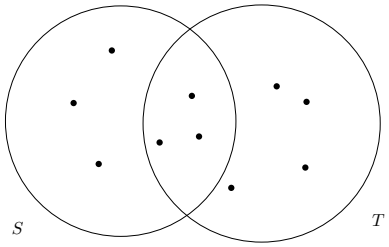


Figure 9.1: Jaccard similarity of two sets. The intersection $|S \cap T| = 3$ and the union $|S \cup T| = 10$. Therefore their Jaccard similarity is $3/10$.

## 9.2 Similarity-Preserving Summaries of Sets

In this section, we will present a solution for the storage of sets of shingles using smaller representations called signatures. These *signatures* will also have the property that the similarity measure between any two will be approximately the same as the similarity between the two sets of shingles which they are derived from.

First, let us consider a natural representation of a set. Let $U$ be the universe from which the elements of the set are drawn. Order the elements of $U$ in some order. A set $S \subseteq U$ can be represented by a 0-1 vector of length $|U|$, where a 1 represents that the corresponding element from the universe is present in the set, and a 0 represents

that the element is absent from the set. Similarly for a collection of sets over the universe $U$, we can associate a *characteristic matrix* $M$, where each column represents the vector corresponding to a set and each row corresponds to an element of $U$.

An example is given in Table 9.1, where we have four sets (representing households in a neighborhood), a universe $U$ consisting of five elements (possible vacation destinations), and the characteristic matrix $M$. Observe that the set $S_1$ prefers cruise and safari, $S_2$ loves to visit resorts, $S_3$ loves Ski, Safari and prefers to stay at home, etc.

| | $S_1$ | $S_2$ | $S_3$ | $S_4$ |
|---|---|---|---|---|
| Cruise | 1 | 0 | 0 | 1 |
| Ski | 0 | 0 | 1 | 0 |
| Resorts | 0 | 1 | 0 | 1 |
| Safari | 1 | 0 | 1 | 1 |
| Stay at Home | 0 | 0 | 1 | 0 |

Table 9.1: A characteristic matrix $M$ for 4 sets $\{S_1, S_2, S_3, S_4\}$. The universe $U$ consists of 5 elements.

One effective way to compute the signature for a collection of sets is to use *minhashing*. For minhashing, the rows of the characteristic matrix are first randomly permuted. Let $\pi$ be a permutation of rows. Then for each set (column in the characteristic matrix), its minhash value $h$ is the index of the first row which is a 1 after applying the permutation $\pi$. In the previous example, suppose we permute the rows by applying the permutation $\pi : 01234 \rightarrow 40312$. The resulting table after permutation is shown as Table 9.2. Observe that the minhash values of the sets with respect to the permutation $\pi$ are: $h(S_1) = 1$, $h(S_2) = 3$, $h(S_3) = 0$, and $h(S_4) = 1$. (Note that the rows are numbered 0,1,2,3,4).

| | $S_1$ | $S_2$ | $S_3$ | $S_4$ |
|---|---|---|---|---|
| Ski | 0 | 0 | 1 | 0 |
| Safari | 1 | 0 | 1 | 1 |
| Stay at Home | 0 | 0 | 1 | 0 |
| Resorts | 0 | 1 | 0 | 1 |
| Cruise | 1 | 0 | 0 | 1 |

Table 9.2: Characteristic matrix after the permutation $\pi : 01234 \rightarrow 40312$ of rows of Table 9.1.

In the following lemma we establish an important connection between the Jaccard similarity of two sets and the probability that their minhash values are the same after a random permutation of the rows of the characteristic matrix. We show that the Jaccard similarity is equal to the probability that these minhash values are the same.

**Lemma 9.2.1** *For any two sets $S_i$ and $S_j$ in a collection of sets $\mathcal{S}$ where the elements are drawn from the universe $U$, the probability that the minhash value $h(S_i)$ equals $h(S_j)$ is equal to the Jaccard similarity of $S_i$ and $S_j$, i.e.,* $Pr[h(S_i) = h(S_j)] = \mathsf{SIM}(S_i, S_j) = \frac{|S_i \cap S_j|}{|S_i \cup S_j|}$.

**Proof.** Focus on the columns representing the sets $S_i$ and $S_j$ in the characteristic matrix before the random permutation. For any row, the entries corresponding to these columns are either (a) both 0, (b) both 1, or (c) one is 0 and the other is 1. Let $X$ be the number of rows of type (b) and let $Y$ be the number of rows of type (c). Observe that the Jaccard similarity of $S_i$ and $S_j$ is $\mathsf{SIM}(S_i, S_j) = \frac{|X|}{|X|+|Y|}$.

Now, what is the probability that when we scan the rows from top to bottom, after the random permutation, we meet a type (b) row before a type (c) row? This is exactly $\frac{|X|}{|X|+|Y|}$, which is also precisely when $h(S_i) = h(S_j)$. ∎

Consider *minhash signature matrix* $\mathsf{SIG}(M)$. These are matrices constructed by repeatedly minhashing a characteristic matrix $M$ of a set system $\mathcal{S}$ with universe $U$ as follows. Pick a set of $n$ random

permutations of rows of $M$. For each set in $\mathcal{S}$ compute its $h$-value with respect to each of the $n$-permutations. This results in $\mathsf{SIG}(M)$ — it consists of $|\mathcal{S}|$ columns and $n$-rows, and the $(i,j)$-th entry corresponds to the signature of the $j$-th set with respect to the $i$-th permutation.

**Example 9.2.2** *Shown in Table 9.3 is the minhash signature matrix* $\mathsf{SIG}(M)$ *created from the characteristic matrix of Table 9.1 using* $n = 2$ *permutations. The first permutation* $\pi_1$ *maps row* $x$ *to* $x + 1 \bmod 5$ *and the second permutation* $\pi_2$ *maps the row* $x$ *to* $3x + 1 \bmod 5$*. (Note that rows are numbered* $0, 1, 2, 3, 4$*.) Let us denote the minhash signatures corresponding to* $\pi_1$ *and* $\pi_2$ *by* $h_1$ *and* $h_2$*, respectively.*

| | $S_1$ | $S_2$ | $S_3$ | $S_4$ |
|---|---|---|---|---|
| $h_1$ | 1 | 3 | 0 | 1 |
| $h_2$ | 0 | 2 | 0 | 0 |

Table 9.3: Signature matrix $\mathsf{SIG}(M)$ for 4 sets corresponding to permutations $\pi_1$ and $\pi_2$ of characteristic matrix $M$ in Table 9.1.

Next we discuss how to compute $\mathsf{SIG}(M)$ efficiently. Since the characteristic matrix $M$ is typically very large, we cannot afford to permute its rows. In place of performing the permutations explicitly, we use several hash functions $h_1, \ldots, h_n$, where each $h_i : \{1, \ldots, U\} \to \{1, \ldots, U\}$, $1 \leq i \leq n$. The $i - th$ row of $M$ is mapped to the row at index $h(i)$ after permutation. Note that $h_i$ may not result in a valid permutation as two different rows of $M$ may hash to the same index due to collisions. Nevertheless this is not a major issue as it avoids the need for explicitly permuting the rows of $M$. Shown below is an outline of the required steps to compute this signature matrix $\mathsf{SIG}(M)$. (If required, we can replace the hash functions by actual permutations, and this will not alter these steps.)

*Step 1:* Initialize each entry of the signature matrix $\mathsf{SIG}(M)$ to $\infty$.

*Step 2:* Pick $n$ random hash functions $h_1, \ldots, h_n$, where $h_i : \{1, \ldots, U\} \to \{1, \ldots, U\}$.

*Step 3:* Execute the following steps for each row $r$ of $M$.

1. Compute $h_1(r), h_2(r), \ldots, h_n(r)$.
2. For $c = 1, \ldots, |\mathcal{S}|$; if $M[r, c] = 1$ then for each $i = 1, \ldots, n$, $\mathsf{SIG}(i, c) := \min(h_i(r), \mathsf{SIG}(i, c))$.

Let us analyze the running time of the above algorithm. Step 1 requires $O(n|\mathcal{S}|)$ time. Step 2 requires time proportional to computing $n$ hash functions. For Step 3, observe that for each non-zero entry of $M$, we compute $n$ hash values, and this requires $O(n)$ time. So the total computation time is upper bounded by $O(n|\mathcal{S}||U|)$, or more precisely $O(|\mathcal{S}||U| + n|K|)$, where $K$ is the total number of elements in all the sets. We do not need any additional memory except to store the description of $n$ hash functions and the signature matrix $\mathsf{SIG}(M)$ and . Although the signature matrix is fairly small compared to the characteristic matrix, its size could still be large.

Just to have some perspective. Every year, Carleton admits approximately 5000 students. Each student typically takes 5 courses and each course consists of usually 4 assignments. So in all $5000 * 5 * 4 = 100,000$ assignments (i.e., documents) are generated by the students each term. If we want to find near similar documents by comparing every pair of documents, we need to evaluate $\binom{100,000}{2} = 10^{10}$ pairs of documents. If the comparison between a pair of documents requires $10^{-5}$ seconds, it will take $\sim$ 28-hours to find near similar documents. Instead, suppose we generate 125 signatures, each of size 4-bytes, for each of the documents. Then we have a signature matrix consisting of $125 * 100,000$ signatures of total size that equals to 50Mb. In the next section, we will introduce the Locality-Sensitive Hashing (LSH) technique and show how we can find near similar documents (without using pairwise direct comparisons) very efficiently using the signature matrix.

## 9.3 *LSH for Minhash Signatures*

First we replace each document by its shingles forming a well-defined set. From these sets and by the application of minhashing concept of the previous section, we construct the signature matrix. Following the notation of the previous section, let the signature matrix be $\mathsf{SIG}(M)$ for the set of documents $\mathcal{S}$. We partition the rows of this matrix into $b = n/r$ *bands*, where each band is made of $r$-consecutive rows. See Table 9.4 for an illustration. For simplicity, we assume that $r$ divides $n$. For each band we define a hash function $h : \mathbb{R}^r \to \mathbb{Z}$, which takes a column vector of length $r$ and maps it to an integer (i.e. a bucket). If we want we can even choose the same hash function for all the bands, but the buckets are kept distinct for each band. Now if two vectors of length $r$ in any one of the bands hash to the same bucket, we declare that the corresponding sets (documents) are potentially similar.

**Lemma 9.3.1** *Let $s > 0$ be the Jaccard similarity of two sets. The probability that the minhash signature matrix agrees in all the rows of at least one of the bands for these two sets is $f(s) = 1 - (1 - s^r)^b$.*

**Proof.** The proof is straightforward and uses the following chain of simple arguments.

1. From Lemma 9.2.1, the probability that the minhash signatures for these two sets are the same in any particular row of the signature matrix is $s$.

2. The probability that the signatures agree in all the rows in one particular band is $s^r$. The probability is computed by taking AND

| Band # | $S_1$ | $S_2$ | $S_3$ | $S_4$ | $S_5$ | $S_6$ | $S_7$ | $S_8$ | $S_9$ | $S_{10}$ | $S_{11}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Band 1 | 2 | 2 | 1 | 0 | 0 | 1 | 3 | 2 | 5 | 0 | 3 |
| | 1 | 3 | 2 | 0 | 2 | 2 | 1 | 4 | 2 | 1 | 2 |
| | 3 | 0 | 3 | 0 | 4 | 3 | 2 | 0 | 0 | 4 | 2 |
| Band 2 | 0 | 4 | 3 | 1 | 5 | 3 | 3 | 2 | 3 | 5 | 4 |
| | 2 | 1 | 1 | 0 | 4 | 1 | 2 | 1 | 4 | 2 | 5 |
| | 4 | 2 | 1 | 0 | 5 | 2 | 3 | 2 | 3 | 5 | 4 |
| Band 3 | 2 | 4 | 3 | 0 | 5 | 3 | 3 | 4 | 4 | 5 | 3 |
| | 0 | 2 | 4 | 1 | 3 | 4 | 3 | 2 | 2 | 2 | 4 |
| | 0 | 2 | 1 | 0 | 5 | 1 | 1 | 1 | 1 | 5 | 1 |
| Band 4 | 0 | 5 | 1 | 0 | 2 | 1 | 3 | 2 | 1 | 5 | 4 |
| | 1 | 3 | 1 | 0 | 5 | 2 | 3 | 3 | 6 | 3 | 2 |
| | 0 | 5 | 2 | 1 | 5 | 1 | 2 | 2 | 6 | 5 | 4 |

Table 9.4: Partitioning of a signature matrix for $|\mathcal{S}| = 11$ sets, with $n = 12$ hash functions, into four bands ($b = 4$) of three rows each ($r = 3$). Note that (a) in Band 1, the sets $\{S_3, S_6\}$ are hashed into the same buckets, (b) in Band 3, $\{S_3, S_6, S_{11}\}$ are hashed into the same bucket, and also $\{S_8, S_9\}$ are hashed into the same bucket (possibly different from the bucket consisting of $\{S_3, S_6, S_{11}\}$), and (c) In Band 4, $\{S_2, S_{10}\}$ are hashed into the same bucket.

of $r$ independent events.

3. The probability that the signatures do not agree in at least one of the rows in this band is $1 - s^r$. This is the probability of the complementary event. Alternatively, one can think of this as the probability of an OR-event, i.e. the signatures do not agree in the first row OR the signatures do not agree in the second row OR ... OR the signatures do not agree in the last row.

4. The probability that the signatures do not agree in any of the $b$ bands is $(1 - s^r)^b$.

5. Therefore, the probability that the signatures agree in at least one of the bands is $f(s) = 1 - (1 - s^r)^b$.

∎

**Corollary 9.3.2** *In the above method, the probability that the two sets with Jaccard similarity $0 \leq s \leq 1$ are detected similar is $1 - (1 - s^r)^b$.*

In Table 9.5 we evaluate the probability function $f(s) = 1 - (1 - s^r)^b$ for different values of $s, b$, and $r$.

The graphical representation of $f(s) = 1 - (1 - s^r)^b$ is in Figure 9.2. In the graph, $x$-axis represents values of $s$ and $y$-axis represents the value of the probability function $f(s)$. As we can see the curve is $S$-shaped for different combinations of values of $b$ and $r$. We observe that as $s \to 1$, the probability function $f(s) = 1 - (1 - s^r)^b \to 1$, i.e., the higher the Jaccard similarity between two sets, the probability that these two sets will map to the same bucket is high.

One important aspect of this curve is that the steepest slope occurs at the value of $s$ which is approximately $t = (1/b)^{(1/r)}$ and this can derived as follows. To find the steepest slope, we need to compute for

| $(b, r)$ <br> $f(s) = 1 - (1 - s^r)^b \searrow$ | $(4,3)$ | $(16,4)$ | $(20,5)$ | $(25,5)$ | $(100,10)$ |
|---|---|---|---|---|---|
| $s = 0.2$ | 0.0316 | 0.0252 | 0.0063 | 0.0079 | 0.0000 |
| $s = 0.4$ | 0.2324 | 0.3396 | 0.1860 | 0.2268 | 0.0104 |
| $s = 0.5$ | 0.4138 | 0.6439 | 0.4700 | 0.5478 | 0.0930 |
| $s = 0.6$ | 0.6221 | 0.8914 | 0.8019 | 0.8678 | 0.4547 |
| $s = 0.8$ | 0.9432 | 0.9997 | 0.9996 | 0.9999 | 0.9999 |
| $s = 1.0$ | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| $t = (\frac{1}{b})^{(\frac{1}{r})}$ | 0.6299 | 0.5 | 0.5492 | 0.5253 | 0.6309 |

Table 9.5: Values of the function $f(s) = 1 - (1 - s^r)^b$ for different values of $s, b$, and $r$.

what values of $s$, $f''(s) = 0$. It turns out that $s = (\frac{r-1}{br-1})^{\frac{1}{r}}$ results in the steepest slope. For values of $br >> 1$, $s \approx (\frac{1}{b})^{\frac{1}{r}}$. In other words, if the Jaccard similarity $s$ of the two sets is above the threshold $t = (\frac{1}{b})^{\frac{1}{r}}$, then the probability that they will be found potentially similar is very high. For example, the last row of Table 9.5 lists the thresholds. Consider the entries in the row corresponding to $s = 0.8$ and observe that most of the values for $f(s = 0.8) \rightarrow 1$ as $s > t$.

What this technique has done is to give us some idea in terms of which sets are very likely to be similar. If required, we can actually compare these potential pairs of sets (or their minhash signatures) to find out whether they are actually similar. Note that in this technique we need to choose appropriate values of parameters $n, b$, and $r$, given the value of the threshold $t$. Then any pairs of sets whose Jaccard similarity $s > t$ will likely be classified as similar sets. Increasing the value of $n$ results in higher running time as we have to compute those many minhash signatures. Choosing smaller values of $n$ results in less accurate results. See, for example, the results for $n = 12$ corresponding to the $b = 4, r = 3$ column in Table 9.5 in comparison to the results for $n = 125$ corresponding to the $b = 25, r = 5$ column or the results for $n = 1000$ corresponding to the $b = 100, r = 10$ column.

At an abstract level what we have done here is to use a family of functions (the minhash functions) and the banding technique (with parameters $b$ and $r$) to distinguish between pairs which are at a low distance (similar) to the pairs which are at a large distance (dissimilar). The steepness of the $S$ curve suggests a threshold where this technique can be effective. In the next section, we will see that there are other families of functions that can be considered to separate the pairs which are at a low distance from the pairs which are at a higher distance.

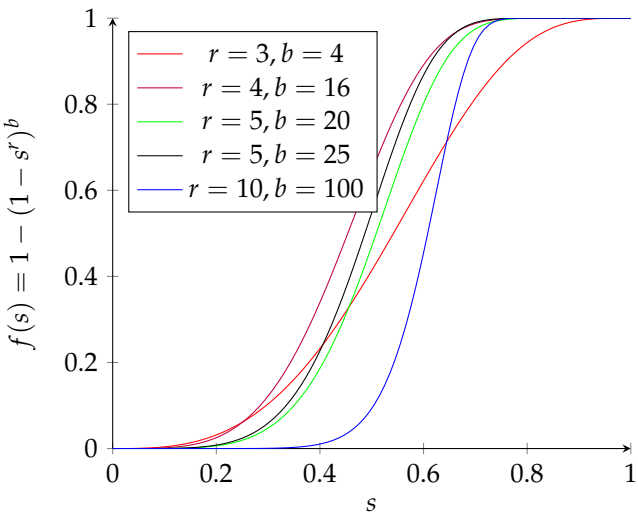



Figure 9.2: The S-curve $f(s) = 1 - (1 - s^r)^b$ for different values of $b$ and $r$ of Table 9.5.

## 9.4 *Metric Space*

Consider a finite set $X$. A *metric* or *distance measure* $d$ on $X$ is a function

$$d : X \times X \to [0, \infty)$$

satisfying the following properties. For all elements $u, v, w \in X$:

1. Non-negativity: $d(u, v) \geq 0$.
2. Symmetric: $d(u, v) = d(v, u)$.
3. Identity: $d(u, v) = 0$ if and only if $u = v$.
4. Triangle Inequality: $d(u, v) + d(v, w) \geq d(u, w)$.

Consider the following examples of metric spaces.

