Algorithms: Parallel and Sequential

Umut A. Acar and Guy E. Blelloch

February 2019

Umut A. Acar
Carnegie Mellon University
Department of Computer Science GHC 9231
Pittsburgh PA 15213 USA

Guy E. Blelloch
Carnegie Mellon University
Department of Computer Science GHC 9211
Pittsburgh PA 15213 USA

Contents

I Probability **1**

1 Probability Theory **3**

1 Probability Spaces 3

2 Properties of Probability Spaces 5

2.1 The Union Bound 5

2.2 Conditional Probability 6

2.3 Law of Total Probability 6

2.4 Independence 7

2 Random Variables **9**

1 Probability Mass Function 10

2 Bernoulli, Binomial, and Geometric RVs 11

3 Functions of Random Variables 12

4 Conditioning 13

5 Independence 14

3 Expectation **15**

1 Definitions 15

2 Markov's Inequality 16
3 Composing Expectations 17
4 Linearity of Expectations 18
5 Conditional Expectation 19

II Graph Contraction and Applications 21

4 Introduction 23
1 Preliminaries 23
2 Graph Contraction 25

5 Edge Contraction 29
1 Edge Partition 29
1.1 Analysis of Parallel Edge Partition 32
2 Edge Contraction 33

6 Star Contraction 35
1 Star Partition 35
1.1 Analysis of Star Partition 40
2 Star Contraction 41

7 Graph Connectivity 45
1 Preliminaries 45
2 Algorithms for Connectivity 46

Part I

Probability

Chapter 1

Probability Theory

This chapter introduces the basics of discrete probability theory.

1 Probability Spaces

Probability theory is a mathematical study of uncertain situations such as a dice game. In probability theory, we model a situation with an uncertain ***outcome*** as an ***experiment*** and reason carefully about the likelihood of various outcomes in precise mathematical terms.

Example 1.1. Suppose we have two *fair* dice, meaning that each is equally likely to land on any of its six sides. If we toss the dice, what is the chance that their numbers sum to 4? To determine the probability we first notice that there are a total of $6 \times 6 = 36$ distinct outcomes. Of these, only three outcomes sum to 4 (1 and 3, 2 and 2, and 3 and 1). The probability of the event that the number sum up to 4 is therefore

$$\frac{\text{\# of outcomes that sum to } 4}{\text{\# of total possible outcomes}} = \frac{3}{36} = \frac{1}{12}$$

Sample Spaces and Events. A ***sample space*** Ω is an arbitrary and possibly infinite (but countable) set of possible outcomes of a probabilistic experiment. Any experiment will return exactly one outcome from the set. For the dice game, the sample space is the 36 possible outcomes of the dice, and an experiment (roll of the dice) will return one of them. An ***event*** is any subset of Ω, and most often representing some property common to multiple outcomes. For example, an event could correspond to outcomes in which the dice add to 4—this subset would be of size 3. We typically denote events by capital letters from the start of the alphabet, e.g. A, B, C. We often refer to the individual elements of Ω as ***elementary events***. We assign a probability to each event. Our model for probability is defined as follows.

Definition 1.1 (Probability Space)**.** A probability space consists of a ***sample space*** Ω representing the set of possible outcomes, and a ***probability measure***, which is a function $\mathbf{P}$ from all subsets of Ω (the ***events***) to a probability (real number). These must satisfy the following axioms.

- **Nonnegativity:** $\mathbf{P}[A] \in [0, 1]$.
- **Additivity:** for any two disjoint events A and B (i.e., $A \cap B = \emptyset$),

$$\mathbf{P}[A \cup B] = \mathbf{P}[A] + \mathbf{P}[B] \ .$$

- **Normalization:** $\mathbf{P}[\Omega] = 1$.

Note (Infinite Spaces). Probability spaces can have countably infinite outcomes. The additivity rule generalizes to infinite sums, e.g., the probability of the event consisting of the union of infinitely many number of disjoint events is the infinite sum of the probability of each event.

Note. When defining the probability space, we have not specified carefully the exact nature of events, because they may differ based on the experiment and what we are interested in. We do, however, need to take care when setting up the probabilistic model so that we can reason about the experiment correctly. For example, each outcome of the sample space must correspond to one unique actual outcome of the experiment. In other words, they must be mutually exclusive. Similarly, any actual outcome of the experiment must have a corresponding representation in the sample space.

Example 1.2 (Throwing Dice)**.** For our example of throwing two dice, the sample space consists of all of the 36 possible pairs of values of the dice:

$$\Omega = \{(1,1), (1,2), \ldots, (2,1), \ldots, (6,6)\}.$$

Each pair in the sample space corresponds to an outcome of the experiment. The outcomes are mutually exclusive and cover all possible outcomes of the experiment.

For example, having the first dice show up 1 and the second 4 is an outcome and corresponds to the element $(1, 4)$ of the sample space Ω.

The event that the "the first dice is 3" corresponds to the set

$$\begin{aligned} A &= \{(d_1, d_2) \in \Omega \mid d_1 = 3\} \\ &= \{(3,1), (3,2), (3,3), (3,4), (3,5), (3,6)\} \ . \end{aligned}$$

The event that "the dice sum to 4" corresponds to the set

$$\begin{aligned} B &= \{(d_1, d_2) \in \Omega \mid d_1 + d_2 = 4\} \\ &= \{(1,3), (2,2), (3,1)\} \ . \end{aligned}$$

Assuming the dice are unbiased, the probability measure is defined by all elementary events having equal probability, i.e.,

$$\forall x \in \Omega, \quad \mathbf{P}[\{x\}] = \frac{1}{36}.$$

The probability of the event A (that the first dice is 3) is thus

$$\mathbf{P}[A] = \sum_{x \in A} \mathbf{P}[\{x\}] = \frac{6}{36} = \frac{1}{6}.$$

If the dice were biased so the probability of a given value is proportional to that value, then the probability measure would be $\mathbf{P}[\{(x,y)\}] = \frac{x}{21} \times \frac{y}{21}$, and the probability of the event B (that the dice add to 4) would be

$$\mathbf{P}[B] = \sum_{x \in B} \mathbf{P}[\{x\}] = \frac{1 \times 3 + 2 \times 2 + 3 \times 1}{21 \times 21} = \frac{10}{441}.$$

2 Properties of Probability Spaces

Given a probability space, we can prove several properties of probability measures by using the three axioms that they must satisfy. For example, if for two events A and B. We have

- if $A \subseteq B$, then $\mathbf{P}[A] \leq \mathbf{P}[B]$,
- $\mathbf{P}[A \cup B] = \mathbf{P}[A] + \mathbf{P}[B] - \mathbf{P}[A \cap B]$.

2.1 The Union Bound

The union bound, also known as Boole's inequality, is a simple way to obtain an upper bound on the probability of any of a collection of events happening. Specifically for a collection of events $A_0, A_2, \ldots, A_{n-1}$ the bound is:

$$\mathbf{P}\left[\bigcup_{0 \leq i < n} A_i\right] \leq \sum_{i=0}^{n-1} \mathbf{P}[A_i]$$

This bound is true unconditionally. To see why the bound holds we note that the elementary events in the union on the left are all included in the sum on the right (since the union comes from the same set of events). In fact they might be included multiple times in the sum on the right, hence the inequality. In fact the sum on the right could add to more than one, in which case the bound is not useful. The union bound can be useful in generating high-probability bounds for algorithms. For example, when the probability of each of n events is very low, e.g. $1/n^5$ and the sum remains very low, e.g. $1/n^4$.