**Example 9.4.1 (Euclidean Distance)** *Let $X$ be a set of n-points in plane. Euclidean distance between any two points $p_i = (x_i, y_i)$ and $p_j = (x_j, y_j)$ of $X$ is defined as $d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. Observe that $X$ with the Euclidean distance measure satisfies the metric properties.*

**Example 9.4.2 (Jaccard Distance)** *Let $\mathcal{S}$ be a collection of sets. First observe that the Jaccard Similarity doesn't satisfy the metric properties as for any set $S \in \mathcal{S}$, the Jaccard similarity $\mathsf{SIM}(S, S) = 1$. This violates the identity property. Define the Jaccard distance between two sets $S_i, S_j \in \mathcal{S}$ as $\mathsf{JD}(S_i, S_j) = 1 - \mathsf{SIM}(S_i, S_j)$. Since $0 \leq \mathsf{SIM}(S_i, S_j) \leq 1$, $\mathsf{JD}(S_i, S_j) \geq 0$. Thus Jaccard distance satisfies the non-negativity property. Symmetry and Identity is obvious. To show the triangle inequality, it may be best to look at the relationship between the minhash signatures and Jaccard similarity. By Lemma 9.2.1 we know that $Pr[h(S_i) = h(S_j)] = \mathsf{SIM}(S_i, S_j)$. Since*

$\mathsf{JD}(S_i, S_j) = 1 - \mathsf{SIM}(S_i, S_j)$, *thus* $\mathsf{JD}(S_i, S_j) = Pr[h(S_i) \neq h(S_j)]$. *Now consider three sets* $S_i, S_j, S_k \in \mathcal{S}$. *To show that* $\mathsf{JD}(S_i, S_j) + \mathsf{JD}(S_j, S_k) \geq \mathsf{JD}(S_i, S_k)$, *it is enough to show that* $Pr[h(S_i) \neq h(S_j)] + Pr[h(S_j) \neq h(S_k)] \geq Pr[h(S_i) \neq h(S_k)]$. *Observe that if* $h(S_i) \neq h(S_k)$ *then at least* $h(S_i) \neq h(S_j)$ *or* $h(S_j) \neq h(S_k)$ *must hold. If* $h(S_i) = h(S_j)$ *and* $h(S_j) = h(S_k)$ *then it will follow that* $h(S_i) = h(S_k)$. *Thus, the set* $\mathcal{S}$ *with the Jaccard distance measure satisfies the metric properties.*

**Example 9.4.3** *Let* $X$ *be a set of* $n$ *elements, where the distances between any pair of elements* $x, y \in X$ *are defined as follows.*

$$d(x, y) = \begin{cases} 1, & \text{if } x \neq y \\ 0, & \text{otherwise} \end{cases}$$

*Observe that* $X$ *with distance function* $d$ *satisfies the metric properties.*

**Example 9.4.4 (Hamming Distance)** *Consider the space* $X$ *of* $d$*-dimensional Boolean vectors. Consider two vectors* $x, y \in X$. *The* Hamming distance $\mathsf{HAM}(x, y)$ *is defined as the number of coordinates in which* $x$ *and* $y$ *differ. For example, let* $x = 110011$ *and* $y = 100111$. *Then* $\mathsf{HAM}(x, y) = 2$, *as they differ in exactly two coordinates. Observe that the Hamming distance is non-negative, symmetric, and that the distance between two identical vectors is* 0. *The Hamming distance between any three vectors* $x$, $y$, *and* $z$ *also satisfies the triangle inequality* $\mathsf{HAM}(x, y) + \mathsf{HAM}(y, z) \geq \mathsf{HAM}(x, z)$ *since the number of components in which* $x$ *differs from* $z$ *cannot be larger than the sum of the number of components in which* $x$ *differs from* $y$ *and* $y$ *differs than* $z$. *Therefore, we can use Hamming distance as a metric over the* $d$*-dimensional vectors.*

## 9.5 Theory of Locality Sensitive Functions

In this section we will consider a family of functions $\mathcal{F}$. The families are typically comprised of hash functions. We say that a hash function $f \in \mathcal{F}$ identifies two items $x$ and $y$ to be similar if $f$ hashes them to the same bucket. We use the notation $f(x) = f(y)$ to denote that $x$ and $y$ are hashed to the same bucket. For example, the minhash functions seen previously form a family of functions. Recall the definition of a distance measure from Section 9.4. Next we define a notion of *sensitive family* that encapsulates the idea that if two items are close to each other with respect to the distance measure, then the probability that they hash to the same bucket by any function $f \in \mathcal{F}$ will be high. Conversely, if the two items are far from each other, then the probability that they hash to the same bucket will be low.

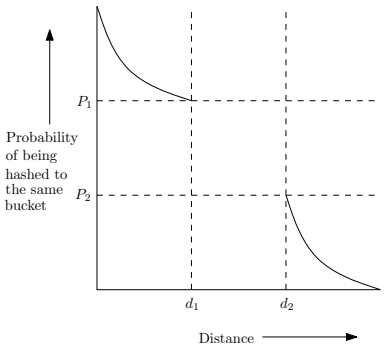


Figure 9.3: A smaller distance between items corresponds to a higher probability of similarity.

**Definition 9.5.1** *Let* $d$ *be a distance measure and let* $d_1 < d_2$ *be two distances in this measure. Let* $0 \leq p_2 < p_1 \leq 1$. *A family of functions* $\mathcal{F}$

*is said to be $(d_1, d_2, p_1, p_2)$-sensitive if for every $f \in \mathcal{F}$ the following two conditions hold;*

1. *If $d(x,y) \leq d_1$ then $Pr[f(x) = f(y)] \geq p_1$.*
2. *If $d(x,y) \geq d_2$ then $Pr[f(x) = f(y)] \leq p_2$.*

*See Figure 9.3 for an illustration.*

**Example 9.5.2** *Consider the Jaccard distance measure for finding similar sets in a collection of sets $\mathcal{S}$. Let $0 \leq d_1 < d_2 \leq 1$. The family of minhash-signatures is $(d_1, d_2, p_1 = 1 - d_1, p_2 = 1 - d_2)$-sensitive and this can be argued as follows. Suppose that the Jaccard similarity between two sets is at least $s$. Then their Jaccard distance is at most $d_1 = 1 - s$. By Lemma 9.2.1 the probability that they will be hashed to the same bucket by minhash signatures is $\geq p_1 = 1 - d_1$. Similarly, suppose that the Jaccard similarity is at most $s'$. Then their Jaccard distance is at least $d_2 = 1 - s'$. The probability that the minhash signatures map them to the same bucket is at most $p_2 = 1 - d_2$.*

Next we look into amplifying sensitive families using AND and OR constructions. Suppose we have a $(d_1, d_2, p_1, p_2)$-sensitive family $\mathcal{F}$. We can construct a new family $\mathcal{G}$ by an *AND-construction* as follows. Each function $g \in \mathcal{G}$ is formed from a set of $r$ independently chosen functions of $\mathcal{F}$, say $f_1, f_2, \ldots, f_r$ for some fixed value of $r$. Now, $g(x) = g(y)$ if and only if for all $i = 1, \ldots, r$, $f_i(x) = f_i(y)$.

**Claim 9.5.3** *$\mathcal{G}$ is an $(d_1, d_2, p_1^r, p_2^r)$-sensitive AND family.*

**Proof.** Each function $f_i$ is chosen independently. For any $0 \leq p \leq 1$, if $p$ is the probability that any $f_i \in \mathcal{F}$ will hash two items $u$ and $v$ to the same bucket, then the probability that any function $g \in \mathcal{G}$ will hash $u$ and $v$ to the same bucket is $p^r$. This is the probability of all the $r$ independent events to occur simultaneously. ∎

Similarly we can construct $\mathcal{G}$ by an *OR-construction*. Each member $g$ in $\mathcal{G}$ is constructed by taking $b$ independently chosen members $f_1, f_2, \ldots, f_b$ from $\mathcal{F}$. We say that $g(x) = g(y)$ if and only if $f_i(x) = f_i(y)$ for at least one of the members in $\{f_1, f_2, \ldots, f_b\}$.

**Claim 9.5.4** *$\mathcal{G}$ is an $(d_1, d_2, 1 - (1 - p_1)^b, 1 - (1 - p_2)^b)$-sensitive OR family.*

**Proof.** Each function $f_i$ is chosen independently, and this is the probability of at least one of the $b$-events to occur. We can compute this by first finding the probability that none of the $b$ events occur. Let $p$ be the probability that any function $f_i \in \mathcal{F}$ hashes two items $u$ and $v$ to the same bucket. Then $1 - p$ is probability that $f_i$ does not

hash them to the same bucket, and $(1-p)^b$ is the probability that none of the functions $\{f_1, f_2, \ldots, f_b\}$ hash them to the same bucket. Thus, $1-(1-p)^b$ is the probability that at least one of the functions $\{f_1, f_2, \ldots, f_b\}$ hash them to the same bucket. ∎

Observe that the AND-construction reflects rows within a band in Table 9.4. Moreover, the OR-construction reflects the combination of various bands in that table. Furthermore, AND construction lowers all the probabilities, and by choosing the family $\mathcal{F}$ and the parameter $r$, we can try to push $p_2^r \to 0$. The OR-construction increases all the probabilities, and by choosing $\mathcal{F}$ and $b$ appropriately we can try to push $1-(1-p_1)^b \to 1$. This is the essence of the idea of amplification. Next we look into some concrete examples.

Let us play with some values of $b$ and $r$ to see the effect of AND and OR-constructions. We first construct an AND-family $\mathcal{F}_1$ for a certain value of $r$, and then construct an OR-family $\mathcal{F}_2$ for a certain value of $b$. Next we construct an AND-OR family $\mathcal{F}_3$ by first constructing an AND family followed by an OR family. Similarly we construct the family $\mathcal{F}_4$ by first constructing an OR-family followed by an AND-family. Let us look at the amplification of the probabilities in Table 9.6 for different values of $p$.

| | $\mathcal{F}_1$ (AND) | $\mathcal{F}_2$ (OR) | $\mathcal{F}_3$ (AND-OR) | $\mathcal{F}_4$ (OR-AND) |
|---|---|---|---|---|
| $p$ | $p^r$ | $1-(1-p)^b$ | $1-(1-p^r)^b$ | $(1-(1-p)^r)^b$ |
| 0.2 | 0.0001 | 0.6723 | 0.0079 | 0.0717 |
| 0.4 | 0.0256 | 0.9222 | 0.1216 | 0.4995 |
| 0.6 | 0.1296 | 0.9897 | 0.5004 | 0.8783 |
| 0.7 | 0.2401 | 0.9975 | 0.7446 | 0.9601 |
| 0.8 | 0.4096 | 0.9996 | 0.9282 | 0.9920 |
| 0.9 | 0.6561 | 0.9999 | 0.9951 | 0.9995 |

Table 9.6: Illustration of four families obtained for different values of $p$. $\mathcal{F}_1$ is the AND family for $r=4$. $\mathcal{F}_2$ is OR family for $b=5$. $\mathcal{F}_3$ is the AND-OR family for $r=4$ and $b=5$. $\mathcal{F}_4$ is the OR-AND family for $r=4$ and $b=5$.

Let us try to understand the columns of Table 9.6. Let $\mathcal{F}$ be a $(0.2, 0.6, 0.8, 0.4)$-sensitive minhash function family. This means if the distance between two sets $S_i$ and $S_j$ is $\leq 0.2$, any function in $\mathcal{F}$ will hash them to the same bucket with probability $\geq 0.8$. Similarly if the distance between them is $\geq 0.6$, with probability at most 0.4 they will hash to the same bucket. Let us focus our attention on rows corresponding to $p_2 = 0.4$ and $p_1 = 0.8$ in Table 9.6. For the AND-family $\mathcal{F}_1$ we see that $p^r = p^4$ is substantially lower than $p$, but still $p_2^4 \to 0$ and $p_1^4$ is away from 0. For the OR-family $\mathcal{F}_2$ for $b=5$, $1-(1-p)^5 \geq p$, but still the value corresponding to $p_1$ tends towards 1 and the value corresponding to $p_2$ is away from 1. More interesting are the last two columns. Notice that the AND-OR family $\mathcal{F}_3$ corresponds to the LSH for minhash signature family

of Section 9.3. Here we first apply a $r$-way AND-construction and then a $b$-way OR-construction. The AND-construction converts any probability $p$ to $p^r$. This, when followed by a $b$-way OR-construction further converts the probability to $1-(1-p^r)^b$. As we can see from Table 9.6 the value corresponding to $p_1$ tends towards 1 and the value corresponding to $p_2$ is closer to 0. Notice that the value of the function $1-(1-p^r)^b$ with respect to the values of $p$ forms an $S$-curve. Its fixed-point $p = 1-(1-p^4)^5$ is $p \approx 0.6672$ and its threshold value $t = (\frac{1}{b})^{(\frac{1}{r})} \approx 0.6687$.[1] This implies that for values of $p$ significantly less than 0.6672, AND-OR family amplifies it towards 0. Similarly, values of $p$ larger than 0.6672 are amplified to 1. Notice that this technique amplifies the probabilities in the right direction (away from threshold and fix-point), provided we can apply the function several times (e.g. 20 times in our example with $r = 4$ and $b = 5$). The OR-AND family $\mathcal{F}_4$ does not provide anything interesting. In this construction first we have applied a $r$-way OR-construction followed by a $b$-way AND-construction.

[1] Thanks to WolframAlpha

## 9.6 LSH Families

In the previous section we saw LSH families for the Jaccard distance measure. In this section we will construct LSH-families for various other distance measures.

### 9.6.1 LSH family for Hamming Distance

Consider two $d$-dimensional Boolean vectors $x$ and $y$. Recall from Example 9.4.4 that the Hamming distance $\mathsf{HAM}(x,y)$ is defined as the number of coordinates in which $x$ and $y$ differ. We construct a locality-sensitive family for the $d$-dimensional Boolean vectors as follows. Let $f_i(x)$ denote the $i$-th coordinate of $x$. For two vectors $x$ and $y$, the probability that $f_i(x) = f_i(y)$ for a randomly chosen coordinate $i$ will equal the number of coordinate agreements out of the total number of coordinates. Since the vectors $x$ and $y$ disagree in $\mathsf{HAM}(x,y)$ positions out of $d$ positions, then they agree in $d - \mathsf{HAM}(x,y)$ positions. Hence $Pr[f_i(x) = f_i(y)] = 1 - \frac{\mathsf{HAM}(x,y)}{d}$.

**Claim 9.6.1** *For any $d_1 < d_2$, $\mathcal{F} = \{f_1, f_2, \ldots, f_d\}$ is a $(d_1, d_2, 1 - d_1/d, 1 - d_2/d)$-sensitive family of hash functions.*

**Proof.** Recall Definition 9.5.1. Let $p_1 = 1 - d_1/d$ and $p_2 = 1 - d_2/d$. A family of functions $\mathcal{F}$ is said to be $(d_1, d_2, p_1, p_2)$-sensitive if for every $f_i \in \mathcal{F}$ the following two conditions hold:

1. If $\mathsf{HAM}(x,y) \leq d_1$ then $Pr[f_i(x) = f_i(y)] \geq p_1$

2. If $\mathsf{HAM}(x, y) \geq d_2$ then $Pr[f_i(x) = f_i(y)] \leq p_2$

■

### 9.6.2 *LSH family for similarity of vectors using dot products*

Consider two $d - dimensional$ vectors $x = (x_1, x_2, \ldots, x_d)$ and $y = (y_1, y_2, \ldots, y_d)$ with tails at the origin $o$. Their dot product is defined as $x \cdot y = \sum_{i=1}^{d} x_i y_i$. Note that the dot product is positive if the angle between the vectors is between 0 and $\pi/2$ and is negative if the angle is between $\pi/2$ and $\pi$. The vectors $x$ and $y$ define a plane, say $P$. Consider the intersection of any d-dimensional hyperplane $H$ (different than $P$) passing through $o$. The intersection between $H$ and $P$ defines a line $h$ passing through $o$. The vectors $x$ and $y$ may or may not be on the same side of $h$. Let $v$ be the normal vector to $H$ and passing through $o$ in the plane $P$. To determine whether $x$ and $y$ are on the same side of $h$, we can compute the dot products $v \cdot x$ and $v \cdot y$. If the dot product has the same same sign, then $h$ does not separate $x$ from $y$. Whereas, if they have different sign, then $h$ separates them, as the angle between $v$ and one of $x$ or $y$ will be less than $\pi/2$, and with the other one it will be more than $\pi/2$.

For example, in Figure 9.4, the hyperplane $H_1$ intersects the plane containing vectors $x$ and $y$ in line $h_1$ passing through $o$. Note that the dot products $v_1 \cdot x$ and $v_1 \cdot y$ have different signs, where $v_1$ is the normal to $H_1$ passing through $o$. Furthermore, $H_2$ intersects the plane containing vectors $x$ and $y$ in the line $h_2$, and $v_2 \cdot x$ and $v_2 \cdot y$ have the same sign, where $v_2$ is the normal to $H_2$ passing through $o$.

Let us choose a random hyperplane $H$ passing through $o$. We want to estimate the probability that it will separate the vectors $x$ and $y$, i.e. what is the probability that $x$ will be one side of $H$ and $y$ is on the other side? Let the angle between $x$ and $y$ be $\theta$. It is easy to see that for $H$ to separate $x$ and $y$, the line $h$ (and the corresponding normal vector $v$) have to be in a particular sector of angle $\theta$ at $o$. If $h$ falls within this sector, then $H$ separates $x$ from $y$. Or equivalently we say that the dot products $v \cdot x$ and $v \cdot y$ have different signs, which we write as $f_v(x) \neq f_v(y)$, where $f_v(x)$ represents the sign of the dot product $v \cdot x$. Therefore, $Pr[f_v(x) \neq f_v(y)] = \theta/\pi$ and $Pr[f_v(x) = f_v(y)] = 1 - \theta/\pi$.

We construct a sensitive family of functions $\mathcal{F}$ as follows. We select a set of random $d - dimensional$ vectors $v$ anchored at origin. These vectors constitute the set $\mathcal{F}$. Given any d-dimensional vector $x$, for each vector $v \in \mathcal{F}$, we compute the sign of the dot product $v \cdot x$, and store it as $f_v(x)$. From the above arguments, it is easy to see that if the angle between two vectors $x$ and $y$ is at most $d_1 = \theta_1$, then

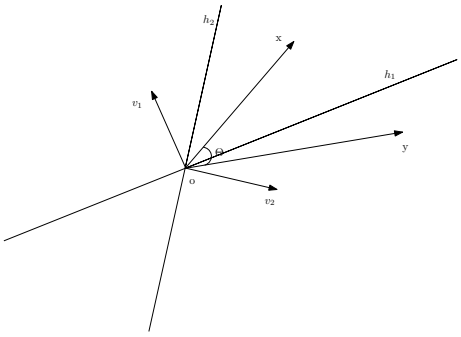



Figure 9.4: Two vectors $x$ and $y$ shown in a plane. The intersection of hyperplanes $H_1$ and $H_2$ with the plane containing $x$ and $y$ forms lines $h_1$ and $h_2$. Vectors $v_1$ and $v_2$ are normal to $H_1$ and $H_2$, respectively, and are in the plane containing $x$ and $y$. Observe that $v_1 \cdot x > 0$, $v_1 \cdot y < 0$, $v_2 \cdot x > 0$, $v_2 \cdot y > 0$.

the probability that for any $v \in \mathcal{F}$, $Pr[f_v(x) = f_v(y)] \geq 1 - d_1/\pi$. Similarly if the angle is at least $d_2 = \theta_2$, then $Pr[f_v(x) = f_v(y)] \leq 1 - d_2/\pi$. Thus, our resulting family $\mathcal{F}$ is a $(d_1, d_2, 1 - d_1/\pi, 1 - d_2/\pi)$-sensitive family of functions.

### 9.6.3 LSH family for Near Neighbors in 2-dimensions

Consider a set of points $P$ in a 2-dimensional space. We are interested in finding pairs of points which are within certain distance to each other, say $\Delta$. Each hash function $f$ in the family $\mathcal{F}$ will be represented by a line $l$ with random orientation in this space. We partition $l$ into intervals of equal size $2\Delta$, and orthogonally project all points of $P$ on $l$. For a point $x \in P$, let $f_l(x)$ denote the interval in which $x$ lies after the projection. If two points $x, y \in P$ lie in the same interval after projection, then $f_l(x) = f_l(y)$. Next, we show that if the distance between $x$ and $y$ is at most $\Delta$, then with probability at least $1/2$, $f_l(x) = f_l(y)$, i.e. if $d(x, y) \leq \Delta$ then $Pr[f_l(x) = f_l(y)] \geq 1/2$. Moreover, if $d(x, y) > 4\Delta$, then $Pr[f_l(x) = f_l(y)] \leq 1/3$. Using this method, if two points are close to each other then there are high chances that they will project to the same interval. Conversely, if the points are far away, it is unlikely they will project to the same interval.

Without loss of generality, we assume that the line $l$ is horizontal. Let $x$ and $y$ be two points in the plane. Now we show that if $d(x, y) \leq \Delta$, then $Pr[f_l(x) = f_l(y)] \geq 1/2$. Let $m$ be the mid-point of the interval $f_l(x)$. With probability $1/2$, the projection of $x$ lies to the left of $m$ in $f_l(x)$. Furthermore, with probability $1/2$, the projection of $y$ lies to the right of projection of $x$. In this case, since $d(x, y) \leq \Delta$, projection of $y$ lies in $f_l(x)$ (i.e., $f_l(x) = f_l(y)$). Thus with probability $1/4$, the projections of $x$ and $y$ lie in $f_l(x)$ where the projection of $x$ is to the left of $m$ and the projection of $y$ is to the right of the projection of $x$. Similarly, with probability $1/4$, projections of $x$ and $y$ lie in $f_l(x)$ where the projection of $x$ is to the right of $m$ and the projection of $y$ is to the left of projection of $x$. Since the above two cases are mutually exclusive, if $d(x, y) \leq \Delta$, then $Pr[f_l(x) = f_l(y)] \geq 1/2$.

Next we show that if $d(x, y) > 4\Delta$, then $Pr[f_l(x) = f_l(y)] \leq 1/3$. As before let $l$ be horizontal and let $\theta$ be the angle of the line passing through $x$ and $y$ with respect to $l$. For the projections of $x$ and $y$ to fall in the same interval, we will need that $d(x, y) \cos\theta \leq 2\Delta$. For this to happen $\cos\theta \leq 1/2$, or the angle the line $xy$ forms with the horizontal needs to be between $60^\circ$ and $90^\circ$. Observe that there is at most $1/3$rd chance that the angle between the horizontal and $xy$ is in that range. See Figure 9.5 for an illustration.

Hence, the family with respect to the projection on a random line with intervals of size $2\Delta$ is a $(\Delta, 4\Delta, 1/2, 1/3)$-sensitive family. Again,

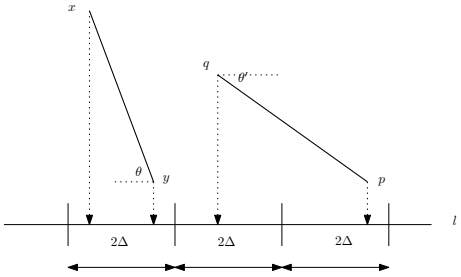



Figure 9.5: Points $x$ and $y$ project to the same interval on $l$, whereas $p$ and $q$ project to different intervals.

we can amplify these probabilities by taking combination of ANDs and ORs as in Section 9.5.

### 9.6.4 LSH family for answering near-neighbor queries

Suppose we are given a set of $n$ points $P$ in $D$-dimensional Euclidean space $\mathbb{R}^D$. For a query point $q \in \mathbb{R}^D$, if there is a point $p \in P$ such that $d(q,p) = R$, then any point $x \in P$ that is within a distance of $(1+\epsilon)R$ from $q$, for some constant $\epsilon > 0$, is called a $(R,\epsilon)$-*near neighbor* (denoted as $(R,\epsilon)$-NN) of $q$. (Note that all the distances are Euclidean distance in this subsection.) If $p$ happens to be the closest point of $q$ in $P$, an $(R,\epsilon)$-NN is usually referred to as an *approximate nearest neighbor*. We first sketch a construction of a family $\mathcal{F}$ of hash functions $f$ to answer $(R,\epsilon)$-NN queries, where for all $x \in P$ and given value of $R$, the following holds (with high probability):

- if $x$ is a $(R,\epsilon)$-NN of $q$, $f(q) = f(x)$.
- if $x$ is not a $(R,\epsilon)$-NN of $q$, $f(q) \neq f(x)$.

We will utilize the data structure for answering $(R,\epsilon)$-NN queries to answer the approximate nearest neighbor queries as follows. Let $d_{\min}$ and $d_{\max}$ be the smallest and the largest inter-point distances in $P$, respectively. We will construct data structures for $(R,\epsilon)$-NN queries for values of $R = d_{\min}/\epsilon, d_{\min}, (1+\epsilon)d_{\min}, (1+\epsilon)^2 d_{\min}, \ldots, d_{\max}$. For the query point $q$, we will perform binary search in the data structures for various values of $R$ to find the smallest value of $R$ for which we obtain an element $x \in P$ such that $x$ is a $(R,\epsilon)$-NN of $q$ (i.e., $f(q) = f(x)$). The binary search incurs an additional cost of $O(\log \frac{D_{\max}}{D_{\min}})$.

Now we turn our attention to answering $(R,\epsilon)$-NN queries. Pick a window size $w > 0$, where $w$ is real number. We will hash points in $P$ to buckets by the following procedure.

1. For $\alpha = 1, \ldots, b$ do
   (a) Pick $r = O(\log n)$ random vectors $v_1, v_2, \ldots, v_r$ in $\mathbb{R}^D$. The components $v_{ij}$, $j = 1, \ldots, D$ for the vectors $v_i$, $i = 1, \ldots, r$, are chosen independently from the standard normal distribution $N(0,1)$.
   (b) For each $v_i$, choose an offset $o_i \in (0, w]$ uniformly at random.
   (c) Each point $x \in P$ is mapped to the bucket $f_\alpha(x) = (\lceil \frac{\langle v_1, x\rangle + o_1}{w} \rceil, \ldots, \lceil \frac{\langle v_r, x\rangle + o_r}{w} \rceil)$. (Note that $\langle v_i, x\rangle$ represents the dot-product of $v_i$ and $x$.)

To answer the $(R,\epsilon)$-NN query for point $q \in \mathbb{R}^D$, we execute the following steps:

1. Compute $f_1(q), f_2(q), \ldots, f_b(q)$ to identify the buckets in which point $q$ falls.

2. Form a set Candidate$(q)$ by taking union of points of $P$ in buckets $f_1(q), f_2(q), \ldots, f_b(q)$.

3. For up to $2b$ elements of Candidate$(q)$, if there exists an element $x \in$ Candidate$(q)$ such that $||x - q||_2 \leq (1+\epsilon)R$, report $x$ as $(R, \epsilon)$-NN of $q$. Otherwise report NIL.

Before we proceed further, let us look at the vectors $v_i$'s chosen in the above procedure. Each component of these vectors are independent identical random variable from the distribution $N(0,1)$. Let $X_1, \ldots, X_r$ be i.i.d. random variables from $N(0,1)$. We are interested in understanding the distribution of the linear combination of $X_1, \ldots, X_r$.

**Claim 9.6.2** *Let $X = (X_1, \ldots, X_r)$, where each random variable $X_i$, $1 \leq i \leq r$, is from the standard normal distribution $N(0,1)$. Let $u = (u_1, \ldots, u_r)$, where each $u_i$ is a real number. The dot product $\langle u, X \rangle$ has the distribution $N(0, ||u||_2^2)$.*

**Proof.** $\mathbb{E}[\langle u, X \rangle] = \mathbb{E}[\sum_{i=1}^{r}(u_i X_i)] = \sum_i u_i \mathbb{E}[X_i] = 0$ by linearity of expectation. Note that for any random variable $Y$, its variance $Var(Y) = \mathbb{E}[Y^2] - \mathbb{E}[Y]^2$. Thus, $Var[\langle u, X \rangle] = \mathbb{E}[(\langle u, X \rangle)^2]$. Now $\mathbb{E}[(\langle u, X \rangle)^2] = \mathbb{E}[(u_1 X_1 + \cdots + u_r X_r)(u_1 X_1 + \cdots + u_r X_r)] = \sum_{i \neq j} u_i u_j \mathbb{E}[X_i X_j] + \sum_i u_i^2 \mathbb{E}[X_i^2] = \sum_{i \neq j} u_i u_j \mathbb{E}[X_i]\mathbb{E}[X_j] + \sum_i u_i^2 \mathbb{E}[X_i^2] = 0 + \sum_i u_i^2 = ||u||_2^2$, since (a) for $i \neq j$, $\mathbb{E}[X_i X_j] = \mathbb{E}[X_i]\mathbb{E}[X_j]$ as $X_i$ and $X_j$ are independent, and (b) $\mathbb{E}[X_i^2] = 1$, since $Var[X_i] = E[X_i^2] - E[X_i]^2 = 1$ as $X_i$ has $N(0,1)$ distribution. ∎

Let $v$ be one of the vectors with offset $o$ computed by the procedure where each of its component is i.i.d. random variable from the standard normal distribution $N(0,1)$. For a point $x \in P$ and a query point $q \in \mathbb{R}^D$, we are interested in finding the probability that both $x$ and $q$ will hash to the same bucket, i.e $Pr\left(\lfloor \frac{\langle v,q \rangle + o}{w} \rfloor = \lfloor \frac{\langle v,x \rangle + o}{w} \rfloor\right)$. Observe that $x$ and $q$ will hash into the same bucket of width $w$ if
(1) $|\langle v, q \rangle - \langle v, x \rangle| \leq w$, and
(2) the 'divider' does not fall between $\langle v, q \rangle$ and $\langle v, x \rangle$.

Note that $|\langle v, q \rangle - \langle v, x \rangle| \leq w$ is equivalent to $|\langle v, q - x \rangle| \leq w$. Let us apply Claim 9.6.2 to $\langle v, q - x \rangle$, where $v$ and $q - x$ play the roles of $X$ and $u$, respectively. By the claim we know that $\langle v, q - x \rangle$ has the distribution of $N(0, ||q - x||_2^2)$. Alternatively, we can say that $\langle v, q - x \rangle$ is a random variable $cZ$, where $c = ||q - x||_2$ and $Z$ has $\phi(z) = N(0,1)$ distribution. Thus the condition $|\langle v, q - x \rangle| \leq w$ can be written as $|cZ| \leq w$. The probability that the divider falls between $\langle v, q \rangle$ and $\langle v, x \rangle$ is $\frac{|\langle v, q-x \rangle|}{w}$. Thus, the probability that the divider does not fall between $\langle v, q \rangle$ and $\langle v, x \rangle$ is $1 - \frac{|\langle v, q-x \rangle|}{w}$.

Now the probability of collision

$$Pr(c) = Pr(\lfloor \frac{\langle v, q \rangle + o}{w} \rfloor = \lfloor \frac{\langle v, x \rangle + o}{w} \rfloor) = \int_{z=0}^{\leq \frac{w}{c}} \phi(z)(1 - \frac{cz}{w})dz. \tag{9.1}$$

Substituting $t = cz$, we obtain

$$Pr(c) = \int_{t=0}^{w} \frac{1}{c} \phi\Big(\frac{t}{c}\Big)\Big(1 - \frac{t}{w}\Big)dt. \tag{9.2}$$

Also it can be reasoned that the function $Pr(c)$ is monotonically decreasing in $c = ||x - q||_2$, i.e. the probability of collision decreases as the distance between $x$ and $q$ increases.

To answer the $(R, \epsilon)$-NN query, we consider two critical values of $c$, namely $c = R$ and $c = (1 + \epsilon)R$, in Equation 9.2 and let the corresponding probabilities be $p_1 = Pr(c = R)$ and $p_2 = Pr((c = 1 + \epsilon)R)$. We know that $p_1 > p_2$ from the monotonicity. We claim the following:

**Claim 9.6.3** *For a query point $q \in \mathbb{R}^D$, let $p^* \in P$ be a point such that $||q - p^*|| \leq R$. With respect to the above procedure:*

1. *For some $\alpha \in \{1, \ldots, b\}$, with probability $\geq 1/2$, $f_\alpha(p^*) = f_\alpha(q)$.*
2. *With probability $> 1/2$ the total number of elements $x \in P$ such that $f_\alpha(q) = f_\alpha(x)$ and $||x - q||_2 > (1 + \epsilon)R$ is at most $2b$.*

**Proof.** We prove the first statement. From Equation 9.2, we know that probability of collision between $p^*$ and $q$ is at least $p_1$. Therefore, for a fixed $\alpha$, $Pr[f_\alpha(p^*) = f_\alpha(q)] \geq p_1^r$. By setting $r = \log_{\frac{1}{p_2}} n$, we have

$$p_1^r = p_1^{\left(\log_{\frac{1}{p_2}} n\right)} = n^{-\left(\frac{\log \frac{1}{p_1}}{\log \frac{1}{p_2}}\right)} = n^{-\rho},$$

where $\rho = \frac{\log \frac{1}{p_1}}{\log \frac{1}{p_2}}$. From the banding technique, we know that the probability that the collision occurs in at least one of the $b$ bands is $1 - (1 - n^{-\rho})^b$. We set $b = n^\rho$. Then this probability can be expressed as

$$1 - (1 - n^{-\rho})^b = 1 - (1 - n^{-\rho})^{n^\rho} \geq 1 - \frac{1}{e} > \frac{1}{2}.$$

Next we prove the second part. Let $p \in P$ such that $||p - q|| > (1 + \epsilon)R$. We know from Equation 9.2, the probability of collision for a fixed $\alpha \in \{1, \ldots, b\}$ is $\leq p_2^r = p_2^{\left(\log_{\frac{1}{p_2}} n\right)} = \frac{1}{n}$. Since the set $P$ has $n$ elements, the expected number of collisions per band is at most 1, and the total number of the collisions is at most $b$. Thus the probability that the number of collisions exceeds more than $2b$ is at most $1/2$ from Markov's inequality. ∎

**Claim 9.6.4** *The queries can be answered in $O(n^\rho D \log n) = o(nD)$ time.*

**Proof.** To evaluate a query $q \in \mathbb{R}^D$, we need to compute $b = n^\rho$ hash functions $f(q)$. Each of them requires a computation of $r = O(\log n)$ quantities of the form $\lceil \frac{\langle v_i, q \rangle + o_i}{w} \rceil$. The dot product requires $O(D)$ time. Thus the total time required to hash $q$ is $O(n^\rho D \log n)$. Once we compute all the buckets in which $q$ lies, we need to evaluate the distance between $q$ and at most $2b$ elements of $P$. This requires a total of $O(n^\rho D)$ time. Thus the queries can be answered in $O(n^\rho D \log n)$ time. Since $p_1 > p_2$, $\rho = \frac{\log \frac{1}{p_1}}{\log \frac{1}{p_2}} < 1$. Hence the queries require $o(nD)$ time. ∎ Furthermore, notice that the dimension $D$ is not in the exponent, as in the Voronoi diagram based methods for finding nearest neighbors.

**Claim 9.6.5** *The procedure requires $O(n^{1+\rho} + Dn^\rho \log n)$ space to store the data structures.*

**Proof.** We need to store the non-empty buckets, i.e. partition of points in $P$, for each of the $b = n^\rho$ bands. Also, we need to store the parameters for each of the hash functions $f$. For each band, this includes $O(r)$ vectors $v$'s and offsets $o_i$'s. In all this requires $O(bn + rbD) = O(n^{1+\rho} + Dn^\rho \log n)$ memory space. ∎

The material of this subsection is adapted from the notes of L. Cayton.

### 9.6.5 *Fingerprint Matching*

Fingerprint matching typically requires comparison of several features of the fingerprint pattern. These include ridge lines which form arches, loops, or circular patterns, along with minutia points and patterns which form ridges, and bifurcations (see Figure 9.6). Typically each fingerprint is mapped to a normalized grid that takes care of the size and orientation of the fingerprint. After the normalization, it is expected that for two fingerprints of the same finger, if a grid cell of one fingerprint contains a minutia, the corresponding grid cell of the other fingerprint, with high probability, will contain that minutia. Therefore, we can abstract a fingerprint to be a set of grid points, where matching any two fingerprints amounts to matching elements in the corresponding grid cells.

Assume that the probability of finding a minutia in a random grid cell of any given fingerprint is 0.2. Also, assuming that we have two fingerprints of the same finger, let the probability of a minutia appearing in the same grid cell of both fingerprints given that one

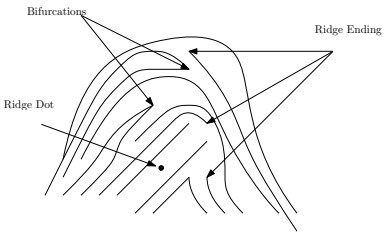



Figure 9.6: Miniutia in fingerprints.

of them does have a minutia there be 0.85. We will define a locality-sensitive family of functions $\mathcal{F}$ as follows. Each function $f \in \mathcal{F}$ sends two fingerprints to the same bucket if they have minutia in each of the three specific grid cells. Let us estimate the chance of ending up with two matching fingerprints. First, the probability that two arbitrary fingerprints will map to the same bucket with respect to the function $f$ is $0.2^6 = 0.000064$. Assuming that we have two fingerprints from the same finger, $f$ maps them to the same bucket with a probability of $0.2^3 \times 0.85^3 = 0.0049$. Note that the probability that the three particular cells of the first fingerprint contains minutia is $0.2^3$, and given that the two fingerprints are from the same same finger, the other fingerprint will have minutia in the same cells with probability $0.85^3$. Now we can use the OR-sensitive families to amplify these probabilities.

As an example, suppose we use 1000-way OR-functions. Then two fingerprints from different fingers will map to the same bucket with a probability of $1 - (1 - 0.000064)^{1000} \approx 0.061$. Similarly, two fingerprints from the same finger will map to the same bucket with a probability of $1 - (1 - 0.0049)^{1000} \approx 0.992$.

For another example, let us now use 2000 OR-functions that are partitioned in two groups of 1000 functions each. Assume also that we have constructed buckets for each group. Given a query fingerprint, we will find the fingerprints which are potential matches using the above scheme in each group independently. We select only those fingerprints which are potential matches in both the groups. This produces a set of fingerprints that we will actually compare against the query fingerprint. Note that in this scheme actual comparisons of fingerprints occur with only a few fingerprints (those in the intersection). Hence, the probability that we will detect matching fingerprints is $(0.992)^2 \approx 0.984$. The probability of false positives (the non-matches which we will detect after making the comparison with the query fingerprint) is $0.061^2 \approx 0.00371$, which is insignificant. Notice that in this scheme we have been able to avoid comparing the query fingerprint with all the fingerprints and with very high probability we will find matching fingerprints.

## 9.7 *Bibliographic Notes*

A method, based on shingling, for finding similar files in a large file system was proposed [69]. The $k$-shingles of a document and minhashing technique are formally introduced in [21, 20]. These concepts were developed as a result of the authors' work on the AltaVista web index algorithm which was used for detecting similar documents. Minwise-independent families also underpin the theory

behind minhashing, as the resemblance of document-sets is shown to be equal to the probability that the min-wise permutation of two sets using random permutation functions are equal.

Locality-sensitive hashing technique was introduced in [49] where the approximate nearest-neighbor search problem was first reduced to the problem of point location in equal balls. Along with this, the distance measures for Hamming distance and the resemblance measure given in [21] were used as schemes. It is worth noting that the approximate search is deemed accurate enough for practical purposes, where the actual closest neighbor can still be found by checking all approximate near-neighbors [5]. A scheme based on p-stable distributions ($p \in (0, 2]$) under the $L_p$ norm for locality-sensitive hashing was introduced in [26]. Further improvements are given in [5].

Locality-sensitive hashing is used in for video identification and retrieval. In this case, feature vectors are usually constructed from video frames using certain color histograms. In [54], the authors address two weaknesses of using LSH for this purpose. Responding to a non-uniform distribution of points in the space, they focus on using a hierarchical hash table to produce a more uniform distribution. In addition to this, they also attempt to partition dimensions more carefully in order to produce a more even hashing. A new scheme for video retrieval is then proposed in [48], where a color histogram is used which better handles the adverse effects of brightness & color variations. This is used in conjunction with an additional uniform distance shrinking projection which is applied to the produced feature vectors.

Locality-sensitive hashing is also used in image search. Similar to applications in video (for a single frame), images are often processed using color histograms to produce feature vectors which can be compared. This method was used in [37], with the histograms compared using the $L_1$ norm. Following this, techniques using $\chi^2$ distance [40], p-stable distributions [52], and kernelized locality-sensitive hashing [65] have been proposed. Kernelized locality-sensitive hashing has also been used as a basis for search on text images in [76].

LSH has recently been proposed for use in a wide range of other areas. One of these is the creation of hash values in P2P networks [27]. This would allow for a conceptual search, as data with similar ontologies would be located near each other. Here, data is defined by its extracted concept vectors, which are then hashed into buckets based on the cosine distance measure. Another, in [51], kernelized LSH is applied to an utterance model in order to identify speakers. In this case the Hamming distance metric is used. Other areas include use for species diversity estimation by allowing ease of grouping

similar DNA sequences [85], and incremental clustering for affinity groups in Hadoop in order to store related data closer together [53]. [94] has proposed a new scheme based on entropy for LSH. The argument is that an improvement can be made for [26] such that the distribution of mapped values will be approximately uniform. [2]

[2] Parts of this section are contributed by Andrew Wylie.

## 9.8 Exercises

**9.1** *Show that Euclidean metric satisfies triangle inequality.*

**9.2** *Given two documents* $D_1 = \{$*"His brow trembled"*$\}$ *and* $D_2 = \{$*"The brown emblem"*$\}$*, compute all k-shingles for each document with* $k = 4$*.*

**9.3** *Compute the Jaccard Similarity of the two sets of k-shingles for* $D_1$ *and* $D_2$*.*

**9.4** *Compute the signature matrix (minhash signature) of a given permutation of characteristic matrix in Table 9.7, using* $n = 3$ *different hash functions.*

| Element | $S_1$ | $S_2$ | $S_3$ | $S_4$ |
|---|---|---|---|---|
| $a$ | 1 | 0 | 1 | 1 |
| $b$ | 1 | 0 | 1 | 0 |
| $c$ | 0 | 1 | 0 | 1 |
| $d$ | 1 | 0 | 1 | 0 |
| $e$ | 0 | 0 | 1 | 0 |

Table 9.7: A characteristic matrix.

**9.5** *Explain the reasoning behind the proof of Lemma 9.2.1.*

**9.6** *Show that the minhash function family is* $(d_1, d_2, 1 - d_1, 1 - d_2)$*-sensitive.*

**9.7** *Calculate hamming distance for the vectors* $v = [5, 1, 3, 2, 4]$ *and* $u = [1, 1, 2, 2, 4]$*.*

**9.8** *Show that the AND amplification construction is* $(d_1, d_2, p_1^r, p_2^r)$*-sensitive.*

**9.9** *When applying amplification constructions to a locality-sensitive family of functions, which order of composition is 'better', and why? Explain when you would want to use different orders of construction.*

**9.10** *Show that the Jaccard Distance which is defined as* $1-$ *the Jaccard Similarity between the two sets is a metric.*

**9.11** *In Section 9.6.3 we considered the problem of computing LSH families for near neighbors in 2-dimensions. This exercise extends that solution to 3-dimensions. Let $P$ be a set of points in 3-dimensions. Consider a line $l$ with a random orientation that is partitioned in intervals of size $2\Delta$. Project points in $P$ orthogonally on $l$, and for a point $x \in P$, let $f_l(x)$ be the interval it projects on $l$. For any pair of points $x, y \in P$, show the following.*

1. *If $d(x,y) \leq \Delta$, $Pr[f_l(x) = f_l(y)] \geq 1/2$.*
2. *If $d(x,y) \geq 4\Delta$, $Pr[f_l(x) = f_l(y)] < 1/2$.*

**9.12** *Show that the function $1 - (1 - s^r)^b$ is non-decreasing with respect to $s$ for fixed values of $r$ and $b$.*

# 10
# *Dimensionality Reduction*

The material of this chapter is from Dasgupta and Gupta [25], Dubhashi & Panconesi [30], Matousek [71, 70], Bourgain [15], and the lecture notes of Harold Racke on a course on metric embeddings offered at IIIT Chicago a while ago. This chapter is under construction. I plan to cover the following.