2.2 Conditional Probability

Conditional probability allows us to reason about dependencies between observations. For example, suppose that your friend rolled a pair of dice and told you that they sum up to 6, what is the probability that one of dice has come up 1? Conditional probability has many practical applications. For example, given that a medical test for a disease comes up positive, we might want to know the probability that the patient has the disease. Or, given that your computer has been working fine for the past 2 years, you might want to know the probability that it will continue working for one more year.

Definition 1.2 (Conditional Probability)**.** For a given probability space, we define the ***conditional probability*** of an event A given B, as the probability of A occurring given that B occurs as

$$\mathbf{P}[A \mid B] = \frac{\mathbf{P}[A \cap B]}{\mathbf{P}[B]}.$$

The conditional probability measures the probability that the event A occurs given that B does. It is defined only when $\mathbf{P}[B] > 0$.

Conditional Probability is a Probability Measure. Conditional probability satisfies the three axioms of probability measures and thus itself a probability measure. We can thus treat conditional probabilities just as ordinary probabilities. Intuitively, conditional probability can be thought as a focusing and re-normalization of the probabilities on the assumed event B.

Example 1.3. Consider throwing two fair dice and calculate the probability that the first dice comes us 1 given that the sum of the two dice is 4. Let A be the event that the first dice comes up 1 and B the event that the sum is 4. We can write A and B in terms of outcomes as

$$\begin{aligned} A &= \{(1,1),(1,2),(1,3),(1,4),(1,5),(1,6)\} \text{ and} \\ B &= \{(1,3),(2,2),(3,1)\}. \end{aligned}$$

We thus have $A \cap B = \{(1,3)\}$. Since each outcome is equally likely,

$$\mathbf{P}[A \mid B] = \frac{\mathbf{P}[A \cap B]}{\mathbf{P}[B]} = \frac{|A \cap B|}{|B|} = \frac{1}{3}.$$

2.3 Law of Total Probability

Conditional probabilities can be useful in estimating the probability of an event that may depend on a selection of choices. The total probability theorem can be handy in such circumstances.

Theorem 1.1 (Law of Total Probability)**.** Consider a probabilistic space with sample space Ω and let $A_0, \ldots, A_{n-1}$ be a partition of Ω such that $\mathbf{P}[A_i] > 0$ for all $0 \leq i < n$. For any event B the following holds:

$$\begin{aligned}\mathbf{P}[B] &= \sum_{i=0}^{n-1} \mathbf{P}[B \cap A_i] \\ &= \sum_{i=0}^{n-1} \mathbf{P}[A_i]\,\mathbf{P}[B \mid A_i]\end{aligned}$$

Example 1.4. Your favorite social network partitions your connections into two kinds, near and far. The social network has calculated that the probability that you react to a post by one of your far connections is 0.1 but the same probability is 0.8 for a post by one of your near connections. Suppose that the social network shows you a post by a near and far connection with probability 0.6 and 0.4 respectively.

Let's calculate the probability that you react to a post that you see on the network. Let A_0 and A_1 be the event that the post is near and far respectively. We have $\mathbf{P}[A_0] = 0.6$ and $\mathbf{P}[A_1] = 0.4$. Let B the event that you react, we know that $\mathbf{P}[B \mid A_0] = 0.8$ and $\mathbf{P}[B \mid A_1] = 0.1$.

We want to calculate $\mathbf{P}[B]$, which by total probability theorem we know to be

$$\begin{aligned}\mathbf{P}[B] &= \mathbf{P}[B \cap A_0] + \mathbf{P}[B \cap A_1] \\ &= \mathbf{P}[A_0]\,\mathbf{P}[B \mid A_0] + \mathbf{P}[A_1]\,\mathbf{P}[B \mid A_1]\,. \\ &= 0.6 \cdot 0.8 + 0.4 \cdot 0.1 \\ &= 0.52.\end{aligned}$$

2.4 Independence

It is sometimes important to reason about the dependency relationship between events. Intuitively we say that two events are independent if the occurrence of one does not affect the probability of the other. More precisely, we define independence as follows.

Definition 1.3 (Independence)**.** Two events A and B are ***independent*** if

$$\mathbf{P}[A \cap B] = \mathbf{P}[A] \cdot \mathbf{P}[B]\,.$$

We say that multiple events $A_0, \ldots, A_{n-1}$ are ***mutually independent*** if and only if, for any non-empty subset $I \subseteq \{0, \ldots, n-1\}$,

$$\mathbf{P}\left[\bigcap_{i \in I} A_i\right] = \prod_{i \in I} \mathbf{P}[A_i]\,.$$

Independence and Conditional Probability. Recall that $\mathbf{P}[A \mid B] = \frac{\mathbf{P}[A \cap B]}{\mathbf{P}[B]}$ when $\mathbf{P}[B] > 0$. Thus if $\mathbf{P}[A \mid B] = \mathbf{P}[A]$ then $\mathbf{P}[A \cap B] = \mathbf{P}[A] \cdot \mathbf{P}[B]$. We can thus define independence in terms of conditional probability but this works only when $\mathbf{P}[B] > 0$.

Example 1.5. For two dice, the events $A = \{(d_1, d_2) \in \Omega \mid d_1 = 1\}$ (the first dice is 1) and $B = \{(d_1, d_2) \in \Omega \mid d_2 = 1\}$ (the second dice is 1) are independent since

$$\begin{array}{rlll} & \mathbf{P}[A] \times \mathbf{P}[B] & = & \frac{1}{6} \times \frac{1}{6} = \frac{1}{36} \\ = & \mathbf{P}[A \cap B] & = & \mathbf{P}[\{(1,1)\}] = \frac{1}{36}\,. \end{array}$$

However, the event $C \equiv \{X = 4\}$ (the dice add to 4) is not independent of A since

$$\begin{array}{rlll} & \mathbf{P}[A] \times \mathbf{P}[C] & = & \frac{1}{6} \times \frac{3}{36} = \frac{1}{72} \\ \neq & \mathbf{P}[A \cap C] & = & \mathbf{P}[\{(1,3)\}] = \frac{1}{36}\,. \end{array}$$

A and C are not independent since the fact that the first dice is 1 increases the probability they sum to 4 (from $\frac{1}{12}$ to $\frac{1}{6}$).

Exercise 1.1. For two dice, let A be the event that first roll is 1 and B be the event that the sum of the rolls is 5. Are A and B independent? Prove or disprove.

Consider now the same question but this time define B to be the event that the sum of the rolls is 7.

Chapter 2

Random Variables

This chapter introduces the random variables and their use in probability theory.

Definition 2.1 (Random Variable)**.** A ***random variable*** X is a real-valued function on the outcomes of an experiment, i.e., $X : \Omega \to \mathbb{R}$, i.e., it assigns a real number to each outcome. For a given probability space there can be many random variables, each keeping track of different quantities. We typically denote random variables by capital letters from the end of the alphabet, e.g. X, Y, and Z. We say that a random variable is ***discrete*** if its range is finite or countable infinite. Throughout this book, we only consider discrete random variables.

Example 2.1. For throwing two dice, we can define random variable as the sum of the two dice

$$X(d_1, d_2) = d_1 + d_2 \ ,$$

the product of two dice

$$Y(d_1, d_2) = d_1 \times d_2 \ ,$$

or the value of the first dice the two dice:

$$Z(d_1, d_2) = d_1 \ .$$

Definition 2.2 (Indicator Random Variable)**.** A random variable is called an ***indicator random variable*** if it takes on the value 1 when some condition is true and 0 otherwise.