## *10.1 Preliminaries: Metric spaces and embeddings*

We will define metric spaces, introduce the concept of metric embedding, introduce isometric and nonisometric embeddings via the concept of distortion - contraction and expansion. We show that any $n$-point metric space $\langle X, d\rangle$ can be embedded isometrically into $n$-dimensional space with $L_\infty$-norm. We also show that there are metric spaces which cannot be embedded isometrically always in the plane endowed with Euclidean distance norm.

**Definition 10.1.1** *Let $X$ be a set of $n$-points and let $d$ be a distance measure associated with pairs of elements in $X$. We say that $\langle X, d\rangle$ is a finite metric space if the function $d$ satisfies metric properties, i.e. (a) $\forall x \in X, d(x,x) = 0$, (b) $\forall x, y \in X, x \neq y, d(x,y) > 0$, (c) $\forall x, y \in X, d(x,y) = d(y,x)$ (symmetry), and (d) $\forall x, y, z \in X$, $d(x,y) \leq d(x,z) + d(z,y)$ (triangle inequality).*

**Definition 10.1.2** *Let $\langle X, d\rangle$ and $\langle X', d'\rangle$ be two metric spaces. A map $f : X \to X'$ is called an embedding. It is said to be isometric (i.e., distance preserving) if for all $x, y \in X$, $d(x,y) = d'(f(x), f(y))$. The contraction of $f$ is defined to be maximum factor by which the distances shrink and it equals $\max_{x,y \in X} \frac{d(x,y)}{d'(f(x),f(y))}$. Similarly the expansion of $f$ is the maximum factor by which the distances are stretched and it equals $\max_{x,y \in X} \frac{d'(f(x),f(y))}{d(x,y)}$. Distortion of an embedding is defined to be the product of its expansion and contraction factor. For an isometric embedding, its contraction factor is 1, its expansion factor is 1 and therefore its distortion is 1. There are other equivalent definitions of distortion.*

## 10.2 *Embeddings into $L_\infty$-normed spaces*

This section covers a theorem due to Matousek, see [71, 70], using the proof technique of Bourgain [15]. Let $\langle X, d\rangle$ be a metric space where $X$ is a set of $n$-points and let $d$ satisfies the metric properties. We show that each point in $X$ can be embedded in $k = O(Dn^{\frac{2}{D}} \log n)$-dimensional space such that the following holds. Let $x, y \in X$ and let $f(x), f(y)$ be their embedding in the $k$-dimensional space, respectively. We measure the distances in the $k$-dimensional space using the $L_\infty$-norm. We show that the distances gets distorted by a factor of at most $D \geq 1$. In fact in this case, the mapping is a contraction, and the maximum amount that the distances can shrink is at most $D$. Succinctly this is specified as

$$\langle X, d\rangle \overset{D}{\hookrightarrow} L_\infty^{k=O(Dn^{\frac{2}{D}} \log n)}.$$

For the special case, when $D = O(\log n)$, $k = O(\log^2 n)$, i.e. $\langle X, d\rangle \overset{O(\log n)}{\hookrightarrow} L_\infty^{O(\log^2 n)}$

One may wonder why should we even worry about these embeddings. There are several applications. One easy application is the following. For an $n$-element set $X$, to represent the the pair-wise distances, we will require $\Omega(n^2)$ space. Whereas, in the embedding, each point requires space proportional to the dimension, i.e. $O(k)$. Thus the total storage requirement is $O(kn)$. For example, when $D = O(\log n)$, the space required is $O(n \log n)$ instead of $O(n^2)$. Note that this is at the cost of replacing exact distances by approximate distances. Let us try to prove the following theorem in this section.

**Theorem 10.2.1** $\langle X, d\rangle \overset{D}{\hookrightarrow} L_\infty^{k=O(Dn^{\frac{2}{D}} \log n)}$.

The proof is constructive and leads to a randomized algorithm for finding an embedding. First we define the concept of a distance of a point $x \in X$ to a set $S \subseteq X$, as the distance of $x$ to the nearest point in $S$. More formally,

**Definition 10.2.2** *Let $S \subseteq X$. For $x \in X$, define $d(x, S) = \min_{z \in S} d(x, z)$.*

**Claim 10.2.3** *Let $x, y \in X$. For all $S \subseteq X$, $|d(x, S) - d(y, S)| \leq d(x, y)$.*

**Proof.** Proof uses triangle inequality. Let $x'$ be the point closest to $x$ in $S$. Similarly, let $y'$ be the point closest to $y$ in $S$. Then $d(x, S) = d(x, x')$ and $d(y, S) = d(y, y')$. Assume that $d(x, x') \geq d(y, y')$. Then

$$d(x, x') - d(y, y') \leq d(x, y') - d(y, y') \leq d(x, y).$$

Similarly, assume that $d(y, y') \geq d(x, x')$. Then

$$d(y, y') - d(x, x') \leq d(y, x') - d(x, x') \leq d(x, y).$$

Thus, $|d(x,x') - d(y,y')| \leq d(x,y)$. ∎

**Definition 10.2.4** *(**Mapping**) Let $x \in X$. Let $S_1, S_2, \cdots, S_k \subseteq X$. The mapping $f$ maps $x$ to the point*

$$f(x) = \{d(x,S_1), d(x,S_2), \cdots, d(x,S_k)\}.$$

**Claim 10.2.5** *Let $S_1, S_2, \cdots, S_k \subseteq X$. For $x, y \in X$, $||f(x) - f(y)||_\infty \leq d(x,y)$.*

**Proof.** Follows from the above claim, as for each $1 \leq i \leq k$, $|d(x,S_i) - d(y,S_i)| \leq d(x,y)$. ∎

Next we are going to construct a set of $k = O(Dm)$ subsets of $X$ by a simple randomized selection process, where $m = O(n^{\frac{2}{D}} \log n)$. For any pair of points $x, y \in X$, we will show that there is at least one subset in this set for which $||f(x) - f(y)||_\infty \geq \frac{d(x,y)}{D}$. By Claim 10.2.5, we know that $||f(x) - f(y)||_\infty \leq d(x,y)$. Thus, this construction will show that the mapping has a distortion of at most $D$. Let us first present Algorithm 10.1. This algorithm returns the required mapping for each point $x \in X$.

---

**Algorithm 10.1:** Compute a set of $O(Dm)$ subsets of $X$ and a mapping of each point $x \in X$

**Input:** Set $X$ consisting of $n$-elements and a distortion parameter $D \geq 1$

**Output:** A set of $O(Dm)$ subsets of $X$

```
p ← min(1/2, n^{-2/D})
m ← O(n^{2/D} log n)
for j ← 1 to ⌈D/2⌉ do
    for i ← 1 to m do
        Choose set S_ij by sampling each element of X
        independently with probability p^j
    end
end
For each x ∈ X return (f(x) = (d(x,S_11), ··· d(x,S_m1), d(x,S_12),
    ··· , d(x,S_m2), ··· , d(x,S_1⌈D/2⌉), ··· , d(x,S_m⌈D/2⌉)).
```

---

Before we show that the subsets produced by Algorithm 10.1 satisfies $||f(x) - f(y)||_\infty \geq \frac{d(x,y)}{D}$ for any pair of points $x, y \in X$, we make the following observation.

**Observation 10.2.6** *Let $x, y$ be two distinct points of $X$. Let $B(x,r)$ be the set of points of $X$ that are within a distance of $r$ from $x$ (think of $B(x,r)$*

*as a ball of radius r centred at x). Similarly, let $B(y, r+\Delta)$ be the set of points of X that are within a distance of $r+\Delta$ from y. Consider a subset $S \subset X$ such that $S \cap B(x,r) \neq \emptyset$ and $S \cap B(y, r+\Delta) = \emptyset$. Then $|d(x,S) - d(y,S)| \geq \Delta$.*

**Lemma 10.2.7** *Let $x, y$ be two distinct points of X. There exists an index $j \in \{1, \cdots, \lceil \frac{D}{2} \rceil\}$ such that if $S_{ij}$ is as chosen in Algorithm 10.1, than*

$$Pr\big[||f(x) - f(y)||_\infty \geq \frac{d(x,y)}{D}\big] \geq \frac{p}{12}.$$

First let us see why the above lemma implies Theorem 10.2.1. For a fix, $x, y \in X$, the probability that none of the $m$ trials for that particular $j$ are good has probability of at most $(1 - \frac{p}{12})^m \leq e^{-\frac{pm}{12}} \leq \frac{1}{n^2}$. Since there are in all $\binom{n}{2}$ pairs in $X$, the probability that we fail to choose a good set for any of the pairs, by the union bound, is strictly less than 1. Now let us prove Lemma 10.2.7.

**Proof.** (Proof of Lemma 10.2.7) Set $\Delta = \frac{d(x,y)}{D}$. For $i = 0, \cdots, \lceil \frac{D}{2} \rceil$, define balls of radius $i\Delta$ as follows. Let $B_0 = \{x\}$. Let $B_1$ be the ball of radius $\Delta$ centred at $y$. Then $B_2$ is the ball of radius $2\Delta$ centred at $x$. $B_3$ is the ball centred at $y$ of radius $3\Delta$ and so on. Hence, all balls with even $i$'s are centred at $x$ and at odd $i$'s are centred at $y$. Since $i \leq D/2$, no even ball overlaps with an odd balls. For even (odd) $i$, let $|B_i|$ denote the number of points of $X$ that are within a distance of at most $i\Delta$ from $x$ (respectively, $y$). Next we claim that, there is an index $t \in \{0, \cdots, \lceil \frac{D}{2} \rceil - 1\}$, such that $|B_t| \geq n^{\frac{2t}{D}}$ and $|B_{t+1}| \leq n^{\frac{2(t+1)}{D}}$.

**Claim 10.2.8** *There is an index $t \in \{0, \cdots, \lceil \frac{D}{2} \rceil - 1\}$, such that $|B_t| \geq n^{\frac{2t}{D}}$ and $|B_{t+1}| \leq n^{\frac{2(t+1)}{D}}$.*

**Proof.** Proof is by contradiction. $|B_0| = 1$ by construction. Thus $|B_1| > n^{\frac{2}{D}}$, otherwise the claim holds.

Since $|B_1| > n^{\frac{2}{D}}$, $|B_2| > n^{\frac{4}{D}}$, otherwise the claim holds.

Since $|B_2| > n^{\frac{4}{D}}$, $|B_3| > n^{\frac{6}{D}}$, otherwise the claim holds.

Continuing this way, for the last possible value of $t = \lceil \frac{D}{2} \rceil - 1$, we obtain $|B_{t+1}| > n^{\frac{2(t+1)}{D}} \geq n$. Since $|X| = n$, this is impossible, and hence there exists an index $t$ that satisfies the statement of the claim. ∎

Consider the index $t$ as stated in this claim. We have two balls, $B_t$ and $B_{t+1}$, such that $|B_t| \geq n^{\frac{2t}{D}}$ and $|B_{t+1}| \leq n^{\frac{2(t+1)}{D}}$. We will next show that for $j = t+1$, in Algorithm 10.1, the set $S_{ij}$ chosen by the algorithm will have a non-empty intersection with $B_t$ with probability at least $p/3$, and it will avoid $B_{t+1}$ with probability at least $1/4$. Formally, consider the following two events:

Let $E_1$ be the event that $S_{ij} \cap B_t \neq \emptyset$.

Let $E_2$ be the event that $S_{ij} \cap B_{t+1} = \emptyset$.
Let us first analyze $E_1$.

$$Pr[E_1] = 1 - Pr[S_{ij} \cap B_t = \emptyset] \tag{10.1}$$
$$= 1 - (1-p^j)^{|B_t|} \tag{10.2}$$
$$\geq 1 - (1-p^j)^{n^{\frac{2(j-1)}{D}}} \tag{10.3}$$
$$\geq 1 - e^{-p^j n^{\frac{2(j-1)}{D}}} \tag{10.4}$$
$$= 1 - e^{-p} \tag{10.5}$$

If $p \leq 1/2$, $1 - e^{-p} \geq p/3$.

Similarly,

$$Pr[E_2] = Pr[S_{ij} \cap B_{t+1} = \emptyset] \tag{10.6}$$
$$= (1-p^j)^{|B_{t+1}|} \tag{10.7}$$
$$\geq (1-p^j)^{n^{\frac{2j}{D}}} \tag{10.8}$$
$$= (1-p^j)^{\frac{1}{p^j}} \tag{10.9}$$

If $p^j \leq 1/2$, $(1-p^j)^{\frac{1}{p^j}} \geq 1/4$. Note that the function $(1-p^j)^{\frac{1}{p^j}}$ achieves its minimum value in the interval $0 \leq p^j \leq 1/2$, at the ends of the interval, i.e. at $p^j = 0$ or at $p^j = 1/2$. At both the extremes, its value is at least $1/4$.

Next we need to estimate what is the $Pr[E_1 \wedge E_2]$. Note that events $E_1$ and $E_2$ are independent as the balls, $B_t$ and $B_{t+1}$ are disjoint. Thus $Pr[E_1 \wedge E_2] \geq p/12$. ∎

By setting $D = \Theta(\log n)$, in Theorem 10.2.1, we obtain the following corollary.

**Corollary 10.2.9** $\langle X, d \rangle \overset{\Theta(\log n)}{\hookrightarrow} L_\infty^{O(\log^2 n)}$.

Next we show the following

**Lemma 10.2.10** $\langle X, d \rangle \overset{\log^2 n}{\hookrightarrow} L_1^{O(\log^2 n)}$.

**Proof.** Let $k = O(\log^2 n)$ be the dimension of embedding in Corollary 10.2.9. Observe that for the same embedding as in Corollary 10.2.9, for a pair of points $x, y \in X$, we have

$$||f(x) - f(y)||_1 \leq kd(x,y).$$

In the proof of Theorem 10.2.1, for a pair $x, y \in X$, we know that there is at least one set (as constructed in Algorithm 10.1) which is good, i.e., with probability at least $1 - 1/n^2$ for which $||f(x) -$

$f(y)||_\infty \geq \frac{d(x,y)}{\Theta(\log n)}$. We can extend the machinery in the Theorem to show that with high probability there are $\log n$ sets which are good. This will require to choose slightly larger value for $m$, but still of order of $O(\log n)$. If this is the case, then

$$||f(x) - f(y)||_1 \geq \log n \frac{d(x,y)}{\Theta(\log n)} = \Theta(d(x,y)).$$

Thus we have

$$\Theta(d(x,y)) \leq ||f(x) - f(y)||_1 \leq kd(x,y),$$

and hence we have a mapping with distortion $O(\log^2 n)$. ∎

Next we show,

**Corollary 10.2.11** $\langle X, d \rangle \overset{\log^{1.5} n}{\hookrightarrow} L_2^{O(\log^2 n)}$.

**Proof.** Let $k = O(\log^2 n)$ be the dimension of embedding in Corollary 10.2.9. Observe that for the same embedding as in Corollary 10.2.9, for a pair of points $x, y \in X$, we have

$$||f(x) - f(y)||_2 = \sqrt{\sum (d(x, S_{ij}) - d(y, S_{ij}))^2} \leq \sqrt{k} d(x,y).$$

With similar arguments as in the proof of Lemma 10.2.10,

$$||f(x) - f(y)||_2 = \sqrt{\sum (d(x, S_{ij}) - d(y, S_{ij}))^2} \geq \sqrt{\log n (\frac{d(x,y)}{\Theta(\log n)})^2} \geq \frac{d(x,y)}{\Theta(\sqrt{\log n})}.$$

This results in a total distortion of $O(\log^{1.5} n)$. ∎

## 10.3 *Embeddings into $L_p$-normed spaces*

This section covers the results of Bourgain [15]. We show that $\langle X, d \rangle \overset{O(\log n)}{\hookrightarrow} L_p^{O(\log^2 n)}$.

## 10.4 *Johnson and Lindenstrauss Theorem*

In this section we prove a celebrated theorem of Johnson and Lindenstrauss from 1984.

**Theorem 10.4.1** *Let $V$ be a set of $n$ points in $d$-dimensions. A mapping $f : \mathbb{R}^d \to \mathbb{R}^k$ can be computed, in randomized polynomial time, so that for all pairs of points $u, v \in V$,*

$$(1 - \epsilon)||u - v||^2 \leq ||f(u) - f(v)||^2 \leq (1 + \epsilon)||u - v||^2,$$

*where $0 < \epsilon < 1$ and $n, d$, and $k \geq 4(\frac{\epsilon^2}{2} - \frac{\epsilon^3}{3})^{-1} \ln n$ are positive integers.*

Intuitively, the theorem says that points in $d$-dimensional space can be mapped to a $k$ dimensional space, where $k$ is significantly less than $d$, and the interpoint Euclidean distance between a pair of points in $V$ is preserved up to a factor of $(1 \pm \epsilon)$. The proof is based on showing that the squared length of a random vector is concentrated around its mean when the vector is projected onto a random $k$-dimensional subspace, i.e., with probability $O(1/n^2)$, the length is within $(1 \pm \epsilon)$ of the mean. In place of estimating the length of a unit vector after projecting it onto a random $k$-dimensional space, we will estimate the length of a random unit vector when we project it to a fixed $k$-dimensional space (e.g. in the space spanned by the first $k$-coordinates).

Recall Standardized Normal Distributions $\mathcal{N}(0,1)$.

**Lemma 10.4.2** *Let $X_1, \ldots, X_d$ be $d$-independent standardized normal $\mathcal{N}(0,1)$ random variables. Let $X = (X_1, X_2, \ldots, X_d)$ and define $Y = \frac{1}{||X||} X$. Then, $Y$ is a random point drawn uniformly from the surface of the $d$-dimensional unit sphere.*

**Proof.** Since the distribution for each $X_i$ is given by $f(x_i) = \frac{1}{\sqrt{2\pi}} e^{-x_i^2/2}$, the distribution of $X$, due to Independence, is $\Pi_{i=1}^{d} [\frac{1}{\sqrt{2\pi}} e^{-x_i^2/2}] = \frac{1}{2\pi^{d/2}} e^{-\frac{1}{2}\sum_{i=1}^{d} x_i^2}$. Observe that the density function of $X$ only depends upon the distance of $(X_1, \ldots, X_d)$ from the origin, and thus it is spherically symmetric. Since $Y$ is obtained by dividing this by $||X||$, $Y$ is a random point on the unit sphere. ∎

From now on let $Y = (Y_1, \ldots, Y_d)$ be a random unit vector as stated in Lemma 10.4.2. Define a vector $Z \in \mathbb{R}^K$, which is obtained by projecting $Y$ onto its first $k$-coordinates (i.e. ignoring all other coordinates, except the first $k$). Let $L$ represent the squared length of $Z$, i.e. $L = ||Z||^2$.

**Lemma 10.4.3** $E[L] = k/d$.

**Proof.**

$$E[L] = E[||Z^2||] = \frac{1}{||X||^2} E[X_1^2 + \cdots + X_k^2] = \frac{1}{||X||^2} \sum_{1=1}^{k} E[X_i^2] = k \frac{E[X_1^2]}{||X||^2}.$$

Observe that

$$||Y||^2 = 1 = \frac{1}{||X||^2} \sum_{i=1}^{d} X_i^2.$$

This implies that

$$d \frac{E[X_1^2]}{||X||^2} = 1.$$

Hence

$$E[L] = k \frac{E[X_1^2]}{||X||^2} = \frac{k}{d}.$$

■

Next we state the main lemma, first without proving it, and show how Theorem 10.4.1 follows from it.

**Lemma 10.4.4** *Let $k < d$. Then*

1. *If $\beta < 1$, then $P(L \leq \beta\frac{k}{d}) \leq e^{\frac{k}{2}(1-\beta+\ln\beta)}$.*
2. *If $\beta > 1$, then $P(L \geq \beta\frac{k}{d}) \leq e^{\frac{k}{2}(1-\beta+\ln\beta)}$.*

**Proof.** (Proof of Theorem 10.4.1) The case of $k \geq d$ is trivial. Therefore, we assume that $k < d$. Let $S$ be a random $k$-dimensional subspace of $\mathbb{R}^d$ (for example we choose randomly $k$ coordinates). Project each point $v \in V$ and let its projection in $\mathbb{R}^k$ be $v'$. Let $Y := \frac{v-w}{||v-w||}$ be a unit vector, and let $Z := \frac{v'-w'}{||v-w||}$ be its random projection. Let $L = ||Z||^2$. Note that $Z$ behaves as required in Lemma 10.4.3. Hence $E[L] = k/d$. For $0 < \epsilon < 1$, set $\beta = 1 - \epsilon$. Note that

$$
\begin{aligned}
P(||v'-w'||^2 \leq (1-\epsilon)||v-w||^2 k/d) &= P(L \leq (1-\epsilon)k/d) && (10.10)\\
&\leq e^{\frac{k}{2}(1-(1-\epsilon)+\ln(1-\epsilon))} && (10.11)\\
&\leq e^{\frac{k}{2}(\epsilon-(\epsilon+\frac{\epsilon^2}{2}))} && (10.12)\\
&= e^{-\frac{k\epsilon^2}{4}} && (10.13)\\
&\leq e^{-2\ln n} && (10.14)\\
&= \frac{1}{n^2} && (10.15)
\end{aligned}
$$

Note that for Equation 10.12 we used that $\ln(1-\epsilon) \leq -\epsilon - \epsilon^2/2$, for $0 < \epsilon < 1$. For Equation 10.14 (and 10.20), we use the fact that $k \geq 4(\frac{\epsilon^2}{2} - \frac{\epsilon^3}{3})^{-1}\ln n$.

$$
\begin{aligned}
P(||v'-w'||^2 \geq (1+\epsilon)||v-w||^2 k/d) &= P(L \geq (1+\epsilon)k/d) && (10.16)\\
&\leq e^{\frac{k}{2}(1-(1+\epsilon)+\ln(1+\epsilon))} && (10.17)\\
&\leq e^{\frac{k}{2}(-\epsilon+(\epsilon-\frac{\epsilon^2}{2}+\frac{\epsilon^3}{3}))} && (10.18)\\
&= e^{-\frac{k(\epsilon^2/2-\epsilon^3/3)}{2}} && (10.19)\\
&\leq e^{-2\ln n} && (10.20)\\
&= \frac{1}{n^2} && (10.21)
\end{aligned}
$$

Set the map of each $v \in V$ as $f(v) = \sqrt{\frac{d}{k}}v'$. Consider a pair of points $u, v \in V$, and by above calculations, the probability that $(1-\epsilon)||u-v||^2 \leq ||f(u)-f(v)||^2 \leq (1+\epsilon)||u-v||^2$ does not

hold is at most $2/n^2$. Thus, for any pair of points, the probability this condition is not true is at most $\binom{n}{2}\frac{2}{n} = 1 - \frac{1}{n}$. Therefore, the mapping $f$ has the desired distortion property. We can boost the success probability further by repeating the experiment several times. Observe that as the mapping is fairly straighforward and can be easily seen to be obtained in polynomial time. ∎

Before we prove Lemma 10.4.4, we need the following.

**Lemma 10.4.5** *Let $X$ be $\mathcal{N}(0,1)$. Then $E[e^{sX^2}] = \frac{1}{\sqrt{1-2s}}$, for $s < 1/2$.*

**Proof.** Note that $X$ is given by the probability distribution $f(x) = \frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}$. Moreover, expected value of a function, say $H(X)$, of a random variable $X$ is given by $E(H(X)) = \int_{-\infty}^{+\infty} H(x)f(x)dx$. Thus,

$$E[e^{sX^2}] = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty} e^{sx^2}e^{-\frac{x^2}{2}}dx \tag{10.22}$$

$$= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty} e^{-(1-2s)x^2/2}dx \tag{10.23}$$

$$\text{set } y = x\sqrt{1-2s} \tag{10.24}$$

$$= \frac{1}{\sqrt{1-2s}}\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty} e^{-y^2/2}dy \tag{10.25}$$

$$= \frac{1}{\sqrt{1-2s}} \text{ (since } \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty} e^{-y^2/2}dy = 1) \tag{10.26}$$

∎

**Proof.**(Proof of Lemma 10.4.4) First, we want to prove that for $\beta < 1$, $P(L \leq \beta\frac{k}{d}) \leq e^{\frac{k}{2}(1-\beta+\ln\beta)}$.

Equivalently, we want to show that

$$P(d(X_1^2 + \cdots + X_k^2) \leq k\beta(X_1^2 + \cdots + X_d^2)) \leq e^{\frac{k}{2}(1-\beta+\ln\beta)}.$$

Note that $P(d(X_1^2 + \cdots + X_k^2) \leq k\beta(X_1^2 + \cdots + X_d^2))$

$$= P(k\beta(X_1^2 + \cdots + X_d^2) - d(X_1^2 + \cdots + X_k^2) \geq 0) \tag{10.27}$$

$$= P(e^{t(k\beta(X_1^2+\cdots+X_d^2)-d(X_1^2+\cdots+X_k^2)} \geq e^{t.0} = 1) \text{ (for } t > 0) \tag{10.28}$$

$$\leq E[e^{t(k\beta(X_1^2+\cdots+X_d^2)-d(X_1^2+\cdots+X_k^2)}] \text{ (Using Markov's inequality)} \tag{10.29}$$

Recall that if $A$ and $B$ are independent r.v. than $E[AB] = E[A]E[B]$. Moreover, $E[A^2] = E[A]^2$. Also, each of these $X_i$'s have identical distribution, and they are independent r.v.'s. After rearranging and

replacing all $X_i$'s by $X_1$, we obtain

$$= E[e^{tk\beta X_1^2}]^{d-k} E[e^{t(k\beta-d)X_1^2}]^k \tag{10.30}$$

$$= \left(\frac{1}{\sqrt{1-2tk\beta}}\right)^{d-k} \left(\frac{1}{\sqrt{1-2t(k\beta-d)}}\right)^{k} \text{ (Using Lemma 10.4.5)} \tag{10.31}$$

The above inequality holds, provided $0 < t < 1/2k\beta$. Now we want to minimize the quantity in Inequality 10.31. This is equivalent to maximizing

$$q(t) = (1-2tk\beta)^{d-k}(1-2t(k\beta-d))^k.$$

We obtain, using derivatives of a product etc., that maximum occurs when $t = \frac{1-\beta}{2\beta(d-k\beta)}$. The value of $q(t = \frac{1-\beta}{2\beta(d-k\beta)}) = \left(\frac{d-k}{d-k\beta}\right)^{d-k}\left(\frac{1}{\beta}\right)^k$. Substituting this back in Inequality 10.31, as $1/\sqrt{q(t)}$, we obtain that $P(L \leq \beta\frac{k}{d}) \leq \beta^{\frac{k}{2}}\left(1+\frac{(1-\beta)k}{(d-k)}\right)^{(d-k)/2}$. With simple manipulation and using the fact that $\ln(1+x) \leq x$, we obtain that $\beta^{\frac{k}{2}}\left(1+\frac{(1-\beta)k}{(d-k)}\right)^{(d-k)/2} \leq e^{\frac{k}{2}(1-\beta+\ln\beta)}$, as desired. One can show this by taking natural log on both the sides.

Next, using the identical method as above, we show that for $\beta > 1$, $P(L \geq \beta\frac{k}{d}) \leq e^{\frac{k}{2}(1-\beta+\ln\beta)}$. This amounts to showing that

$$P(d(X_1^2+\cdots+X_k^2) \geq k\beta(X_1^2+\cdots+X_d^2)) \leq \left(\frac{1}{\sqrt{1+2tk\beta}}\right)^{d-k}\left(\frac{1}{\sqrt{1+2t(k\beta-d)}}\right)^{k} \tag{10.32}$$

$$= \frac{1}{\sqrt{q(-t)}} \tag{10.33}$$

Now the rest of the proof requires finding the value that maximizes $q(-t)$. This is maximized at $t = -\frac{1-\beta}{2\beta(d-k\beta)}$. Rest of the proof is same. ∎

## 10.5 *Applications of Metric Embeddings*

## 10.6 *Exercises*

**10.1** *Show that for any embedding $f$ of a metric space $(X,d)$ to another metric space $(X',d')$, the distortion of $f$ is at least* 1.

# 11
# *Second moment method with applications*

Main part of this chapter is a copy of the article published in ALGOSENSORS 2015 titled Plane and Planarity Thresholds for Random Geometric Graphs by Ahmad Biniaz, Evangelos Kranakis, Anil Maheshwari and Michiel Smid [1]. The material for Cliques in $G(n, 1/2)$ is adapted from Nikal Bansal's class notes as well as from [2]. Over time this will be refined to give a broader overview of the second moment method and some more applications and exercises will be added.

[1] Ahmad Biniaz, Evangelos Kranakis, Anil Maheshwari, and Michiel Smid. Plane and planarity thresholds for random geometric graphs. In *Proc. ALGOSENSORS 2015 (Patras, Greece)*, Lecture Notes in Computer Science, Berlin, Germany, 2015. Springer

[2] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005

## 11.1 *Preliminaries*

Recall basic probability definitions from Chapter 1. The *variance* of a r.v. $X$ is defined to be $V[X] = E[(X-\mu)^2]$, where $\mu = E[X]$. Now consider $E[(X-\mu)^2] = E[X^2 - 2\mu X + \mu^2] = E[X^2] - 2\mu E[X] + E[\mu^2] = E[X^2] - E[X]^2$. Thus, $V[X] = E[(X-\mu)^2] = E[X^2] - E[X]^2$. Moreover, for two independent random variables, $X$ and $Y$, $Var[X+Y] = Var[X] + Var[Y]$. Markov's inequality states that for a non-negative discrete r.v. $X$ and $t > 0$ a constant, $P(X \geq t) \leq E[X]/t$. Furthermore, Chebyshev's inequality states that $Pr(|X-\mu| \geq t\sqrt{Var[X]}) \leq \frac{1}{t^2}$.

Let us toss a fair coin $n$ times and estimate what is the probability of getting $\frac{3}{4}n$ heads? Let $X_i$ be a 0-1 r.v. indicating the outcome of the $i$-th toss, where $X_i = 1$ ($X_i = 0$) indicates that the outcome is a head (tail). Let $X = \sum_{i=1}^{n} X_i$ and $X$ represents the total number of heads in $n$-tosses. Observe that $E[X_i] = 1/2$ and $V[X_i] = E[{X_i}^2] - E[X_i]^2 = 1/2 * 1^2 + 1/2 * 0^2 - (1/2)^2 = 1/4$. Thus $E[X] = n/2$ and since $X_i$'s are independent, $V[X] = n/4$. To estimate $Pr[X \geq \frac{3}{4}n]$ using Markov's inequality, set $t = 3/2$ and it results in $Pr[X \geq 3/2 * n/2] \leq 2/3$. To estimate it using Chebyshev's inequality, $Pr(|X-\mu| \geq t\sqrt{Var[X]}) \leq \frac{1}{t^2}$, we need to set $t = \sqrt{n/4}$ to obtain $Pr(|X - n/2| \geq n/4) \leq 4/n$.

We will use the following result for the second moment method.

**Theorem 11.1.1** *If $X \geq 0$ is a r.v. taking integer values, $Pr[X = 0] \leq \frac{Var[X]}{E[X]^2}$. Moreover, if $Var[X] = o(E[X]^2)$ for large values of $n$, $Pr[X = 0] =$*

$o(1)$.

**Proof.** We will use Chebyshev's inequality. Observe that $Pr[X = 0] \leq Pr[|X - E[X]| \geq E[X]]$. By substituting $t = \frac{E[X]}{\sqrt{Var[X]}}$ in Chebyshev's inequality $Pr(|X - E[X]| \geq t\sqrt{Var[X]}) \leq \frac{1}{t^2}$, we obtain $Pr[X = 0] \leq \frac{Var[X]}{E[X]^2}$. If $Var[X] = o(E[X]^2)$ for large values of $n$, $Pr[X = 0] = \lim_{n\to\infty} \frac{Var[X]}{E[X]^2} \to 0$, or in other words $Pr[X = 0] = o(1)$. ■

We will see applications of this theorem in this chapter.

## 11.2 Cliques in a random graph

Consider a random graph $G(n, p = 1/2)$. Note that $G(n, p)$ is a graph on $n$ vertices where each edge between a pair of vertices occurs (independently of other edges) with probability $p$. We will provide an estimate on the size of the largest clique in $G(n, 1/2)$ using the second moment method. In particular we will prove that

**Theorem 11.2.1** *Let $\omega(G)$ denote the size of the largest clique in a graph $G$. Given an $\epsilon > 0$, (a) $Pr[\omega(G(n, 1/2)) > (2 + \epsilon) \log n] = o(1)$ and (b) $Pr[(2 - \epsilon) \log n \leq \omega(G(n, 1/2)) \leq (2 + \epsilon) \log n] = 1 - o(1)$.*

To prove the first part of this theorem, we will estimate how many subsets of $k$ vertices in $G(n, 1/2)$ will form a clique. Fix a subset $S$ of $k$ vertices and let $X_S$ be a 0-1 indicator r.v. indicating whether $S$ is a clique or not. If $S$ is a clique then $X_S = 1$, otherwise $X_S = 0$. Since each edge in $S$ is chosen independent of other edges, $Pr[X_S = 1] = (\frac{1}{2})^{\binom{k}{2}}$ and $E[X_S] = Pr[X_S = 1]$. Let $X$ denote the total number of cliques of size $k$ in $G(n, 1/2)$ and let $V$ be the set of vertices in $G(n, 1/2)$. Observe that

$$E[X] = \sum_{S \subset V : |S| = k} E[X_S] = \binom{n}{k} \left(\frac{1}{2}\right)^{\binom{k}{2}} \leq \left(\frac{\sqrt{2}n}{2^{k/2}}\right)^k.$$

By setting $k = (2 + \epsilon) \log n$ as in Theorem, we observe that $\frac{\sqrt{2}n}{2^{k/2}} \leq \sqrt{2}n^{-\epsilon/2}$. For large values of $n$, $n^{-\epsilon/2} < 1$. Also $(n^{-\epsilon/2})^k$ is $o(1)$ (as $n$ increases, $k = (2 + \epsilon) \log n$ also increases). Thus by Markov's inequality $Pr[X \geq 1] \leq E[X] = o(1)$, i.e. the probability that there is a clique on $k$ vertices in $G(n, 1/2)$ is very small for large values of $n$.

Next we show the second part of Theorem 11.2.1 using the Second Moment Method. Let $X_S$ and $X$ be as before and let $k = (2 - \epsilon) \log n$. Using this value of $k$, it is easy to see that for large values of $n$, $E[X] = \binom{n}{k}\left(\frac{1}{2}\right)^{\binom{k}{2}} \to \infty$. Therefore, we will show that $Var[X] = o(E[X]^2)$ and then use Theorem 11.1.1 to conclude that the probability that there is no clique is $o(1)$ and hence the probability that there

is a clique is $1 - o(1)$. Let us first compute $Var[X] = E[X^2] - E[X]^2$. Note that

$$E[X^2] = E[\sum_{S \subset V:|S|=k} \sum_{T \subset V:|T|=k} X_S X_T] = \sum_{S \subset V:|S|=k} \sum_{T \subset V:|T|=k} E[X_S X_T].$$

Moreover, $E[X]^2 = E[X]E[X] = \sum_{S \subset V:|S|=k} E[X_S] \sum_{T \subset V:|T|=k} E[X_T]$. Now consider two sets of $k$ vertices $S$ and $T$. If they have fewer than two vertices in common than whether $S$ is a clique or not has no influence on whether $T$ is a clique or not. Therefore, for $|S \cap T| < 2$, $E[X_S X_T] = E[X_S]E[X_T]$. Thus we need to consider

$$Var[X] = \sum_{S \subset V:|S|=k} \sum_{T \subset V:|T|=k} E[X_S X_T] - E[X_S]E[X_T] \tag{11.1}$$

only for those pairs of $S$ and $T$ such that $|S \cap T| \geq 2$, since for the other values $E[X_S X_T]$ cancels $E[X_S]E[X_T]$. Thus, Equation 11.1 can be rewritten as

$$Var[X] = \sum_{S \subset V:|S|=k} \sum_{l=2}^{k} \sum_{T \subset V:|T|=k, |S \cap T|=l} E[X_S X_T] - E[X_S]E[X_T]. \tag{11.2}$$

Since, $E[X_S]E[X_T]$ is non-negative, we can obtain the following inequality from Equation 11.2.

$$Var[X] \leq \sum_{S \subset V:|S|=k} \sum_{l=2}^{k} \sum_{T \subset V:|T|=k, |S \cap T|=l} E[X_S X_T]. \tag{11.3}$$

Now observe that there are $\binom{n}{k}$ possibilities for choosing vertices for forming the set $S$ of size $k$, there are $\binom{k}{l}$ possibilities for selecting $l$ vertices that are common between $S$ and $T$ and there are $\binom{n-k}{k-l}$ possibilities for choosing the remaining vertices in $T \setminus S$. Hence,

$$Var[X] \leq \sum_{l=2}^{k} \binom{n}{k} \binom{k}{l} \binom{n-k}{k-l} E[X_S X_T]. \tag{11.4}$$

Let us compute $E[X_S X_T]$. Since $X_S$ and $X_T$ are 0-1 r.v., $E[X_S X_T]$ is same as the probability that $S$ is a clique and $T$ is a clique, or in other words, all pairs of vertices in $S$ are connected and all pairs of vertices in $T$ are connected. In all there are $k$ vertices in $S$ and $k - l$ vertices in $T$. The total number of edges in $S \cup T$ is $2\binom{k}{2} - \binom{l}{2}$. Thus,

$$E[X_S X_T] = \frac{1}{2}^{2\binom{k}{2} - \binom{l}{2}} \tag{11.5}$$

Substituting expression for $E[X_S X_T]$ in Equation 11.4, we obtain

$$Var[X] \leq \sum_{l=2}^{k} \binom{n}{k} \binom{k}{l} \binom{n-k}{k-l} 2^{\binom{l}{2} - 2\binom{k}{2}}. \tag{11.6}$$

Next we show that $Var[X]/E[X]^2$ is $o(1)$. Note that $E[X] = \binom{n}{k}2^{-\binom{k}{2}}$. Define

$$f(l) = \frac{\binom{n}{k}\binom{k}{l}\binom{n-k}{k-l}2^{\binom{l}{2}-2*\binom{k}{2}}}{(\binom{n}{k}2^{-\binom{k}{2}})^2}.$$

This can be rewritten as

$$f(l) = \frac{\binom{k}{l}\binom{n-k}{k-l}2^{\binom{l}{2}}}{\binom{n}{k}}.$$

Thus,

$$\frac{Var[X]}{E[X]^2} \leq \sum_{l=2}^{k} f(l). \tag{11.7}$$

Consider

$$\begin{aligned} f(2) &= \frac{\binom{k}{2}\binom{n-k}{k-2}2^{\binom{2}{2}}}{\binom{n}{k}} \\ &= \frac{k(k-1)\binom{n-k}{k-2}}{\binom{n}{k}} \end{aligned}$$

Since $k = (2+\epsilon)\log n << n$, it can be shown that $f(2) = o(1)$. Similarly, $f(k) = o(1)$. It seems that (but I don't know how to show this in a simple way) that $f(l) = o(1)$ for $l = 2, \cdots, k$. This concludes the proof of Theorem 11.2.1.

## 11.3 *Thresholds for Random Geometric Graphs*

Wireless networks are usually modelled as disk graphs in the plane. Given a set $P$ of points in the plane and a positive parameter $r$, the *disk graph* is the geometric graph with vertex set $P$ which has a straight-line edge between two points $p, q \in P$ if and only if $|pq| \leq r$, where $|pq|$ denotes the Euclidean distance between $p$ and $q$. If $r = 1$, then the disk graph is referred to as *unit disk graph*. A *random geometric graph*, denoted by $G(n, r)$, is a geometric graph formed by choosing $n$ points independently and uniformly at random in a unit square; two points are connected by a straight-line edge if and only if they are at Euclidean distance at most $r$, where $r = r(n)$ is a function of $n$ and $r \to 0$ as $n \to \infty$.

We say that two line segments in the plane *cross* each other if they have a point in common that is interior to both edges. Two line segments are *non-crossing* if they do not cross. Note that two non-crossing line segments may share an endpoint. A geometric graph is said to be *plane* if its edges do not cross, and *non-plane*, otherwise. A graph is *planar* if and only if it does not contain $K_5$ (the complete graph on 5 vertices) or $K_{3,3}$ (the complete bipartite graph

on six vertices partitioned into two parts each of size 3) as a minor. A *non-planar graph* is a graph which is not planar.

A graph property $\mathcal{P}$ is *increasing* if a graph $G$ satisfies $\mathcal{P}$, then by adding edges to $G$, the property $\mathcal{P}$ remains valid in $G$. Similarly, $\mathcal{P}$ is *decreasing* if a graph $G$ satisfies $\mathcal{P}$, then by removing edges from $G$, the property $\mathcal{P}$ remains valid in $G$. $\mathcal{P}$ is called a *monotone* property if $\mathcal{P}$ is either increasing or decreasing. Connectivity and "having a clique of size $k$" are increasing monotone properties, while planarity and "being plane" are decreasing monotone properties in $G(n,r)$, where the value of $r$ increases.

By [3] any monotone property of a random geometric graphs has a threshold function. The thresholds in random geometric graphs are expressed by the distance $r$. In the sequel, the term w.h.p. (with high probability) is to be interpreted to mean that the probability tends to 1 as $n \to \infty$. For an increasing property $\mathcal{P}$, the threshold is a function $t(n)$ such that if $r = o(t(n))$ then w.h.p. $\mathcal{P}$ does not hold in $G(n,r)$, and if $r = \omega(t(n))$ then w.h.p. $\mathcal{P}$ holds in $G(n,r)$. Symmetrically, for a decreasing property $\mathcal{P}$, the threshold is a function $t(n)$ such that if $r = o(t(n))$ then w.h.p. $\mathcal{P}$ holds in $G(n,r)$, and if $r = \omega(t(n))$ then w.h.p. $\mathcal{P}$ does not hold in $G(n,r)$. Note that a threshold function may not be unique. It is well known that $\sqrt{\ln n/n}$ is a connectivity threshold for $G(n,r)$; see [4]. Here we investigate thresholds in random geometric graphs for having a connected subgraph of constant size, being plane, and being planar.