Example 2.2. For throwing two dice, we can define indicator random variable as getting doubles

$$Y(d_1, d_2) = \begin{cases} 1 & \text{if } d_1 = d_2 \\ 0 & \text{if } d_1 \neq d_2 \ . \end{cases}$$

Using our shorthand, the event $\{X = 4\}$ corresponds to the event "the dice sum to 4".

Notation. For a random variable X and a value $x \in \mathbb{R}$, we use the following shorthand for the event corresponding to X equaling x:

$$\{X = x\} \equiv \{y \in \Omega \mid X(y) = x\} \ ,$$

and when applying the probability measure we use the further shorthand

$$\mathbf{P}\left[X = x\right] \equiv \mathbf{P}\left[\{X = x\}\right] \ .$$

Example 2.3. For throwing two dice, and X being a random variable representing the sum of the two dice, $\{X = 4\}$ corresponds to the event "the dice sum to 4", i.e. the set

$$\{y \in \Omega \mid X(y) = 4\} = \{(1,3),(2,2),(3,1)\} \ .$$

Assuming unbiased coins, we have that

$$\mathbf{P}\left[X = 4\right] = 1/12 \ .$$

Remark. The term random variable might seem counter-intuitive since it is actually a function not a variable, and it is not really random since it is a well defined deterministic function on the sample space. However if you think of it in conjunction with the random experiment that selects a elementary event, then it is a variable that takes on its value based on a random process.

1 Probability Mass Function

Definition 2.3 (Probability Mass Function)**.** For a discrete random variable X, we define its ***probability mass function*** or ***PMF***, written $\mathbf{P}_X(\cdot)$, for short as a function mapping each element x in the range of the random variable to the probability of the event $\{X = x\}$, i.e.,

$$\mathbf{P}_X(x) = \mathbf{P}\left[X = x\right] .$$

Example 2.4. The probability mass function for the indicator random variable X indicating whether the outcome of a roll of dice is comes up even is

$$\begin{aligned} &\mathbf{P}_X(0) = \mathbf{P}\left[\{X = 0\}\right] = \mathbf{P}\left[\{1,3,5\}\right] = 1/2, \text{ and} \\ &\mathbf{P}_X(1) = \mathbf{P}\left[\{X = 1\}\right] = \mathbf{P}\left[\{2,4,6\}\right] = 1/2. \end{aligned}$$

The probability mass function for the random variable X that maps each outcome in a roll of dice to the smallest Mersenne prime number no less than the outcome is

$$\begin{aligned} &\mathbf{P}_X(3) = \mathbf{P}\left[\{X = 3\}\right] = \mathbf{P}\left[\{1,2,3\}\right] = 1/2, \text{ and} \\ &\mathbf{P}_X(7) = \mathbf{P}\left[\{X = 7\}\right] = \mathbf{P}\left[\{4,5,6\}\right] = 1/2. \end{aligned}$$

Note that much like a probability measure, a probability mass function is a non-negative function. It is also additive in a similar sense: for any distinct x and x', the events $\{X = x\}$ and $\{X = x'\}$ are disjoint. Thus for any set $\bar{x}$ of values of X, we have

$$\mathbf{P}[X \in \bar{x}] = \sum_{x \in \bar{x}} \mathbf{P}_X(x).$$

Furthermore, since X is a function on the sample space, the events corresponding to the different values of X partition the sample space, and we have

$$\sum_x \mathbf{P}_X(x) = 1.$$

These are the important properties of probability mass functions: they are non-negative, normalizing, and are additive in a certain sense.

We can also compute the probability mass function for multiple random variables defined for the same probability space. For example, the ***joint probability mass function*** for two random variables X and Y, written $\mathbf{P}_{X,Y}(x, y)$ denotes the probability of the event $\{X = x\} \cap \{Y = y\}$, i.e.,

$$\mathbf{P}_{X,Y}(x, y) = \mathbf{P}[\{X = x\} \cap \{Y = y\}] = \mathbf{P}[X = x, Y = y].$$

Here $\mathbf{P}[X = x, Y = y]$ is shorthand for $\mathbf{P}[\{X = x\} \cap \{Y = y\}]$.

In our analysis or randomized algorithms, we shall repeatedly encounter a number of well-known random variables and create new ones from existing ones by composition.

2 Bernoulli, Binomial, and Geometric RVs

Bernoulli Random Variable. Suppose that we toss a coin that comes up a head with probability p and a tail with probability $1 - p$. The ***Bernoulli random variable*** takes the value 1 if the coin comes up heads and 0 if it comes up tails. In other words, it is an indicator random variable indicating heads. Its probability mass function is

$$\mathbf{P}_X(x) = \begin{cases} p & \text{if } x = 1 \\ 1 - p & \text{if } x = 0. \end{cases}$$

Binomial Random Variable. Consider n Bernoulli trials with probability p. We call the random variable X denoting the number of heads in the n trials as the ***Binomial random variable***. Its probability mass function for any $0 \leq x \leq n$ is

$$\mathbf{P}_X(x) = \binom{n}{x} p^x (1 - p)^{n-x}.$$

Geometric Random Variable. Consider performing Bernoulli trials with probability p until the coin comes up heads and X denote the number of trials needed to observe the first head. The random variable X is called the ***geometric random variable***. Its probability mass function for any $0 \leq x$ is

$$\mathbf{P}_X(x) = (1-p)^{x-1}p.$$

3 Functions of Random Variables

It is often useful to "apply" a function to one or more random variables to generate a new random variable. Specifically if we a function $f : \mathbb{R} \to \mathbb{R}$ and a random variable X we can compose the two giving a new random variable:

$$Y(x) = f(X(x))$$

We often write this shorthand as $Y = f(X)$. Similarly for two random variables X and Y we write $Z = X + Y$ as shorthand for

$$Z(x) = X(x) + Y(x)$$

or equivalently

$$Z = \lambda x.(X(x) + Y(x))$$

The probability mass function for the new variable can be computed by "massing" the probabilities for each value. For example, for a function of a random variable $Y = f(X)$, we can write the probability mass function as

$$\mathbf{P}_Y(y) = \mathbf{P}\left[Y = y\right] = \sum_{x \mid f(x)=y} \mathbf{P}_X(x)\ .$$

Example 2.5. Let X be a Bernoulli random variable with parameter p. We can define a new random variable Y as a transformation of X by a function $f(\cdot)$. For example, $Y = f(X) = 9X + 3$ is random variable that transforms X, e.g., $X = 1$ would be transformed to $Y = 12$. The probability mass function for Y reflects that of X, Its probability mass function is

$$\mathbf{P}_Y(y) = \begin{cases} p & \text{if } y = 12 \\ 1-p & \text{if } y = 3. \end{cases}$$

Example 2.6. Consider the random variable X with the probability mass function

$$\mathbf{P}_X(x) = \begin{cases} 0.25 & \text{if} \quad x = -2 \\ 0.25 & \text{if} \quad x = -1 \\ 0.25 & \text{if} \quad x = 0 \\ 0.25 & \text{if} \quad x = 1 \end{cases}$$

We can calculate the probability mass function for the random variable $Y = X^2$ as follows $\mathbf{P}_Y(y) = \sum_{x \mid x^2=y} \mathbf{P}_X(x)$. This yields

$$\mathbf{P}_Y(y) = \begin{cases} 0.25 & \text{if} \quad y = 0 \\ 0.5 & \text{if} \quad y = 1 \\ 0.25 & \text{if} \quad y = 4. \end{cases}$$

4 Conditioning

In the same way that we can condition an event on another, we can also condition a random variable on an event or on another random variable. Consider a random variable X and an event A in the same probability space, we define the ***conditional probability mass function*** of X conditioned on A as

$$\mathbf{P}_{X \mid A} = \mathbf{P}\left[X = x \mid A\right] = \frac{\mathbf{P}\left[\{X = x\} \cap A\right]}{\mathbf{P}\left[A\right]}.$$

Since for different values of x, $\{X = x\} \cap A$'s are disjoint and since X is a function over the sample space, conditional probability mass functions are normalizing just like ordinary probability mass functions, i.e., $\mathbf{P}_{X \mid A}(x) = 1$. Thus just as we can treat conditional probabilities as ordinary probabilities, we can treat conditional probability mass functions also as ordinary probability mass functions.