**Related Work:** Random graphs were first defined and formally studied by Gilbert in [5] and Erdös and Rényi [6]. It seems that the concept of a random geometric graph was first formally suggested by Gilbert in [7] and for that reason is also known as Gilbert's disk model. These classes of graphs are known to have numerous applications as a model for studying communication primitives (broadcasting, routing, etc.) and topology control (connectivity, coverage, etc.) in idealized wireless sensor networks as well as extensive utility in theoretical computer science and many fields of the mathematical sciences.

An instance of Erdös-Rényi graph [8] is obtained by taking $n$ vertices and connecting any two with probability $p$, independently of all other pairs; the graph derived by this scheme is denoted by $G_{n,p}$. In $G_{n,p}$ the threshold is expressed by the edge existence probability $p$, while in $G(n,r)$ the threshold is expressed in terms of $r$. In both random graphs and random geometric graphs, property thresholds are of great interest [9]. Note that edge crossing configurations in $G(n,r)$ have a geometric nature, and as such, have no analogues in the context of the Erdös-Rényi model for random graphs. However, planarity, and having a clique of specific size are of interest in both

[3] A. Goel, S. Rai, and B. Krishnamachari. Sharp thresholds for monotone properties in random geometric graphs. In *Proceedings of STOC*, pages 580–586. ACM, 2004

[4] Piyush Gupta and P. R. Kumar. Critical power for asymptotic connectivity in wireless networks. In *Stochastic Analysis, Control, Optimization and Applications*, pages 547–566, 1998; Padma Panchapakesan and D Manjunath. On the transmission range in dense ad hoc radio networks. In *Proceedings of IEEE Signal Processing Communication (SPCOM)*, 2001; and Mathew D. Penrose. The longest edge of the random minimal spanning tree. *The annals of applied probability*, pages 340–361, 1997

[5] E. Gilbert. Random graphs. *The Annals of Mathematical Statistics*, pages 1141–1144, 1959

[6] P. Erdös and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci*, 5:17–61, 1960

[7] E. Gilbert. Random plane networks. *Journal of the Society for Industrial & Applied Mathematics*, 9(4):533–543, 1961

[8] P. Erdös and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci*, 5:17–61, 1960

[9] B. Bollobás. *Random graphs*. Cambridge University Press, 2001; Milan Bradonjić and Will Perkins. On sharp thresholds in random geometric graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 500–514, 2014; E. Friedgut and G. Kalai. Every monotone graph property has a sharp threshold. *Proceedings of the American Mathematical Society*, 124(10):2993–3002, 1996; A. Goel, S. Rai, and B. Krishnamachari. Sharp thresholds for monotone properties in random geometric graphs. In *Proceedings of STOC*, pages 580–586. ACM, 2004; and Gregory L. Mccolm.

$G_{n,p}$ and $G(n,r)$.

Bollobás and Thomason [10] showed that any monotone property in random graphs has a threshold function. See also a result of Friedgut and Kalai [11], and a result of Bourgain and Kalai [12]. In the Erdös-Rényi random graph $G_{n,p}$, the connectivity threshold is $p = \log n/n$ and the threshold for having a giant component is $p = 1/n$; see [13]. The planarity threshold for $G_{n,p}$ is $p = 1/n$; see [14].

A general reference on random geometric graphs is [15]. There is extensive literature on various aspects of random geometric graphs of which we mention the related work on coverage by [16] and a review on percolation, connectivity, coverage and colouring by [17]. As in random graphs, any monotone property in geometric random graphs has a threshold function [18].

Random geometric graphs have a connectivity threshold of $\sqrt{\ln n/n}$; see [19]. Gupta and Kumar [20] provided a connectivity threshold for points that are uniformly distributed in a disk. By a result of Penrose [21], in $G(n,r)$, any threshold function for having no isolated vertex (a vertex of degree zero) is also a connectivity threshold function. Panchapakesan and Manjunath [22] showed that $\sqrt{\ln n/n}$ is a threshold for being an isolated vertex in $G(n,r)$. This implies that $\sqrt{\ln n/n}$ is a connectivity threshold for $G(n,r)$. For $k \geq 2$, the details on the $k$-connectivity threshold in random geometric graphs can be found in [23]. Connectivity of random geometric graphs for points on a line is studied by Godehardt and Jaworski [24]. Appel and Russo [25] considered the connectivity under the $L_\infty$-norm.

**Organization:** In Section 11.3.1 we show that for a constant $k$, the distance threshold for having a connected subgraph on $k$ points is $n^{\frac{-k}{2k-2}}$. We show that the same threshold is valid for the existence of a clique of size $k$. Based on that, we prove the following thresholds for a random geometric graph to be plane or planar. In Section 11.3.2, we prove that $n^{-2/3}$ is a distance threshold for a random geometric graph to be plane. In Section 11.3.3, we prove that $n^{-5/8}$ is a distance threshold for a random geometric graph to be planar.

### 11.3.1 *The threshold for having a connected subgraph on k points*

In this section, we look for the distance threshold for "existence of connected subgraphs of constant size"; this is an increasing property. For a given constant $k$, we show that $n^{\frac{-k}{2k-2}}$ is the threshold function for the existence of a connected subgraph on $k$ points in $G(n,r)$. Specifically, we show that if $r = o(n^{\frac{-k}{2k-2}})$, then w.h.p. $G(n,r)$ has no connected subgraph on $k$ points, and if $r = \omega(n^{\frac{-k}{2k-2}})$, then w.h.p. $G(n,r)$ has a connected subgraph on $k$ points. We also show that the same threshold function holds for the existence of a clique of size $k$.

[10] Béla Bollobás and Andrew Thomason. Threshold functions. *Combinatorica*, 7(1):35–38, 1987

[11] E. Friedgut and G. Kalai. Every monotone graph property has a sharp threshold. *Proceedings of the American Mathematical Society*, 124(10):2993–3002, 1996

[12] Jean Bourgain and Gil Kalai. Threshold intervals under group symmetries. *Convex Geometric Analysis MSRI Publications Volume 34*, pages 59–63, 1998

[13] N. Alon and J. H. Spencer. *The probabilistic method*. John Wiley & Sons, 3rd edition, 2007

[14] B. Bollobás. *Random graphs*. Cambridge University Press, 2001; and J. H. Spencer. *Ten lectures on the probabilistic method*, volume 52. SIAM, 1987

[15] Mathew D. Penrose. *Random geometric graphs*, volume 5. Oxford University Press Oxford, 2003

[16] P. Hall. On the coverage of $k$-dimensional space by $k$-dimensional spheres. *The Annals of Probability*, 13(3):991–1002, 1985; and S. Janson. Random coverings in several dimensions. *Acta Mathematica*, 156(1):83–118, 1986

[17] P. Balister, A. Sarkar, and B. Bollobás. Percolation, connectivity, coverage and colouring of random geometric graphs. In *Handbook of Large-Scale Random Networks*, pages 117–142. Springer, 2008

[18] Milan Bradonjić and Will Perkins. On sharp thresholds in random geometric graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 500–514, 2014; A. Goel, S. Rai, and B. Krishnamachari. Sharp thresholds for monotone properties in random geometric graphs. In *Proceedings of STOC*, pages 580–586. ACM, 2004; Bhaskar Krishnamachari, Stephen B. Wicker, Rámon Béjar, and Marc Pearlman. Critical density thresholds in distributed wireless networks. In *Communications, information and network security*, 2002; and Gregory L. Mccolm. Threshold functions for random graphs on a line segment. *Combinatorics, Probability and Computing*, 13:373–387, 2001

[19] Piyush Gupta and P. R. Kumar. Critical power for asymptotic connectivity in wireless networks. In *Stochastic Analysis, Control, Optimization and Applications*, pages 547–566, 1998; Padma Panchapakesan and D Manjunath. On the transmission range in dense ad hoc radio networks. In *Proceedings of IEEE Signal Processing Communication (SPCOM)*, 2001; and Mathew D. Penrose. The longest edge of the random minimal spanning tree. *The annals of applied probability*, pages 340–361, 1997

[20] Piyush Gupta and P. R. Kumar. Critical power for asymptotic connectivity in wireless networks. In *Stochastic Analysis,*

**Theorem 11.3.1** *Let $k \geq 2$ be an integer constant. Then, $n^{\frac{-k}{2k-2}}$ is a distance threshold function for $G(n,r)$ to have a connected subgraph on $k$ points.*

**Proof.** Let $P_1, \ldots, P_{\binom{n}{k}}$ be an enumeration of all subsets of $k$ points in $G(n,r)$. Let $DG[P_i]$ be the subgraph of $G(n,r)$ that is induced by $P_i$. Let $X_i$ be the random variable such that

$$X_i = \begin{cases} 1 & \text{if } DG[P_i] \text{ is connected,} \\ 0 & \text{otherwise.} \end{cases}$$

Let the random variable $X$ count the number of sets $P_i$ for which $DG[P_i]$ is connected. It is clear that

$$X = \sum_{i=1}^{\binom{n}{k}} X_i. \tag{11.8}$$

Observe that $\mathrm{E}[X_i] = \Pr[X_i = 1]$. Since the random variables $X_i$ have identical distributions, we have

$$\mathrm{E}[X] = \binom{n}{k} \mathrm{E}[X_1]. \tag{11.9}$$

We obtain an upper bound and a lower bound for $\Pr[X_i = 1]$. First, partition the unit square into squares of side equal to $r$. Let $\{s_1, \ldots, s_{1/r^2}\}$ be the resulting set of squares. For a square $s_t$, let $S_t$ be the $kr \times kr$ square which has $s_t$ on its left bottom corner; see Figure 11.1(a). $S_t$ contains at most $k^2$ squares each of side length $r$ (each $S_t$ on the boundary of the unit square contains less than $k^2$ squares). Let $A_{i,t}$ be the event that all points in $P_i$ are contained in $S_t$. Observe that if $DG[P_i]$ is connected then $P_i$ lies in $S_t$ for some $t \in \{1, \ldots, 1/r^2\}$. Therefore,

$$\text{if } DG[P_i] \text{ is connected, then } (A_{i,1} \vee A_{i,2} \vee \cdots \vee A_{i,1/r^2}),$$

and hence we have

$$\Pr[X_i = 1] \leq \sum_{t=1}^{1/r^2} \Pr[A_{i,t}] \leq \sum_{t=1}^{1/r^2} (k^2 r^2)^k = k^{2k} r^{2k-2}. \tag{11.10}$$

Now, partition the unit square into squares with diagonal length equal to $r$. Each such square has side length equal to $r/\sqrt{2}$. Let $\{s_1, \ldots, s_{2/r^2}\}$ be the resulting set of squares. Let $B_{i,t}$ be the event that all points of $P_i$ are in $s_t$. Observe that if all points of $P_i$ are in the same square, then $DG[P_i]$ is a complete graph and hence connected. Therefore,

$$\text{if } (B_{i,1} \vee B_{i,2} \vee \cdots \vee B_{i,2/r^2}), \text{ then } DG[P_i] \text{ is connected,}$$

and hence we have

$$\Pr[X_i = 1] \geq \sum_{t=1}^{2/r^2} \Pr[B_{i,t}] = \sum_{t=1}^{2/r^2} \left(\frac{r^2}{2}\right)^k = \frac{1}{2^{k-1}} r^{2k-2}. \tag{11.11}$$

Since $k \geq 2$ is a constant, Inequalities (11.10) and (11.11) and Equation (11.9) imply that

$$\mathrm{E}[X_i] = \Theta(r^{2k-2}), \tag{11.12}$$

$$\mathrm{E}[X] = \Theta(n^k r^{2k-2}). \tag{11.13}$$

If $n \to \infty$ and $r = o(n^{\frac{-k}{2k-2}})$ we conclude that the following inequalities are valid

$$\begin{aligned} \Pr[X \geq 1] &\leq \mathrm{E}[X] \text{ (by Markov's Inequality)} \\ &= \Theta(n^k r^{2k-2}) \text{ (by (11.13))} \\ &= o(1). \end{aligned} \tag{11.14}$$

Therefore, w.h.p. $G(n, r)$ has no connected subgraph on $k$ points.

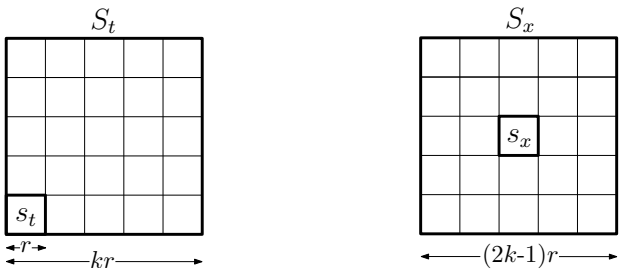



Figure 11.1: (a) The square $S_t$ has $s_t$ on its left bottom corner. (b) The square $S_x$ which is centered at $s_x$.

In the rest of the proof, we assume that $r = \omega(n^{\frac{-k}{2k-2}})$. In order to show that w.h.p. $G(n, r)$ has at least one connected subgraph on $k$ vertices, we show, using the second moment method [26], that $\Pr[X = 0] \to 0$ as $n \to \infty$. Recall from Chebyshev's inequality that

$$\Pr[X = 0] \leq \frac{\mathrm{Var}(X)}{\mathrm{E}[X]^2}. \tag{11.15}$$

Therefore, in order to show that $\Pr[X = 0] \to 0$, it suffices to show that

$$\frac{\mathrm{Var}(X)}{\mathrm{E}[X]^2} \to 0. \tag{11.16}$$

In view of Identity (11.8) we have

$$\mathrm{Var}(X) = \sum_{1 \leq i,j \leq \binom{n}{k}} \mathrm{Cov}(X_i, X_j), \tag{11.17}$$

where $\mathrm{Cov}(X_i, X_j) = \mathrm{E}[X_i X_j] - \mathrm{E}[X_i]\mathrm{E}[X_j] \leq \mathrm{E}[X_i X_j]$. If $|P_i \cap P_j| = 0$ then $DG[P_i]$ and $DG[P_j]$ are disjoint. Thus, the random variables $X_i$

[26] N. Alon and J. H. Spencer. *The probabilistic method*. John Wiley & Sons, 3rd edition, 2007

and $X_j$ are independent, and hence $\mathrm{Cov}(X_i, X_j) = 0$. It is enough to consider the cases when $P_i$ and $P_j$ are not disjoint. Assume $|P_i \cap P_j| = w$, where $w \in \{1, \ldots, k\}$. Thus, in view of Equation (11.17), we have

$$\begin{aligned} \mathrm{Var}(X) &= \sum_{w=1}^{k} \sum_{|P_i \cap P_j| = w} \mathrm{Cov}(X_i, X_j) \\ &\leq \sum_{w=1}^{k} \sum_{|P_i \cap P_j| = w} \mathrm{E}[X_i X_j]. \end{aligned} \tag{11.18}$$

The computation of $\mathrm{E}[X_i, X_j]$ involves some geometric considerations which are being discussed in detail below. Since $X_i$ and $X_j$ are 0-1 random variables, $X_i X_j$ is a 0-1 random variable and

$$X_i X_j = \begin{cases} 1 & \text{if both } DG[P_i] \text{ and } DG[P_j] \text{ are connected,} \\ 0 & \text{otherwise.} \end{cases}$$

By the definition of the expected value we have

$$\begin{aligned} \mathrm{E}[X_i X_j] &= \Pr[X_j = 1 | X_i = 1] \Pr[X_i = 1] \\ &= \Pr[X_j = 1 | X_i = 1] \mathrm{E}[X_i]. \end{aligned} \tag{11.19}$$

By (11.12), $\mathrm{E}[X_i] = \Theta(r^{2k-2})$. It remains to compute $\Pr[X_j = 1 | X_i = 1]$, i.e., the probability that $DG[P_j]$ is connected given that $DG[P_i]$ is connected. Consider the $k$-tuples $P_i$ and $P_j$ under the condition that $DG[P_i]$ is connected. Let $x$ be a point in $P_i \cap P_j$. Partition the unit square into squares of side length equal to $r$. Let $s_x$ be the square containing $x$. Let $S_x$ be the $(2k-1)r \times (2k-1)r$ square centered at $s_x$. $S_x$ contains at most $(2k-1)^2$ squares each of side length $r$ (if $S_x$ is on the boundary of the unit square then it contains less than $(2k-1)^2$ squares); see Figure 11.1(b). The area of $S_x$ is at most $(2kr)^2$, and hence the probability that a specific point of $P_j$ is in $S_t$ is at most $4k^2r^2$. Since $P_i$ and $P_j$ share $w$ points, in order for $DG[P_j]$ to be connected, the remaining $k - w$ points of $P_j$ must lie in $S_x$. Thus, the probability that $DG[P_j]$ is connected given that $DG[P_i]$ is connected is at most $(4k^2r^2)^{k-w} \leq c_w r^{2k-2w}$, for some constant $c_w > 0$. Thus, $\Pr[X_j = 1 | X_i = 1] \leq c_w r^{2k-2w}$. In view of Equation (11.19), we have

$$\mathrm{E}[X_i X_j] \leq c'_w \cdot r^{2k-2w} \cdot r^{2k-2} = c'_w r^{4k-2w-2}, \tag{11.20}$$

for some constant $c'_w > 0$.

Since $P_i$ and $P_j$ are $k$-tuples which share $w$ points, $|P_i \cup P_j| = 2k - w$. There are $\binom{n}{2k-w}$ ways to choose $2k - w$ points for $P_i \cup P_j$. Since we choose $w$ points for $P_i \cap P_j$, $k - w$ points for $P_i$ alone, and $k - w$ points for $P_j$ alone, there are $\binom{2k-w}{w, k-w, k-w}$ ways to split the $2k - w$

chosen points into $P_i$ and $P_j$. Based on this and Inequality (11.20), Inequality (11.18) turns out to

$$\begin{aligned}\mathrm{Var}(X) &\leq \sum_{w=1}^{k} \sum_{|P_i \cap P_j| = w} \mathrm{E}[X_i X_j] \\ &\leq \sum_{w=1}^{k} \binom{n}{2k-w} \binom{2k-w}{w, k-w, k-w} c'_w r^{4k-2w-2} \\ &\leq \sum_{w=1}^{k} c''_w n^{2k-w} r^{4k-2w-2}.\end{aligned}$$

for some constants $c''_w > 0$. Consider (11.16) and note that by (11.13), $\mathrm{E}[X]^2 \geq c'' n^{2k} r^{4k-4}$, for some constant $c'' > 0$. Thus,

$$\begin{aligned}\frac{\mathrm{Var}(X)}{\mathrm{E}[X]^2} &\leq \sum_{w=1}^{k} \frac{c''_w n^{2k-w} r^{4k-2w-2}}{c'' n^{2k} r^{4k-4}} = \sum_{w=1}^{k} \frac{c''_w}{c''} \cdot \frac{1}{n^w r^{2w-2}} \\ &= \frac{c''_1}{c''} \cdot \frac{1}{n^1 r^0} + \frac{c''_2}{c''} \cdot \frac{1}{n^2 r^2} + \cdots + \frac{c''_k}{c''} \cdot \frac{1}{n^k r^{2k-2}} \qquad (11.21)\end{aligned}$$

Since $r = \omega(n^{\frac{-k}{2k-2}})$, all terms in (11.21) tend to zero. This proves the convergence in (11.16). Thus, $\Pr[X = 0] \to 0$ as $n \to \infty$. This implies that if $r = \omega(n^{\frac{-k}{2k-2}})$, then $G(n, r)$ has a connected subgraph on $k$ vertices with high probability. ■ ■

In the following theorem we show that if $k = O(1)$, then $n^{\frac{-k}{2k-2}}$ is also a threshold for $G(n, r)$ to have a clique of size $k$; this is an increasing property.

**Theorem 11.3.2** *Let $k \geq 2$ be an integer constant. Then, $n^{\frac{-k}{2k-2}}$ is a distance threshold function for $G(n, r)$ to have a clique of size $k$.*

**Proof.** By Theorem 11.3.1, if $r = o(n^{\frac{-k}{2k-2}})$, then w.h.p. $G(n, r)$ has no connected subgraph on $k$ vertices, and hence it has no clique of size $k$. This proves the first statement. We prove the second statement by adjusting the proof of Theorem 11.3.1, which is based on the second moment method. Assume $r = \omega(n^{\frac{-k}{2k-2}})$. Let $P_1, \ldots, P_{\binom{n}{k}}$ be an enumeration of all subsets of $k$ points. Let $X_i$ be equal to 1 if $DG[P_i]$ is a clique, and 0 otherwise. Let $X = \sum X_i$.