Example 2.7. Roll a pair of dice and let X be the sum of the face values. Let A be the event that the second roll came up 6. We can find the conditional probability mass function

$$\begin{aligned} \mathbf{P}_{X \mid A}(x) &= \frac{\mathbf{P}[\{X=x\} \cap A]}{\mathbf{P}[A]} \\ &= \begin{cases} \frac{1/36}{1/6} = 1/6 & \text{if } x = 7, \ldots, 12. \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

Conditional Probability Mass Function. Since random variables closely correspond with events, we can condition a random variable on another. More precisely, let X and Y be two random variables defined on the same probability space. We define the ***conditional probability mass function*** of X with respect to Y as

$$\mathbf{P}_{X \mid Y}(x \mid y) = \mathbf{P}\left[X = x \mid Y = y\right].$$

We can rewrite this as

$$\begin{aligned} \mathbf{P}_{X \mid Y}(x \mid y) &= \mathbf{P}\left[X = x \mid Y = y\right] \\ &= \frac{\mathbf{P}[X=x, Y=y]}{\mathbf{P}[Y=y]} \\ &= \frac{\mathbf{P}_{X,Y}(x,y)}{\mathbf{P}_Y y}. \end{aligned}$$

Conditional PMFs are PMFs. Consider the function $\mathbf{P}_{X \mid Y}(x \mid y)$ for a fixed value of y. This is a non-negative function of x, the event corresponding to different values of x are disjoint, and they partition the sample space, the conditional mass functions are normalizing

$$\sum_x \mathbf{P}_{X \mid Y}(x \mid y) = 1.$$

Conditional probability mass functions thus share the same properties as probability mass functions.

By direct implication of its definition, we can use conditional probability mass functions to calculate joint probability mass functions as follows

$$\mathbf{P}_{X,Y}(x,y) = \mathbf{P}_X(x)\mathbf{P}_{Y \mid X}(y \mid x)$$
$$\mathbf{P}_{X,Y}(x,y) = \mathbf{P}_Y(y)\mathbf{P}_{X \mid Y}(x \mid y).$$

As we can compute total probabilities from conditional ones as we saw earlier in this section, we can calculate marginal probability mass functions from conditional ones:

$$\mathbf{P}_X(x) = \sum_y \mathbf{P}_{X,Y}(x,y) = \sum_y \mathbf{P}_Y(y)\mathbf{P}_{X \mid Y}(x \mid y).$$

5 Independence

As with the notion of independence between events, we can also define independence between random variables and events. We say that a random variable X is ***independent of an event*** A, if

$$\text{for all } x : \mathbf{P}\left[\{X = x\} \cap A\right] = \mathbf{P}\left[X = x\right] \cdot \mathbf{P}\left[A\right] \ .$$

When $\mathbf{P}\left[A\right]$ is positive, this is equivalent to

$$\mathbf{P}_{X \mid A}(x) = \mathbf{P}_X(x).$$

Generalizing this to a pair of random variables, we say a random variable X is ***independent of a random variable*** Y if

$$\text{for all } x, y : \mathbf{P}\left[X = x, Y = y\right] = \mathbf{P}\left[X = x\right] \cdot \mathbf{P}\left[Y = y\right]$$

or equivalently

$$\text{for all } x, y : \mathbf{P}_{X,Y}(x,y) = \mathbf{P}_X(x) \cdot \mathbf{P}_Y(y).$$

In our two dice example, a random variable X representing the value of the first dice and a random variable Y representing the value of the second dice are independent. However X is not independent of a random variable Z representing the sum of the values of the two dice.

Chapter 3

Expectation

This chapter introduces expectation and its use in probability theory.

1 Definitions

The ***expectation*** of a random variable X in a probability space $(\Omega, \mathbf{P})$ is the sum of the random variable over the elementary events weighted by their probability, specifically:

$$\mathbf{E}_{\Omega,\mathbf{P}}[X] = \sum_{y \in \Omega} X(y) \cdot \mathbf{P}\left[\{y\}\right].$$

For convenience, we usually drop the $(\Omega, \mathbf{P}\,[])$ subscript on $\mathbf{E}$ since it is clear from the context.

Example 3.1. Assuming unbiased dice ($\mathbf{P}\left[(d_1, d_2)\right] = 1/36$), the expectation of the random variable X representing the sum of the two dice is:

$$\mathbf{E}\left[X\right] = \sum_{(d_1,d_2)\in\Omega} X(d_1, d_2) \times \frac{1}{36} = \sum_{(d_1,d_2)\in\Omega} \frac{d_1 + d_2}{36} = 7.$$

If we bias the coins so that for each dice the probability that it shows up with a particular value is proportional to the value, we have $\mathbf{P}\left[(d_1, d_2)\right] = (d_1/21) \times (d_2/21)$ and:

$$\mathbf{E}\left[X\right] = \sum_{(d_1,d_2)\in\Omega} \left((d_1 + d_2) \times \frac{d_1}{21} \times \frac{d_2}{21}\right) = 8\,\frac{2}{3}.$$

It is usually more natural to define expectations in terms of the probability mass function of the random variable

$$\mathbf{E}\left[X\right] = \sum_{x} x \cdot \mathbf{P}_X(x).$$

Example 3.2. The expectation of an indicator random variable X is the probability that the associated predicate is true (i.e. that $X = 1$):

$$\begin{aligned}\mathbf{E}[X] &= 0 \cdot \mathbf{P}_X(0) + 1 \cdot \mathbf{P}_X(1). \\ &= \mathbf{P}_X(1).\end{aligned}$$

Example 3.3. Recall that the probability mass function for a Bernoulli random variable is

$$\mathbf{P}_X(x) = \begin{cases} p & \text{if } x = 1 \\ 1-p & \text{if } x = 0. \end{cases}$$

Its expectation is thus

$$E[X] = p \cdot 1 + (1-p) \cdot 0 = p.$$

Example 3.4. Recall that the probability mass function for geometric random variable X with parameter p is

$$\mathbf{P}_X(x) = (1-p)^{x-1}p.$$

The expectation of X is thus

$$\begin{aligned}E[X] &= \sum_{x=1}^{\infty} x \cdot (1-p)^{x-1}p \\ &= p \cdot \sum_{x=1}^{\infty} x \cdot (1-p)^{x-1}\end{aligned}$$

Bounding this sum requires some basic manipulation of sums. Let $q = (1-p)$ and rewrite the sum as $p \cdot \sum_{x=0}^{\infty} xq^{x-1}$. Note now the term xq^{x-1} is the derivative of q^x with respect to q. Since the sum $\sum_{x=0}^{\infty} q^x = 1/(1-q)$, its derivative is $1/(1-q)^2 = 1/p^2$. We thus have conclude that $E[X] = 1/p$.

Example 3.5. Consider performing two Bernoulli trials with probability of success $1/4$. Let X be the random variable denoting the number of heads.

The probability mass function for X is

$$\mathbf{P}_X(x) = \begin{cases} 9/16 & \text{if } x = 0 \\ 3/8 & \text{if } x = 1 \\ 1/16 & \text{if } x = 2. \end{cases}$$

Thus $\mathbf{E}[X] = 0 + 1 \cdot 3/8 + 2 * 1/16 = 7/8$.