Partition the unit square into a set $\{s_1, \ldots, s_{1/r^2}\}$ of squares of side length $r$. Let $S_t$ be the $2r \times 2r$ square which has $s_t$ on its left bottom corner. If $DG[P_i]$ is a clique then $P_i$ lies in $S_t$ for some $t \in \{1, \ldots, 1/r^2\}$. Therefore,

$$\Pr[X_i = 1] \leq 4^k r^{2k-2}.$$

Now, partition the unit square into a set $\{s_1, \ldots, s_{2/r^2}\}$ of squares with diagonal length $r$. If all points of $P_i$ fall in the square $s_t$, then $DG[P_i]$ is a clique. Thus,

$$\Pr[X_i = 1] \geq \frac{1}{2^{k-1}} r^{2k-2}.$$

Since $k \geq 2$ is a constant, we have

$$\begin{aligned} \mathrm{E}[X_i] &= \Theta(r^{2k-2}), \\ \mathrm{E}[X] &= \Theta(n^k r^{2k-2}). \end{aligned}$$

In view of Chebyshev's inequality we need to show that $\frac{\mathrm{Var}(X)}{\mathrm{E}[X]^2}$ tends to 0 as $n$ goes to infinity. We bound $\mathrm{Var}(X)$ from above by Inequality (11.18). Consider the $k$-tuples $P_i$ and $P_j$ under the condition that $DG[P_i]$ is a clique. Let $|P_i \cap P_j| = w$, and let $x$ be a point in $P_i \cap P_j$. Partition the unit square into squares of side length $r$. Let $s_x$ be the square containing $x$. Let $S_x$ be the $3r \times 3r$ square centered at $s_x$. In order for $DG[P_j]$ to be a clique, the remaining $k - w$ points of $P_j$ must lie in $S_x$. Thus,

$$\mathrm{E}[X_i X_j] \leq c'_w r^{4k-2w-2},$$

for some constant $c'_w > 0$. By a similar argument as in the proof of Theorem 11.3.1, we can show that for some constants $c'', c''_w > 0$ the followings inequalities are valid:

$$\begin{aligned} \mathrm{Var}(X) &\leq \sum_{w=1}^{k} c''_w n^{2k-w} r^{4k-2w-2}, \\ \frac{\mathrm{Var}(X)}{\mathrm{E}[X]^2} &\leq \sum_{w=1}^{k} \frac{c''_w}{c''} \cdot \frac{1}{n^w r^{2w-2}}. \end{aligned}$$

Since $r = \omega(n^{\frac{-k}{2k-2}})$, the last inequality tends to 0 as $n$ goes to infinity. This completes the proof for the second statement. ■ ■ As a direct consequence of Theorem 11.3.2, we have the following corollary.

**Corollary 11.3.3** *$n^{-1}$ is a threshold for $G(n,r)$ to have an edge, and $n^{-\frac{3}{4}}$ is a threshold for $G(n,r)$ to have a triangle.*

### 11.3.2 *The threshold for $G(n,r)$ to be plane*

In this section we investigate the threshold for a random geometric graph to be plane; this is a decreasing property. Recall that $G(n,r)$ is plane if no two of its edges cross. As a warm-up exercise we first prove a simple result which is based on the connectivity threshold for random geometric graphs, which is known to be $\sqrt{\ln n / n}$.

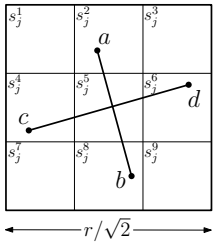



Figure 11.2: An square of diameter $r$ which is partitioned into nine sub-squares.

**Theorem 11.3.4** *If $r \geq \sqrt{\frac{c \ln n}{n}}$, with $c \geq 36$, then w.h.p. $G(n,r)$ is not plane.*

**Proof.** In order to prove that w.h.p. $G(n,r)$ is not plane, we show that w.h.p. it has a pair of crossing edges. Partition the unit square into squares each with diagonal length $r$. Then subdivide each such square into nine sub-squares as depicted in Figure 11.2. There are $\frac{18}{r^2}$ sub-squares, each of side length $\frac{r}{3\sqrt{2}}$. The probability that no point lies in a specific sub-square is $(1 - \frac{r^2}{18})^n$. Thus, the probability that there exists an empty sub-square is at most

$$\frac{18}{r^2}\left(1 - \frac{r^2}{18}\right)^n \leq n\left(1 - \frac{c \ln n}{18n}\right)^n \leq n^{1-c/18} \leq \frac{1}{n},$$

when $c \geq 36$. Therefore, with probability at least $1 - \frac{1}{n}$ all sub-squares contain points. By choosing four points $a$, $b$, $c$, and $d$ as depicted in Figure 11.2, it is easy to see that the edges $(a,b)$ and $(c,d)$ cross. Thus, w.h.p. $G(n,r)$ has a pair of crossing edges, and hence w.h.p. it is not plane. ■ ■

In fact, Theorem 11.3.4 ensures that w.h.p. there exists a pair of crossing edges in each of the squares. This implies that there are $\Omega\left(\frac{n}{\ln n}\right)$ disjoint pair of crossing edges, while for $G(n,r)$ to be not plane we need to show the existence of at least one pair of crossing edges. Thus, the value of $r$ provided by the connectivity threshold seems rather weak. By a different approach, in the rest of this section we show that $n^{-\frac{2}{3}}$ is the correct threshold.

**Lemma 11.3.5** *Let $(a,b)$ and $(c,d)$ be two crossing edges in $G(n,r)$, and let $Q$ be the convex quadrilateral formed by $a$, $b$, $c$, and $d$. Then, two adjacent sides of $Q$ are edges of $G(n,r)$.*

**Proof.** Refer to Figure 11.3. At least one of the angles of $Q$, say $\angle cad$, is bigger than or equal to $\pi/2$. It follows that in the triangle $\triangle cad$ the side $cd$ is the longest, i.e., $|cd| \geq \max\{|ac|, |ad|\}$. Since $|cd| \leq r$, both $|ac|$ and $|ad|$ are at most $r$. Thus, $ac$ and $ad$—which are adjacent—are edges of $G(n,r)$. ■ ■


Figure 11.3: (a) Illustration of Lemma 11.3.5. (b) Crossing edges $(a, b)$ and $(c, d)$ form an anchor.

In the proof of Lemma 11.3.5, $a$ is connected to $b$, $c$, and $d$. So the distance between $a$ to each of $b$, $c$, and $d$ is at most $r$. Thus, we have the following corollary.

**Corollary 11.3.6** *The endpoints of every two crossing edges in $G(n, r)$ are at distance at most $2r$ from each other. Moreover, there exists an endpoint which is within distance $r$ from other endpoints.*

Based on the proof of Lemma 11.3.5, we define an *anchor* as a set $\{a, b, c, d\}$ of four points in $G(n, r)$ such that three of them form a triangle, say $\triangle cad$, and the fourth vertex, $b$, is connected to $a$ by an edge which crosses $cd$; see Figure 11.3(b). We call $a$ as the *crown* of the anchor. The crown is within distance $r$ from the other three points. Note that $bc$ and $bd$ may or may not be edges of $G(n, r)$. In view of Lemma 11.3.5, two crossing edges in $G(n, r)$ form an anchor. Conversely, every anchor in $G(n, r)$ introduces a pair of crossing edges.

**Observation 11.3.7** *$G(n, r)$ is plane if and only if it has no anchor.*

**Theorem 11.3.8** *$n^{-\frac{2}{3}}$ is a threshold for $G(n, r)$ to be plane.*

**Proof.** In order to show that $G(n, r)$ is plane, by Observation 11.3.7, it is enough to show that it has no anchors. Every anchor has four points and it is connected. By Theorem 11.3.1, if $r = o(n^{-\frac{2}{3}})$, then w.h.p. $G(n, r)$ has no connected subgraph on 4 points, and hence it has no anchors. This proves the first statement.

We prove the second statement by adjusting the proof of Theorem 11.3.1 for $k = 4$. Assume $r = \omega(n^{-\frac{2}{3}})$. Let $P_1, \ldots, P_{\binom{n}{4}}$ be an enumeration of all subsets of 4 points. Let $X_i$ be equal to 1 if $DG[P_i]$ contains an anchor, and 0 otherwise. Let $X = \sum X_i$. In view of Chebyshev's inequality we need to show that $\frac{\mathrm{Var}(X)}{\mathrm{E}[X]^2}$ tends to 0 as $n$ goes to infinity.

Partition the unit square into a set $\{s_1, \ldots, s_{2/r^2}\}$ of squares with diagonal length $r$. Then, subdivide each square $s_j$, into nine subsquares $s_j^1, \ldots, s_j^9$ as depicted in Figure 11.2. If each of $s_j^1, s_j^3, s_j^7, s_j^9$ or each of $s_j^2, s_j^4, s_j^6, s_j^8$ contains a point of $P_i$, then $DG[P_i]$ is a convex clique of size four and hence it contains an anchor. Thus,

$$\Pr[X_i = 1] \geq \frac{r^6}{2^3} \cdot \frac{2}{9^4}.$$

This implies that $\mathrm{E}[X_i] = \Omega(r^6)$, and hence $\mathrm{E}[X] = \Omega(n^4 r^6)$. Therefore,

$$\mathrm{E}[X]^2 \geq c'' n^8 r^{12},$$

for some constant $c'' > 0$. By a similar argument as in the proof of Theorem 11.3.1 we bound the variance of $X$ from above by

$$\mathrm{Var}(X) \leq c_1'' n^7 r^{12} + c_2'' n^6 r^{10} + c_3'' n^5 r^8 + c_4'' n^4 r^6.$$

Since $r = \omega(n^{-\frac{2}{3}})$, $\frac{\mathrm{Var}(X)}{\mathrm{E}[X]^2}$ tends to 0 as $n$ goes to infinity. That is, w.h.p. $G(n,r)$ has an anchor. By Observation 11.3.7, w.h.p. $G(n,r)$ is not plane. ■ ■

As a direct consequence of the proof of Theorem 11.3.8, we have the following:

**Corollary 11.3.9** *With high probability if a random geometric graph is not plane, then it has a clique of size four.*

Note that every anchor introduces a crossing and each crossing introduces an anchor. Since, every anchor is a connected graph and has four points, by (11.13) we have the following corollary.

**Corollary 11.3.10** *The expected number of crossings in $G(n,r)$ is $\Theta(n^4 r^6)$.*

### *11.3.3 The threshold for $G(n,r)$ to be planar*

In this section we investigate the threshold for the planarity of a random geometric graph; this is a decreasing property. By Kuratowski's theorem, a finite graph is planar if and only if it does not contain a subgraph that is a subdivision of $K_5$ or of $K_{3,3}$. Note that any plane random geometric graph is planar too; observe that the reverse statement may not be true. Thus, the threshold for planarity seems to be larger than the threshold of being plane. By a similar argument as in the proof of Theorem 11.3.4 we can show that if $r \geq \sqrt{c \ln n / n}$, then w.h.p. each square with diagonal length $r$ contains $K_5$, and hence $G(n,r)$ is not planar.

**Theorem 11.3.11** *$n^{-\frac{5}{8}}$ is a threshold for $G(n,r)$ to be planar.*

**Proof.** By Theorem 11.3.2, if $r = \omega(n^{-\frac{5}{8}})$, then w.h.p. $G(n,r)$ has a clique of size 5. Thus, w.h.p. $G(n,r)$ contains $K_5$ and hence it is not planar. This proves the second statement of the theorem.

If $r = o(n^{-\frac{5}{8}})$, then by Theorem 11.3.1, w.h.p. $G(n,r)$ has no connected subgraph on 5 points, and hence it has no $K_5$. Similarly, if $r = o(n^{-\frac{3}{5}})$, then w.h.p. $G(n,r)$ has no connected subgraph on 6 points, and hence it has no $K_{3,3}$. Since $n^{-\frac{5}{8}} < n^{-\frac{3}{5}}$, it follows that if $r = o(n^{-\frac{5}{8}})$, then w.h.p. $G(n,r)$ has neither $K_5$ nor $K_{3,3}$ as a subgraph.

Note that, in order to prove that $G(n,r)$ is planar, we have to show that it does not contain any subdivision of either $K_5$ or $K_{3,3}$. Any subdivision of either $K_5$ or $K_{3,3}$ contains a connected subgraph on $k \geq 5$ vertices. Since $n^{-5/8} < n^{-k/(2k-2)}$ for all $k \geq 5$, in view of Theorem 11.3.1, we conclude that if $r = o(n^{-\frac{5}{8}})$, then w.h.p. $G(n,r)$ has no subdivision of $K_5$ and $K_{3,3}$, and hence $G(n,r)$ is planar. This proves the first statement of the theorem. ■ ■

As a direct consequence of the proof of Theorem 11.3.11, we have the following:

**Corollary 11.3.12** *With high probability if a random geometric graph does not contain a clique of size five, then it is planar.*

### *11.3.4 Conclusions*

We presented thresholds for random geometric graphs to have a connected subgraph of constant size, to be plane, and to be planar. A natural open problem is to extend Theorem 11.3.1 for connected subgraphs of $k$ vertices where $k$ is not necessarily a constant, and for connected subgraphs of $k$ vertices which have diameter $\delta$.

## *11.4 Exercises*

**11.1** *Show that for two independent random variables, $X$ and $Y$, $Var[X+Y] = Var[X] + Var[Y]$.*

# 12
# *Additional Exercises*

This is mainly a dump of questions which have been asked in the assignments over years. Some of them will be incomplete as they are based on the seminars in the course.

## *12.1 Problems*

1. Construct the string matching automaton for the pattern $P = aabab$ and illustrate its operation on the text string $T = aaababaabaab$.
2. Assume that the decision version of the 3-SAT, Vertex Cover and Clique problems are NP-complete. An independent set of an undirected graph $G = (V, E)$ is a subset $I$ of $V$ such that no two vertices in $I$ are connected by an edge in $E$. The decision version of the independent set problem $(G, k)$, $k \geq 0$, is to determine whether $G$ has an independent set of size at least $k$. Prove that the Independent Set problem is NP-Complete.
3. Let $G = (V, E)$ be an undirected connected simple graph. A matching is a set of edges of $G$ such that no two edges in the set are incident on the same vertex. A matching is **maximal** if it is not a proper subset of any other matching. A matching is **maximum** if the number of edges in the matching is largest. A vertex cover of $G = (V, E)$ is a set of vertices $V' \subseteq C$ such that if $(u, v) \in E$, then either $u \in V'$ or $v \in V'$ or both in $V'$. The size of the vertex cover is the cardinality of the set $V'$.
   (a) Show that the size of a maximum matching in $G$ is a lower bound on the size of any vertex cover of $G$.
   (b) Consider a maximal matching $M$ in $G = (V, E)$. Let

   $$T = \{v \in V : \text{some edge in M is incident on v}\}.$$

   What can you say about the subgraph of $G$ induced by the vertices of $G$ that are not in $T$. Conclude that $2|M|$ is the size of a vertex cover for $G$.

(c) Present an $O(|E|)$ time (greedy) algorithm to compute maximal matching.

(d) Conclude that the above algorithm for computing maximal matching is a 2-approximation algorithm for maximum matching.

Consider the Proposal Algorithm that we discussed in the class. Its outlined below:
Input: A list of n men and n women, where each man has an ordered list of n women (a permutation) whom he will like to marry. Similarly, each woman has an ordered list (a permutation) of n men whom she will like to marry.
Output: A set of n pairs forming a stable perfect matching. Each pair consists of a man and a woman.

```
Proposal-Algorithm (M,W)
1. While there exists an unmarried man m do
2.     m proposes to the most preferred woman w on his list
         whom he has not proposed so far.
3.     if w is not married then w marries m (end if)
4.     if w is married to m' but prefers m over m' then
5.        w divorces m' and marries m
       (end if)
   (endwhile)
```

(a) Prove or Disprove: The output of this algorithm is always the same and is independent of in which order the men are picked up in Step 1. In other words the above algorithm produces the same set of stable matchings and is independent of the choice of men m in Line 1.

(b) Show that this algorithm is favorable to men compared to women. One possible way to show this is to construct example(s) where women do much better than men when the same algorithm is run by interchanging men with women.

(c) List all the data structures (including the operations) that you will use to implement the Proposal Algorithm. (e.g. Stacks/Arrays/...).

(d) State the complexity of the Proposal Algorithm using $O()$ notation (Remember to use your data structures and their operations from Problem 3). (Recall that there are at most $n * n$ iterations in this algorithm, since a man never proposes to the same woman twice and in each iteration there is a proposal made)

(e) Can you devise a proposal algorithm that is unbiased?

4. Given a binary tree with all the relevant pointers (child/parent), describe a simple algorithm, running in linear time, that can compute and store the size of the subtrees at each node in the tree. (Size of a subtree at a node $v$ is the total number of nodes, including itself, in the subtree rooted at $v$.) Justify why your algorithm is correct and analyze it and show that it runs in linear time.

5. Outline a search algorithm, running in $O(\log n)$ time, to report the $i$-th smallest number in a set consisting of $n$ elements. The set is represented as a red-black tree where in addition to the usual information that we store at a node (pointer to left child, right child, parent, key, colour) we also store the size of the subtree at that node. The parameter $i$ is supplied to the search algorithm at the run-time and assume that the red-black tree with the additional information about the size of the subtrees has been precomputed.

6. Assume that we have $n$-integers in the range 1000 to 9999. In the radix-sort method we sorted them by first sorting them using the (stable) counting sort by the Least-Significant digit and then the next least significant digit and so on. Why does this algorithm works? Sketch the main idea in the proof. Why does this algorithm fails when we first sort them using most-significant bit and then the second most significant bit and so on?

7. Assume that we are given $n$ intervals (possibly overlapping) on a line. Each interval is specified by its left and right end points. Devise an efficient algorithm that can find a point on the line which is contained in the maximum number of intervals. Your algorithm should run in $O(n^2)$ time. (Bonus Problem: Devise an algorithm that runs in $O(n \log n)$ time? Another Bonus Problem: Show that this problem has $\Omega(n \log n)$ as its lower bound.)

8. Devise an $O(n \log k)$ time algorithm to merge $k$-sorted lists into a single sorted list, where $n$ is the total number of elements in all input lists.

9. Suppose you are given $n$-points in the plane. We can define a complete graph on these points, where the weight of an edge $e = (u, v)$, is Euclidean distance between $u$ and $v$. We need to partition these points into $k$ non-empty clusters, for some $n > k > 0$. The property that this clustering should satisfy is that the minimum distance between any two clusters is maximized. (The distance between two clusters $A$ and $B$ is defined to be the minimum among the distances between pair of points, where one point is from cluster $A$ and the other from cluster $B$.) Show

that the connected components obtained after running Kruskal's algorithm till it finds all but the last $k-1$ (most expensive) edges of MST produces an optimal clustering.

10. Although the 3CNF-SAT is NP-Complete, show that in polynomial time we can determine whether a boolean formula given in disjunctive normal form is satisfiable; the formula consists of $n$ variables and $k$ clauses. A formula is in Disjunctive normal form, if clauses are joined by ORs and literals within a clause are joined by ANDs (DNF is OR of ANDs and CNF is AND of ORs!). You need to provide an algorithm and show that its correct and its running time is polynomial in $n$ and $k$.

11. Although the 3CNF-SAT is NP-Complete, show that 2CNF-SAT is solvable in polynomial time. (This is same as exercise 34.4-7 in Book and contains Hints!).

12. Given an integer $m \times n$ matrix $A$ and an integer $m$-vector $b$, the 0-1 integer programming problem asks whether there is an integer $n$-vector $x$ with elements in the set {0,1} such that $Ax \leq b$. Prove that 0-1 integer programming problem is NP-Complete by providing a reduction from 3CNF-SAT or Subset-Sum problem.

13. Construct an instance of the subset-sum problem corresponding to the following 3CNF-SAT:

    $$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$

    Provide a satisfying assignment to 3CNF-SAT and show that it provides a valid solution for the subset-sum problem.

14. For the same 3CNF-SAT in Problem 4, illustrate the reduction to the clique problem. Construct an equivalent graph and show that for a satisfying assignment you have an appropriate size clique and vice verse (Use colours to illustrate the clique!)

15. Consider the problem of CNF-Satisfiability (we are not putting any restriction on number of literals in each clause), where each variable only occurs at most twice (positive and negative form of a variable are not counted separately). Show that satisfiability can be determined in polynomial time in this case.

    Hint: If a variable only occurs in its positive (or negative) form, then we can just satisfy those clauses and remove them from consideration. The problem starts when we have both version of the variable (one clause containing it in positive form and the other in the negative form - but then the following resolution rule applies: if we have two clauses of the form $(x \vee w \vee y \vee z) \wedge$

$(\neg x \vee u \vee y \vee z)$, then this is satisfiable if and only if the clause $(y \vee z \vee u \vee w)$ is satisfiable. In other words we can remove the variable $x$ from consideration!)

16. You have invited 500 guests for your graduation party in a huge hall in Chateau Laurier. The guests needs to be seated, where each table has 20 seats. Unfortunately, the guests are not all friendly with each other; for sure you do not want two of your guests to sit on the same table if they are not friendly. Suppose you know the complete friendship matrix $F$ (a 0-1 matrix indicating whether a pair of guests $(i, j)$ are friendly or not). Can you devise a decision algorithm to decide whether it is possible to hold this party with the restriction of 20 per table and no two enemies land up on the same table! What about in general, where $n$ is the number of guests and $k$ is the number of guests per table and we can assume $k$ divides $n$? Is this problem NP-Complete?

17. Consider two sets $A$ and $B$, each having $n$ integers in the range from 0 to $cn$, where $c > 1$ is a constant. Define the Cartesian sum of $A$ and $B$ as the set $C$ given by $C = \{x + y : x \in A \text{ and } y \in B\}$. Note that the integers in $C$ are in the range 0 to $2cn$. We want to find all the elements of $C$ and the number of times each element of $C$ is realized as a sum of elements in $A$ and $B$. Provide an $O(n \log n)$ algorithm for this problem.

18. Given an undirected graph $G = (V, E)$ in which each vertex $v \in V$ has an associated positive weight $w(v)$. For any vertex cover $V' \subseteq V$, define the weight of the vertex cover $w(V') = \sum_{v \in V'} w(v)$. The goal is to find a vertex cover of minimum weight. Provide an Integer Linear Programming formulation for this problem. Then provide a relaxation of the integer program. Show that using the rounding techniques we can obtain a 2-approximation algorithm for this problem. Provide a formal proof that your solution is a 2-approximation.

19. Given a simple graph $G = (V, E)$, we define a cut to be a partition of the vertex set $V$ into two non-empty sets $A$ and $B$, where $A \cup B = V$ and $A \cap B = \emptyset$. An edge $(a, b) \in E$ is said to cross the cut if $a \in A$ and $b \in B$. The size of the cut corresponding to the partition $(A, B)$ is defined to be the number of edges crossing the cut. The maximum cut problem is to find a partition of $V$ such that the size of the cut is maximized. Consider the following algorithm:
Step 1: Find any partition of $V$.
Step 2: For every vertex $v \in V$, if $v$ would have more edges crossing the cut if placed in the opposite partition, then move $v$ to the opposite partition.

Prove the following

(a) Prove that the above algorithm runs in polynomial time. What is the running time?

(b) Prove that the size of the cut produced by the above algorithm is at least half of the size of the maximum cut. (In other words its an $1/2$-approximation algorithm.)

20. Although the 3CNF-SAT is $\mathcal{NP}$-Complete, show that in polynomial time we can determine whether a boolean formula given in disjunctive normal form is satisfiable. The formula consists of $n$ variables and $k$ clauses. (A formula is in Disjunctive normal form, if clauses are joined by ORs and literals within a clause are joined by ANDs). You need to provide an algorithm whose running time is polynomial in $n$ and $k$.
Show what is wrong with the following argument: Given a 3CNF-SAT, we can use distributive law to construct an equivalent formula in Disjunctive Normal Form. Here is an example:
$(x_1 \vee x_2 \vee \bar{x_3}) \wedge (\bar{x_1} \vee \bar{x_2}) = (x_1 \wedge \bar{x_1}) \vee (x_1 \wedge \bar{x_2}) \vee (x_2 \wedge \bar{x_1}) \vee (x_2 \wedge \bar{x_2}) \vee (\bar{x_3} \wedge \bar{x_1}) \vee (\bar{x_3} \wedge \bar{x_2})$.
We have just now shown that DNF is in $\mathcal{P}$. This implies that 3CNF-SAT is in $\mathcal{P}$ and $\mathcal{P} = \mathcal{NP}$.

21. Look at the list of A.M. Turing award winners - this is like the Nobel prize in CS. Identify at least three award winners who have worked in the field of Algorithms and/or Data Structures. Give some reasoning for your choice. For each of them, list two of their main contributions (i.e. publications), and state in your own words what is their main contribution. [I am not expecting more than two paragraphs per researcher.]

22. Prove that the expected running time of finding the closest pair among a set of $n$-ponts in plane using the randomized incremental construction is $O(n)$.
(Refer to the talk on Closest Pairs.)

23. Given a well-separated pair decomposition for a set of $n$-points in plane,

(a) Show how you can determine the closest pair of points in $O(n)$ time.

(b) Show how you can find an approximation to the diameter of the point set in $O(n)$ time. What kind of approximation factor it will be in terms of the separation parameter $s$. (Diameter is the largest distance among the pairs.)

24. Let $P$ be a set of $n$-points in plane. Define the complete graph $G = (V, E)$ to be the graph where $V = P$, and there is an edge $e = (uv)$ between each pairs of points $u, v \in P$. The weight of $e$ is the Euclidean distance between $u$ and $v$. Euclidean minimum spanning tree (EMST) of $P$ is defined to be the minimum spanning tree of $G$. Show that using the well-separated pair decomposition, EMST can be approximated in $O(n \log n)$ time. Note that $G$ has $\Omega(n^2)$ edges, and hence we cannot directly apply any of the minimum spanning tree algorithms.

25. Show that the Jaccard Distance which is defined as $1-$ {the Jaccard Similarity} between the two sets is a metric.

26. (a) Prove that a matching is maximum if and only if there are no augmenting paths with respect to that matching.
(b) Prove that a bipartite graph $G = (V = A \cup B, E)$ has a perfect matching if and only if for any subset $S \subseteq A$ the number of vertices adjacent to $S$ in $B$ (denote it by $N(S)$) must be as large as $|S|$ (i.e. $|N(S)| \geq |S|, \forall S \in A$).

27. Present a proof, in your own words, of the Isolating Lemma, (see the report on "Parallel Algorithm for Maximum Matching"). Where is it required in the parallel algorithm?

28. When applying amplification constructions to a locality-sensitive family of functions, we can apply an AND composition followed by ORs or vice-versa. Which order of composition is 'better', and why? Explain when you would apply AND followed by ORs, and when will you like to use ORs followed by ANDs.

29. Let $C$ be a circle, and let $V$ be a set of $n$ distinct vertices on its boundary. Form a maximal plane graph on $V$ (i.e. we connect as many pairs of vertices as possible by straight line segments, so that no two edges cross each other in their interior). Notice that we obtain a plane triangulation of $V$. Call this triangulation $X$. Show that $X$ has tree width of 2, and its tree decomposition can be computed in polynomial time. (Keywords: Triangulated simple polygon, dual graph, dual tree, tree-width)

30. Look at Procedure 1 in the report on Data Streams and Frequency Moment. Argue why $2^B$ is a good estimate of $F_0$.

31. Let $X = \{x_1, x_2, \cdots, x_n\}$ be a set of $n$-elements. Each element of $x_i$ has a positive weight $w_i > 0$. Let $\mathcal{Y} = \{Y_1, Y_2, \cdots, Y_m\}$ be a set of subsets of $X$ (i.e. each $Y_i \subset X$). A subset $H \subseteq X$ is called *nice* if $H \cap Y_i \neq \emptyset$, for $i = 1, \cdots, m$. The decision problem of finding a nice set of weight at most $W$ is $NP$-Hard. Let $W^*$

be the weight of the nice set with smallest possible weight. Let $\gamma = \max\{|Y_i|, i = 1, \cdots, m\}$. Provide an approximation algorithm, running in polynomial time, that computes a nice set whose weight is at most $\gamma W^*$.

32. Given a simple graph $G = (V, E)$, we define a cut to be a partition of the vertex set $V$ into two non-empty sets $A$ and $B$, where $A \cup B = V$ and $A \cap B = \emptyset$. An edge $(a, b) \in E$ is said to cross the cut if $a \in A$ and $b \in B$. The size of the cut corresponding to the partition $(A, B)$ is defined to be the number of edges crossing the cut. The maximum cut problem is to find a partition of $V$ such that the size of the cut is maximized. Consider the following algorithm:
Step 1: Find any partition of $V$.
Step 2: For every vertex $v \in V$, if $v$ would have more edges crossing the cut if placed in the opposite partition, then move $v$ to the opposite partition.
Prove the following

    (a) Prove that the above algorithm runs in polynomial time. What is the running time?

    (b) Prove that the size of the cut produced by the above algorithm is at least half of the size of the maximum cut. (In other words its an 1/2-approximation algorithm.)

33. Given a set $P$ of points the plane. For each point $p \in P$ we denote by $b(p)$ the maximum Euclidean distance between $p$ and any point in $P$, i.e., $b(p) = \max\{|pq| : q \in P\}$. We denote a point $p \in P$ with minimum $b(p)$ as the *center* of $P$. Let $p$ be the center of $P$. Present a constant time algorithm that finds a point $p'$ in $P$ such that $b(p') \leq 2 \cdot b(p)$. Analyze the running time and prove the correctness of your algorithm.

34. Suppose you have a set $S$ of $n$-points in the plane and you need to construct an approximate travelling salesperson tour $TSP(S)$ of $S$. All distances are measured with respect to Euclidean distance. You follow the following strategy. Choose any point $s \in S$, and initialize a trivial tour $T = \langle ss \rangle$. Now we will grow this tour. Find a point $v \in S \setminus \{s\}$ that is closest to $s$, and update the tour to include $v$ and the current tour becomes $T = \langle svs \rangle$. In general, suppose currently our tour consists of $k + 1$ vertices $T = \langle su_1u_2...u_ks \rangle$. Now find a vertex in $v \in S$ that is closest to (but distinct from) $s, u_1, u_2, ..., u_k$. Let $v$ be closest to $u \in \{s, u_1, u_2, ..., u_k\}$. Then the new tour is obtained by inserting $v$ just after $u$ in $T$. (For example, if $v$ was closest to $u_3$, than the new tour will be $T = \langle su_1u_2u_3vu_4...u_ks \rangle$.) We repeat this process till all the points in $S$ are added to $T$. Show that the cost of $T$ is at most twice the cost of an optimal tour. (Note

that the cost of a tour is the sum total of the costs of all the edges in the tour.)

# *Bibliography*


[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, 1974.

[2] L. Aleksandrov and H. Djidjev. Linear algorithms for partitioning embedded graphs of bounded genus. *SIAM J. Discret. Math.*, 9(1):129–150, February 1996.

[3] Lyudmil Aleksandrov, Hristo Djidjev, Hua Guo, and Anil Maheshwari. Partitioning planar graphs with costs and weights. *J. Exp. Algorithmics*, 11, February 2007.

[4] N. Alon and J. H. Spencer. *The probabilistic method*. John Wiley & Sons, 3rd edition, 2007.

[5] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS '06. 47th Annual IEEE Symposium on*, pages 459–468, 2006.

[6] Martin J. B. Appel and Ralph P. Russo. The connectivity of a graph on uniform points on $[0,1]^d$. *Statistics & Probability Letters*, 60(4):351–357, 2002.

[7] P. Balister, A. Sarkar, and B. Bollobás. Percolation, connectivity, coverage and colouring of random geometric graphs. In *Handbook of Large-Scale Random Networks*, pages 117–142. Springer, 2008.

[8] Ahmad Biniaz, Evangelos Kranakis, Anil Maheshwari, and Michiel Smid. Plane and planarity thresholds for random geometric graphs. In *Proc. ALGOSENSORS 2015 (Patras, Greece)*, Lecture Notes in Computer Science, Berlin, Germany, 2015. Springer.

[9] B. Bollobás. *Random graphs*. Cambridge University Press, 2001.

[10] Béla Bollobás and Andrew Thomason. Threshold functions. *Combinatorica*, 7(1):35–38, 1987.

[11] BÃľla BollobÃąs. *Modern Graph Theory*. Graduate texts in mathematics. Springer, Heidelberg, corrected edition, 1998.

[12] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. Elsevier, New York, 1976.

[13] Otakar BorÅŕvka. O jistém problému minimálním. *PrÃące mor. pÅŹÃ■rodovÄŽd. spol. v BrnÄŽ III*, 3:37–58, 1926.

[14] Otakar BorÅŕvka. Příspěvek k řešení otázky ekonomické stavby elektrovodních sítí. *Elektrotechnickỳ obzor*, 15:153–154, 1926.

[15] J. Bourgain. On lipschitz embedding of finite metric spaces in hilbert space. *Israel Journal of Mathematics*, 52(1-2):46–52, 1985.

[16] Jean Bourgain and Gil Kalai. Threshold intervals under group symmetries. *Convex Geometric Analysis MSRI Publications Volume 34*, pages 59–63, 1998.

[17] Milan Bradonjić and Will Perkins. On sharp thresholds in random geometric graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 500–514, 2014.

[18] Sergey Brin, Rajeev Motwani, Lawrence Page, and Terry Winograd. What can you do with a web in your pocket? *IEEE Data Eng. Bull.*, 21(2):37–47, 1998.

[19] Sergey Brin and Lawrence Page. Reprint of: The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 56(18):3825–3833, 2012.

[20] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60:327–336, 1998.

[21] A.Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29, 1997.

[22] Timothy M. Chan. Backwards analysis of the Karger-Klein-Tarjan algorithm for minimum spanning. *Inf. Process. Lett.*, 67(6):303–304, 1998.

[23] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[24] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[25] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Struct. Algorithms*, 22(1):60–65, 2003.

[26] Mayur Datar and Piotr Indyk. Locality-sensitive hashing scheme based on p-stable distributions. In *In SCG âĂŹ04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM Press, 2004.

[27] L.B. de Paula, R.S. Villaca, and M.F. Magalhaes. A locality sensitive hashing approach for conceptual classification. In *Semantic Computing (ICSC), 2010 IEEE Fourth International Conference on*, pages 408–413, 2010.

[28] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[29] B. Dixon, M. Rauch, and R. Tarjan. Verification and sensitivity analysis of minimum spanning trees in linear time. *SIAM Journal on Computing*, 21(6):1184–1192, 1992.

[30] D.P. Dubhashi and A. Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.

[31] P. Erdös and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci*, 5:17–61, 1960.

[32] Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16(6):1004–1022, 1987.

[33] Greg N. Frederickson. Optimal algorithms for tree partitioning. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '91, pages 168–177, Philadelphia, PA, USA, 1991. Society for Industrial and Applied Mathematics.

[34] E. Friedgut and G. Kalai. Every monotone graph property has a sharp threshold. *Proceedings of the American Mathematical Society*, 124(10):2993–3002, 1996.

[35] E. Gilbert. Random graphs. *The Annals of Mathematical Statistics*, pages 1141–1144, 1959.

[36] E. Gilbert. Random plane networks. *Journal of the Society for Industrial & Applied Mathematics*, 9(4):533–543, 1961.

[37] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. pages 518–529, 1997.

[38] Erhard Godehardt and Jerzy Jaworski. On the connectivity of a random interval graph. *Random Struct. Algorithms*, 9(1-2):137–161, 1996.

[39] A. Goel, S. Rai, and B. Krishnamachari. Sharp thresholds for monotone properties in random geometric graphs. In *Proceedings of STOC*, pages 580–586. ACM, 2004.

[40] D. Gorisse, M. Cord, and F. Precioso. Locality-sensitive hashing for chi2 distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(2):402–409, 2012.

[41] Piyush Gupta and P. R. Kumar. Critical power for asymptotic connectivity in wireless networks. In *Stochastic Analysis, Control, Optimization and Applications*, pages 547–566, 1998.

[42] Torben Hagerup. An even simpler linear-time algorithm for verifying minimum spanning trees. In Christophe Paul and Michel Habib, editors, *Graph-Theoretic Concepts in Computer Science*, volume 5911 of *Lecture Notes in Computer Science*, pages 178–189. Springer Berlin Heidelberg, 2010.

[43] Torben Hagerup and Christine Rüb. A guided tour of Chernoff bounds. *Inf. Process. Lett.*, 33(6):305–308, 1990.

[44] P. Hall. On the coverage of $k$-dimensional space by $k$-dimensional spheres. *The Annals of Probability*, 13(3):991–1002, 1985.

[45] Frank Harary. *Graph theory*. Addison-Wesley, 1991.

[46] D. Harel and R. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

[47] Dov Harel. A linear time algorithm for the lowest common ancestors problem. In *Foundations of Computer Science, 1980., 21st Annual Symposium on*, pages 308–319, 1980.

[48] S. Hu. Efficient video retrieval by locality sensitive hashing. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on*, volume 2, pages 449–452, 2005.

[49] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613, New York, NY, USA, 1998. ACM.

[50] S. Janson. Random coverings in several dimensions. *Acta Mathematica*, 156(1):83–118, 1986.

[51] Woojay Jeon and Yan-Ming Cheng. Efficient speaker search over large populations using kernelized locality-sensitive hashing. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 4261–4264, 2012.

[52] Yushi Jing and S. Baluja. Visualrank: Applying pagerank to large-scale image search. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(11):1877–1890, 2008.

[53] K.A. Kala and K. Chitharanjan. Locality sensitive hashing based incremental clustering for creating affinity groups in hadoop; hdfs - an infrastructure extension. In *Circuits, Power and Computing Technologies (ICCPCT), 2013 International Conference on*, pages 1243–1249, 2013.

[54] Zixiang Kang, Wei Tsang Ooi, and Qibin Sun. Hierarchical, non-uniform locality sensitive hashing and its application to video identification. In *Multimedia and Expo, 2004. ICME '04. 2004 IEEE International Conference on*, volume 1, pages 743–746 Vol.1, 2004.

[55] David R. Karger, Philip N. Klein, and Robert E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *J. ACM*, 42(2):321–328, March 1995.

[56] V. King. A simpler minimum spanning tree verification algorithm. *Algorithmica*, 18(2):263–270, 1997.

[57] Valerie King. A simpler minimum spanning tree verification algorithm. In SelimG. Akl, Frank Dehne, JÃűrg-RÃijdiger Sack, and Nicola Santoro, editors, *Algorithms and Data Structures*, volume 955 of *Lecture Notes in Computer Science*, pages 440–448. Springer Berlin Heidelberg, 1995.

[58] Philip N. Klein and Robert E. Tarjan. A randomized linear-time algorithm for finding minimum spanning trees. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, STOC '94, pages 9–15, New York, NY, USA, 1994. ACM.

[59] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[60] Donald E. Knuth. *The art of computer programming, volume 1-3*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[61] J. Komlos. Linear verification for spanning trees. In *Foundations of Computer Science, 1984. 25th Annual Symposium on*, pages 201–206, 1984.

[62] J. KomlÃşs. Linear verification for spanning trees. *Combinatorica*, 5(1):57–65, 1985.

[63] D. Kozen. *The design and analysis of algorithms*. Springer, 1992.

[64] Bhaskar Krishnamachari, Stephen B. Wicker, Rámon Béjar, and Marc Pearlman. Critical density thresholds in distributed wireless networks. In *Communications, information and network security*, 2002.

[65] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(6):1092–1104, 2012.

[66] Richard J. Lipton and Robert E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[67] L. Lovász and M.D. Plummer. *Matching Theory*. Akadémiai Kiadó, Budapest, 1986. Also published as Vol. 121 of the North-Holland Mathematics Studies, North-Holland Publishing, Amsterdam.

[68] A. Maheshwari and M. Smid. *Introduction to Theory of Computation*. Free Online, 2012.

[69] Udi Manber. Finding similar files in a large file system. In *in Proceedings of the USENIX Winter 1994 Technical Conference*, pages 1–10, 1994.

[70] Jiri Matousek. *Lectures on Discrete Geometry*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[71] JiÅŹÃ■ MatouÅąek. On the distortion required for embedding finite metric spaces into normed spaces. *Israel Journal of Mathematics*, 93(1):333–344, 1996.

[72] Gregory L. Mccolm. Threshold functions for random graphs on a line segment. *Combinatorics, Probability and Computing*, 13:373–387, 2001.

[73] P.L. Meyer. *Introductory probability and statistical applications*. Addison-Wesley, Boston, MA, USA, 1970.

[74] Gary L. Miller, Shang-Hua Teng, William Thurston, and Stephen A. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *J. ACM*, 44(1):1–29, January 1997.

[75] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005.

[76] Tanmoy Mondal, Nicolas Ragot, Jean-Yves Ramel, and Umapada Pal. A fast word retrieval technique based on kernelized locality sensitive hashing. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pages 1195–1199, 2013.

[77] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, New York, NY, USA, 1995.

[78] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, New York, NY, USA, 1995.

[79] Jaroslav NeÅąetÅŹil, Eva MilkovÃą, and Helena NeÅąetÅŹilovÃą. Otakar borÅŕvka on minimum spanning tree problem translation of both the 1926 papers, comments, history. *Discrete Mathematics*, 233(1âĂŞ3):3 – 36, 2001. <ce:title>Czech and Slovak 2</ce:title>.

[80] Padma Panchapakesan and D Manjunath. On the transmission range in dense ad hoc radio networks. In *Proceedings of IEEE Signal Processing Communication (SPCOM)*, 2001.

[81] Mathew D. Penrose. The longest edge of the random minimal spanning tree. *The annals of applied probability*, pages 340–361, 1997.

[82] Mathew D. Penrose. On $k$-connectivity for a geometric random graph. *Random Struct. Algorithms*, 15(2):145–164, 1999.

[83] Mathew D. Penrose. *Random geometric graphs*, volume 5. Oxford University Press Oxford, 2003.

[84] Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2011.

[85] Z. Rasheed, H. Rangwala, and D. Barbara. Lsh-div: Species diversity estimation using locality sensitive hashing. In *Bioinformatics and Biomedicine (BIBM), 2012 IEEE International Conference on*, pages 1–6, 2012.

[86] Kenneth H. Rosen. *Discrete Mathematics and Its Applications*. McGraw-Hill Higher Education, 5th edition, 2002.

[87] Baruch Schieber and Uzi Vishkin. On finding lowest common ancestors: Simplification and parallelization. In JohnH. Reif, editor, *VLSI Algorithms and Architectures*, volume 319 of *Lecture Notes in Computer Science*, pages 111–123. Springer New York, 1988.

[88] J. H. Spencer. *Ten lectures on the probabilistic method*, volume 52. SIAM, 1987.

[89] Gilbert Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, Wellesley, MA, fifth edition, 2016.

[90] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

[91] Robert Tarjan. *Data Structures and Network Algorithms*, volume 44, chapter 6. Minimum Spanning Trees, pages 71–83. SIAM, 1983.

[92] Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, April 1975.

[93] Robert Endre Tarjan. Applications of path compression on balanced trees. *J. ACM*, 26(4):690–715, October 1979.

[94] Qiang Wang, Zhiyuan Guo, Gang Liu, and Jun Guo. Entropy based locality sensitive hashing. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 1045–1048, 2012.

# *Index*