2 Markov's Inequality

Consider a non-negative random variable X. We can ask how much larger can X's maximum value be than its expected value. With small probability it can be arbitrarily much

larger. However, since the expectation is taken by averaging X over all outcomes, and it cannot take on negative values, X cannot take on a much larger value with significant probability. If it did it would contribute too much to the sum.

Exercise 3.1.

More generally X cannot be a multiple of β larger than its expectation with probability greater than $1/\beta$. This is because this part on its own would contribute more than $\beta\mathbf{E}[X] \times \frac{1}{\beta} = \mathbf{E}[X]$ to the expectation, which is a contradiction. This gives us for a non-negative random variable X the inequality:

$$\mathbf{P}[X \geq \beta\mathbf{E}[X]] \leq \frac{1}{\beta}$$

or equivalently (by substituting $\beta = \alpha/\mathbf{E}[X]$),

$$\mathbf{P}[X \geq \alpha] \leq \frac{\mathbf{E}[X]}{\alpha}$$

which is known as Markov's inequality.

3 Composing Expectations

Recall that functions or random variables are themselves random variables (defined on the same probability space), whose probability mass functions can be computed by considering the random variables involved. We can thus also compute the expectation of a random variable defined in terms of others. For example, we can define a random variable Y as a function of another variable X as $Y = f(X)$. The expectation of such a random variable can be calculated by computing the probability mass function for Y and then applying the formula for expectations. Alternatively, we can compute the expectation of a function of a random variable X directly from the probability mass function of X as

$$E[Y] = E[f(X)] = \sum_x f(x)\mathbf{P}_X(x).$$

Similarly, we can calculate the expectation for a random variable Z defined in terms of other random variables X and Y defined on the same probability space, e.g., $Z = g(X, Y)$, as computing the probability mass function for Z or directly as

$$E[Z] = E[g(X, Y)] = \sum_{x,y} g(x, y)\mathbf{P}_{X,Y}(x, y).$$

These formulas generalize to function of any number of random variables.

4 Linearity of Expectations

An important special case of functions of random variables is the linear functions. For example, let $Y = f(X) = aX + b$, where $a, b \in \mathbb{R}$.

$$\begin{aligned}
\mathbf{E}[Y] = \mathbf{E}[f(X)] &= \mathbf{E}[aX + b] \\
&= \textstyle\sum_x f(x)\mathbf{P}_X(x) \\
&= \textstyle\sum_x (ax + b)\mathbf{P}_X(x) \\
&= a\textstyle\sum_x x\mathbf{P}_X(x) + b\textstyle\sum_x \mathbf{P}_X(x) \\
&= a\mathbf{E}[X] + b.
\end{aligned}$$

Similar to the example, above we can establish that the linear combination of any number of random variables can be written in terms of the expectations of the random variables. For example, let $Z = aX + bY + c$, where X and Y are two random variables. We have

$$\mathbf{E}[Z] = \mathbf{E}[aX + bY + c] = a\mathbf{E}[X] + b\mathbf{E}[Y] + c.$$

The proof of this statement is relatively simple.

$$\begin{aligned}
\mathbf{E}[Z] &= \mathbf{E}[aX + bY + c] \\
&= \textstyle\sum_{x,y} (ax + by + c)\mathbf{P}_{X,Y}(x, y) \\
&= a\textstyle\sum_{x,y} x\mathbf{P}_{X,Y}(x, y) + b\textstyle\sum_{x,y} y\mathbf{P}_{X,Y}(x, y) + \textstyle\sum_{x,y} c\mathbf{P}_{X,Y}(x, y) \\
&= a\textstyle\sum_x \textstyle\sum_y x\mathbf{P}_{X,Y}(x, y) + b\textstyle\sum_y \textstyle\sum_x y\mathbf{P}_{X,Y}(x, y) + \textstyle\sum_{x,y} c\mathbf{P}_{X,Y}(x, y) \\
&= a\textstyle\sum_x x\textstyle\sum_y \mathbf{P}_{X,Y}(x, y) + b\textstyle\sum_y y\textstyle\sum_x \mathbf{P}_{X,Y}(x, y) + \textstyle\sum_{x,y} c\mathbf{P}_{X,Y}(x, y) \\
&= a\textstyle\sum_x x\mathbf{P}_X(x) + b\textstyle\sum_y y\mathbf{P}_Y(y) + c \\
&= a\mathbf{E}[X] + b\mathbf{E}[Y] + c.
\end{aligned}$$

An interesting consequence of this proof is that the random variables X and Y do not have to be defined on the same probability space. They can be defined for different experiments and their expectation can still be summed. To see why note that we can define the joint probability mass function $\mathbf{P}_{X,Y}(x, y)$ by taking the Cartesian product of the sample spaces of X and Y and spreading probabilities for each arbitrarily as long as the marginal probabilities, $\mathbf{P}_X(x)$ and $\mathbf{P}_Y(y)$ remain unchanged.

The property illustrated by the example above is known as the ***linearity of expectations***. The linearity of expectations is very powerful often greatly simplifying analysis. The reasoning generalizes to the linear combination of any number of random variables.

Linearity of expectation occupies a special place in probability theory, the idea of replacing random variables with their expectations in other mathematical expressions do not generalize. Probably the most basic example of this is multiplication of random variables. We might ask is $\mathbf{E}[X] \times \mathbf{E}[Y] = \mathbf{E}[X \times Y]$? It turns out it is true when X and Y are independent, but otherwise it is generally not true. To see that it is true for independent random variables we have (we assume x and y range over the values of X and Y respectively):

$$\begin{aligned}
\mathbf{E}[X] \times \mathbf{E}[Y] &= \left(\sum_x x\mathbf{P}[\{X=x\}]\right)\left(\sum_y y\mathbf{P}[\{Y=y\}]\right) \\
&= \sum_x \sum_y (xy\mathbf{P}[\{X=x\}]\,\mathbf{P}[\{Y=y\}]) \\
&= \sum_x \sum_y (xy\mathbf{P}[\{X=x\} \cap \{Y=y\}]) \quad \text{due to independence} \\
&= \mathbf{E}[X \times Y]
\end{aligned}$$

For example, the expected value of the product of the values on two (independent) dice is therefore $3.5 \times 3.5 = 12.25$.

Example 3.6. In a previous example , we analyzed the expectation on X, the sum of the two dice, by summing across all 36 elementary events. This was particularly messy for the biased dice. Using linearity of expectations, we need only calculate the expected value of each dice, and then add them. Since the dice are the same, we can in fact just multiply by two. For example for the biased case, assuming X_1 is the value of one dice:

$$\begin{aligned}
\mathbf{E}[X] &= 2\mathbf{E}[X_1] \\
&= 2 \times \textstyle\sum_{d \in \{1,2,3,4,5,6\}} d \times \frac{d}{21} \\
&= 2 \times \frac{1+4+9+16+25+36}{21} \\
&= 8\,\tfrac{2}{3}\,.
\end{aligned}$$

5 Conditional Expectation

Definition 3.1 (Conditional Expectation)**.** We define the conditional expectation of a random variable X for a given value y of Y as

$$\mathbf{E}[X \mid Y=y] = \sum_x x\mathbf{P}_{X|Y}(x \mid y).$$

Theorem 3.1 (Total Expectations Theorem)**.** The expectation of a random variable can be calculated by "averaging" over its conditional expectation given another random variable:

$$\mathbf{E}[X] = \sum_y \mathbf{P}_Y(y)\mathbf{E}[X \mid Y=y].$$

Part II

Graph Contraction and Applications

Chapter 4

Introduction

Overview. In earlier chapters, we have mostly covered techniques for solving problems on graphs that were developed in the context of sequential algorithms. Some of the algorithms we considered were parallel while others were not. For example, we saw that BFS has some parallelism since each level can be explored in parallel but there was no parallelism in DFS . There was no parallelism in Dijkstra's algorithm , but there was plenty of parallelism in the Bellman-Ford algorithm and Johnson's algorithm .

In this part of the book, we cover the "graph contraction" technique. This technique was specifically designed to be used in parallel algorithms and allows obtaining polylogarithmic span for certain graph problems. This chapter presents an overview of graph contraction. The following chapters present two specializations Edge Contraction and Star Contraction of graph contraction, and apply the technique to graph connectivity .

1 Preliminaries

Note. The material here and the followup chapters on graph contraction relies on the graph terminology introduced in the background chapter on graph theory .

Definition 4.1 (Graph Partition)**.** Given a graph G, a ***graph partition*** of G is a collection of graphs

$$H_0 = (V_0, E_0), \ldots, H_{k-1} = (V_{k-1}, E_{k-1}),$$

such that $\{V_0, \ldots, V_{k-1}\}$ is a set partition of V and $H_0, \ldots, H_{k-1}$ are vertex-induced subgraphs of G with respect to $V_0, \ldots, V_{k-1}$.

We refer to each subgraph H_i as a ***block*** or ***part*** of G.

Definition 4.2 (Internal and Cut Edges)**.** Given a partition $H_0 = (V_0, E_0), \ldots, H_{k-1} =$

(V_{k_1}, E_{k-1}) of a graph $G = (V, E)$, we define two kinds of edges: internal edges and cut edges.

- We call an edge $\{v_1, v_2\}$ an ***internal edge***, if $v_1 \in V_i$ and $v_2 \in V_i$. Note that $\{v_1, v_2\} \in E_i$.
- We call an edge $\{v_1, v_2\}$ a ***cut edge***, if $v_1 \in V_i$ and $v_2 \in V_j$ and $i \neq j$.

Exercise 4.1. One way to partition a graph is to make each connected component a block. What are the internal and cut edges in such a partition?

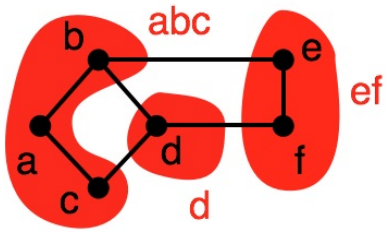
Solution. There are no cut edges between the partitions. All edges of the graph are internal edges.

Definition 4.3 (Partition Map)**.** We sometimes describe a graph partition with a tuple consisting of

1. a set of labels for the blocks, and
2. a ***partition map*** that maps each vertex to the label of its block.

The labels can be chosen arbitrarily but it is usually conceptually and computationally easier to use a vertex inside a block as a representative for that block.

Example 4.1. The partition $\{\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, \{\mathtt{d}\}, \{\mathtt{e}, \mathtt{f}\}\}$ of the vertices $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}\}$, defines three blocks as the corresponding vertex-induced subgraphs .



The edges $\{\mathtt{a}, \mathtt{b}\}$, $\{\mathtt{a}, \mathtt{c}\}$, and $\{\mathtt{e}, \mathtt{f}\}$ are internal edges, and the edges $\{\mathtt{c}, \mathtt{d}\}$, $\{\mathtt{b}, \mathtt{d}\}$, $\{\mathtt{b}, \mathtt{e}\}$ and $\{\mathtt{d}, \mathtt{f}\}$ are cut edges.

By labeling the blocks `'abc'`, `'d'` and `'ef'`, we can specify the graph partition with following partition map:

$$(\{\mathtt{abc}, \mathtt{d}, \mathtt{ef}\}, \tag{4.1}$$

$$\{\mathtt{a} \mapsto \mathtt{abc}, \mathtt{b} \mapsto \mathtt{abc}, \mathtt{c} \mapsto \mathtt{abc}, \mathtt{d} \mapsto \mathtt{d}, \mathtt{e} \mapsto \mathtt{ef}, \mathtt{f} \mapsto \mathtt{ef}\}). \tag{4.2}$$

Instead of assigning a fresh label to each block, we can choose a representative vertex. For example, by picking $\mathtt{a}, \mathtt{d}$, and $\mathtt{e}$ as representatives, we can represent the partition above using the following partition map

$$(\{\mathtt{a}, \mathtt{d}, \mathtt{e}\}, \tag{4.3}$$
$$\{\mathtt{a} \mapsto \mathtt{a}, \mathtt{b} \mapsto \mathtt{a}, \mathtt{c} \mapsto \mathtt{a}, \mathtt{d} \mapsto \mathtt{d}, \mathtt{e} \mapsto \mathtt{e}, \mathtt{f} \mapsto \mathtt{e}\}). \tag{4.4}$$

2 Graph Contraction

Graph contraction is a contraction technique for computing properties of graphs in parallel. As a contraction technique, it is used to solve a problem instance by reducing it to a smaller instance of the same problem.

Graph contraction plays important role in parallel algorithm design, because divide-and-conquer can be difficult to apply to graph problems efficiently. Divide-and-conquer techniques usually require partitioning graphs into smaller graphs in a balanced fashion such that the number of cut edges is minimized. Because graphs can be highly irregular, they can be difficult to partition. In fact, graph partitioning problems are typically NP-hard.

Quotient Graph. The key idea behind graph contraction is to contract the input graph to a smaller ***quotient graph***, solve the problem on the quotient graph, and then use that solution to construct the solution for the input graph. We can specify this technique as an inductive algorithm-design technique as follows.

Definition 4.4 (Graph-Contraction Technique)**.** Graph contraction technique has a base case and an inductive case. Each application of the inductive step is called a ***round*** of graph contraction. In a graph contraction, rounds are repeated until the graph is small, e.g., the graph has no remaining edges.

Base case: If the graph is small (e.g., it has no edges), then compute the desired result.

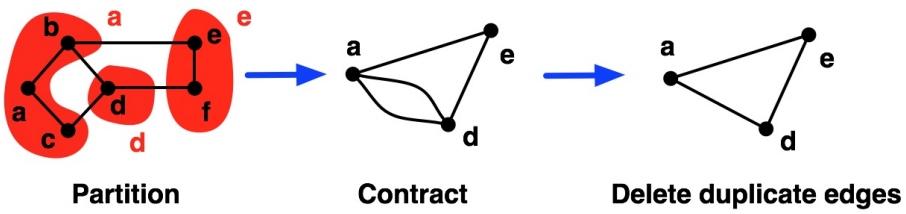
Inductive case:

- **Contraction step:** contract the graph into a smaller quotient graph.
  - Partition the graph into blocks.
  - Contract each block to a single super-vertex.
  - Drop internal edges.

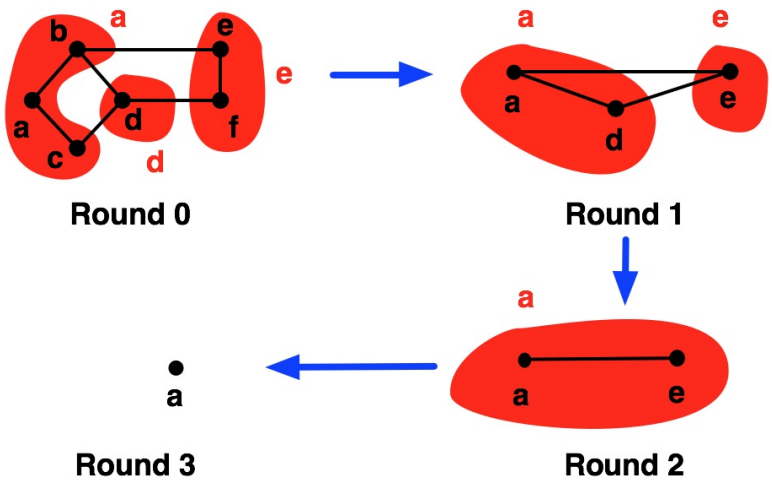
  - Reroute cut edges to corresponding super-vertices.

- **Recursive step:** Recursively solve the problem for the quotient graph.
- **Expansion step:** By using the result for the quotient graph, compute the result for the input graph.

Example 4.2. One round of graph contraction:



Contracting a graph down to a single vertex in three rounds:



Construction of the Quotient Graph. To construct a quotient graph, we represent each block in the partition with a vertex, which we call a ***super-vertex***. We then "map" the edges of the graph to the quotient graph. Consider each edge (u, v) in the graph.

- If the edge is an internal edge, then we skip the edge.
- If the edge is a cut edge, then we create a new edge between the super-vertices representing the blocks containing u and v.

Because there can be many cut edges between two blocks, this approach may create multiple edges between two super-vertices. We may remove duplicate edges or leave them in the graph, in which case we would be working with multigraphs. Either approach has its benefits and may, depending on the application, be preferable over the other.

Important. Graph contraction is guided by a graph partition, which leads to blocks whose vertices are disjoint. During the construction of the quotient graph, each vertex in the graph is therefore mapped to a unique vertex in the quotient graph.

Applying Graph Contraction. The ultimate goal of graph contraction technique is to reduce the size of the graph by a constant fraction (possibly in expectation) at each round of contraction. Depending on the graphs of interest many different graph-partition techniques can be used to achieve this goal. As described, the graph-contraction technique is generic in the kind of graph partition used. In the following chapters on Edge Contraction and Star Contraction we consider two techniques, edge partitioning and star partitioning, and the resulting graph-contraction algorithms.

Chapter 5

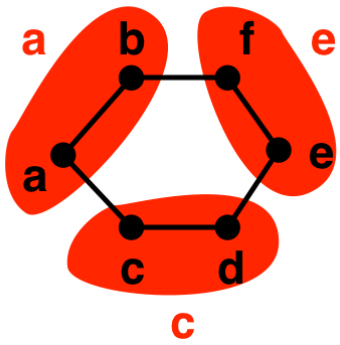
Edge Contraction

This section describes the edge partition and edge contraction. Edge contraction is an instance of a graph-contraction where blocks being contracted correspond to edges.

1 Edge Partition

Definition 5.1 (Edge Partition)**.** An ***edge partition*** is a graph partition where each block is either a single vertex or two vertices connected by an edge.

Example 5.1. An example edge partition in which every block consists of two vertices and an edge between them.



Exercise 5.1. Give an example graph whose edge partitions always contain a block that consists of a single vertex.

Solution. Any graph which has an isolated vertex, i.e., a vertex with no incident edges would work.

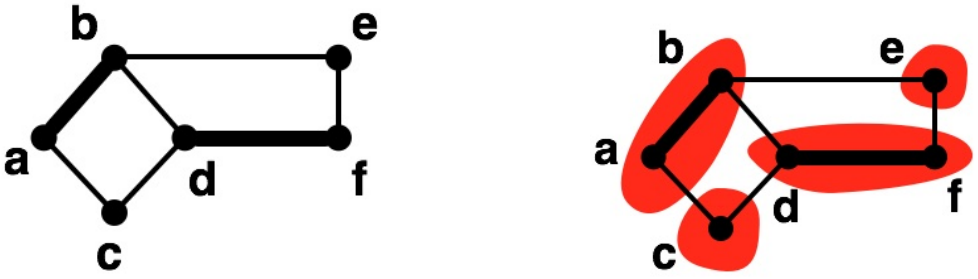
Edge Partitions and Vertex Matching. Finding an edge partition of a graph is closely related to the problem of finding an independent edge set or a vertex matching. A vertex matching in a graph is a subset of the edges that do not share an endpoint, i.e., no two edges are incident on the same vertex. We can construct an edge partition from a vertex matching by constructing a block for each edge in the matching and placing all the remaining vertices into their own singleton blocks.

Definition 5.2 (Vertex Matching)**.** A ***vertex matching*** for an undirected graph $G = (V, E)$ is a subset of edges $M \subseteq E$ such that no two edges in M are incident on the same vertex. In other words, each vertex in M have degree at most 1.

The problem of finding the largest vertex matching for a graph is called the ***maximum vertex matching*** problem.

Algorithms for Maximum Vertex Matching. Maximum Vertex Matching is a well-studied problem and many algorithms have been proposed, including one that can solve the problem in $O(\sqrt{|V|}|E|)$ work.

Example 5.2 (Vertex Matching)**.** A vertex matching for a graph (highlighted edges) and the corresponding blocks.



The vertex matching defines four blocks (circled), two of them defined by the edges in the matching, $\{\texttt{a}, \texttt{b}\}$ and $\{\texttt{d}, \texttt{f}\}$, and two of them are the unmatched vertices $\texttt{c}$ and $\texttt{e}$.

Note. For edge contraction, we do not need a maximum matching but one that it is sufficiently large.

Algorithm 5.3 (Greedy Vertex Matching)**.** We can use a greedy algorithm to construct a vertex matching by iterating over the edges while maintaining an initially empty matching M. The greedy algorithm considers each edge and proceeds as follows:

- if no edge in M is already incident on its endpoints, then the algorithm adds the edge to M,

- otherwise, the algorithm tosses away the edge.

Exercise 5.2. Does the greedy vertex matching algorithm always returns a maximum vertex matching?

Solution. No.

Exercise 5.3. Prove that the greedy algorithm finds a solution within a factor two of optimal.

Exercise 5.4. Is the greedy algorithm parallel?

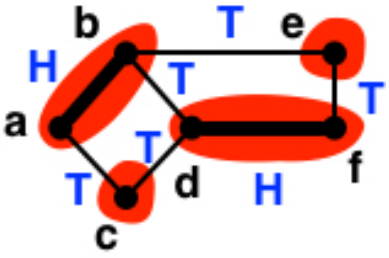
Solution. The greedy algorithm is sequential, because each decision depends on previous decisions.

Randomized Symmetry Breaking. To find a vertex matching in parallel, we want to make local and parallel decisions at each vertex independent of other vertices. One possibility is for each vertex to select one of its neighbors arbitrarily but in some deterministic fashion. Such a selection can be made in parallel but there is one problem: multiple vertices might select the same vertex to match with.

We therefore need a way to *break the symmetry* that arises when two vertices try to match with the same vertex. To this end, we can use randomization. There are several different ways to use randomization but they are all essentially the same and yield the same bounds with a constant factor.

Algorithm 5.4 (Parallel Vertex Matching)**.** To compute a vertex matching the ***parallel vertex matching algorithm*** flips a coin for each edge in parallel. The algorithm then selects an edge (u, v) and matches u and v, if the coin for the edge comes up heads and all the edges incident on u and v flip tails.

Example 5.3 (Parallel Vertex Matching)**.** An example run of he parallel vertex matching algorithm.



Exercise 5.5. Prove that the algorithm produces a vertex matching, i.e., it guarantees that a vertex is matched with at most one other vertex.

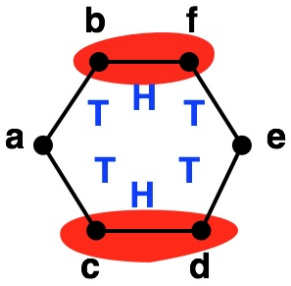
1.1 Analysis of Parallel Edge Partition

We analyze the effectiveness of the parallel edge partition algorithm in selecting a matching that consists of as many edge blocks (equivalently as few singleton blocks) as possible. We first consider cycle graphs and then general graphs.

1.1.1 Cycle Graphs

Probability of Selecting an Edge in a Cycle. We want to determine the probability that an edge is selected in a cycle, where each vertex has exactly two neighbors. Because the coins are flipped independently at random, and each vertex has degree two, the probability that an edge picks heads and its two adjacent edges pick tails is $\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{8}$.

Example 5.4 (Edge Partition of a Cycle)**.** A graph consisting of a single cycle.



Each edge flips a coin that comes up either heads (H) or tails (T). We select an edge if it turns up heads and all other edges incident on its endpoints are tails. In the example the edges $\{\texttt{c}, \texttt{d}\}$ and $\{\texttt{b}, \texttt{f}\}$ are selected.

Expected Number of Edges Selected. To analyze the number of edges (blocks) selected in expectation, let R_e be an indicator random variable denoting whether e is selected or not, that is $R_e = 1$ if e is selected and 0 otherwise. Recall that the expectation of indicator random variables is the same as the probability it has value 1 (true). Therefore we have $E[R_e] = 1/8$. Thus summing over all edges, we conclude that expected number of edges selected is $\frac{m}{8}$ (note, $m = n$ in a cycle graph). Thus we conclude that in expectation, a constant fraction $\left(\frac{1}{8}\right)$ of the edges are selected to be their own blocks.

Exercise 5.6. Modify the algorithm to improve the expected number of edges selected.

Improving the Expectation. There are several ways to improve the number of select edges. One way is for each vertex to pick one of its neighbors and to select an edge (u, v) if

it was picked by both u and v. In the case of a circle, this increases the expected number of selected edges to $\frac{m}{4}$.

Another way is let each edge pick a random number in some range and then select and edge if it is the local maximum, i.e., it picked the highest number among all the edges incident on its end points. This increases the expected number of selected edges to $\frac{m}{3}$.

1.1.2 Star Graphs

Limitation of Edge Partition. Although our edge partition algorithm works quite well on cycle graphs, it does not work well for arbitrary graphs. The problem is in an edge partition, only one edge incident on a vertex can be its own block. Therefore if there is a vertex with high degree, then only one of its edges can be selected. Star graphs are a canonical example of such graphs, although there are many others.

Definition 5.5 (Star Graph)**.** A ***star graph*** $G = (V, E)$ is an undirected graph with

- a ***center*** vertex $v \in V$, and
- a set of edges E that attach v directly to the rest of the vertices, called ***satellites***, i.e., $E = \{\{v, u\} : u \in V \setminus \{v\}\}$.

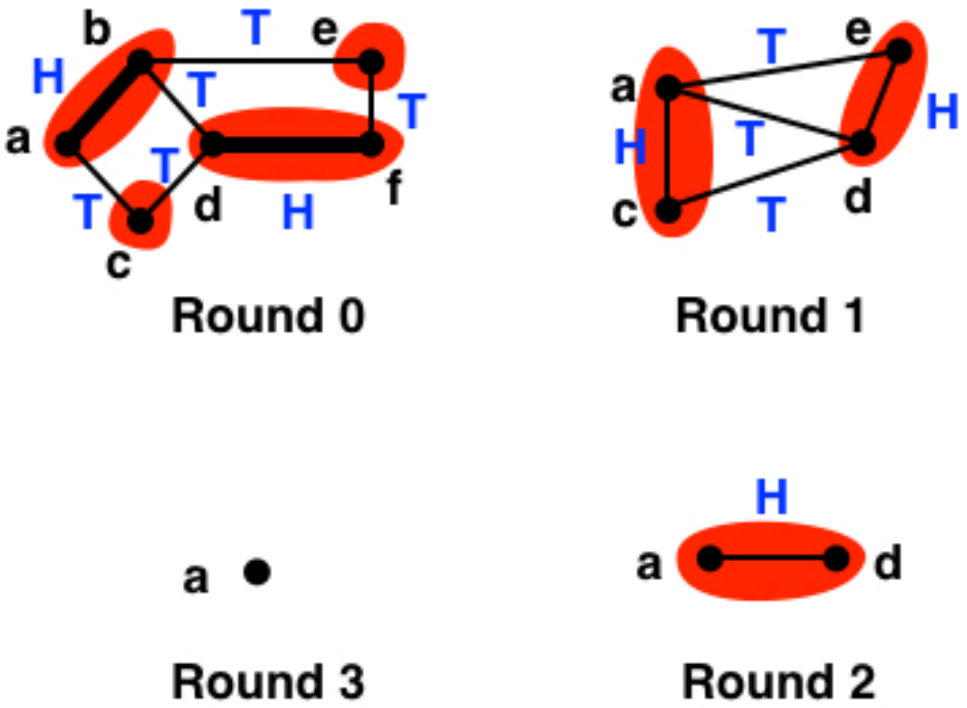
Example 5.5. The following are star graphs:

- a single vertex, and
- a single edge.

2 Edge Contraction

Algorithm 5.6 (Parallel Edge Contraction)**.** Parallel edge contraction algorithm is a specialization of the graph contraction technique that uses the parallel vertex matching algorithm to partition the graph for contraction.

Example 5.6 (Edge contraction)**.** An example parallel edge contraction illustrated.



Analysis of Edge Contraction. The analysis of edge partition established that using edge partition, we are able to select in expectation $\frac{1}{8}$ of the edges as their own blocks if the graph is a cycle. Therefore, after one round of contraction, the number of vertices and edges in a cycle decrease by an expected constant fraction.

In randomized algorithms chapter , we showed that if each round of an algorithm reduces the size by a constant fraction in expectation, and if the random choices in the rounds are independent, then the algorithm will finish in $O(\lg n)$ rounds with high probability. Recall that all we needed to do is multiply the expected fraction that remain across rounds and then use Markov's inequality to show that after some $k \lg n$ rounds the probability that the problem size is a least 1 is very small. For a cycle graph, this technique leads to an algorithm for graph contraction with linear work and $O(\lg^2 n)$ span.

Analysis for Star Graphs. Edge contraction works quite poorly on other graphs such as star graphs, and can result in a partition with many singleton blocks. This is because in an edge partition, only one of the edges incident on a vertex can be its own block (Section 1.1.2), leading to a poor contraction ratio. Edge contraction therefore is not effective for general graphs.

Chapter 6

Star Contraction

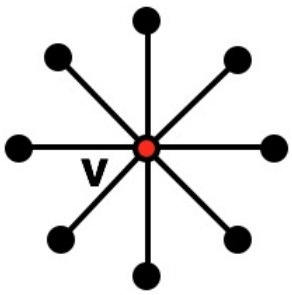
This chapter covers star partition and star contraction, an efficient and parallel graph-contraction technique for general graphs.

1 Star Partition

In an edge partition , if an edge incident on a vertex v is selected as a block, then none of the other edges incident on v can be their own block. This limits the effectiveness of the edge partition technique, because it is unable to contract graphs with high-degree vertices significantly. In this section, we describe an alternative technique, star partition, that does not have this limitation.

Definition 6.1 (Star Partition)**.** A ***star partition*** of a graph G is a partition of G where each block is vertex-induced subgraph with respect to a star graph .

Example 6.1. Consider star graph with center v and eight satellites.

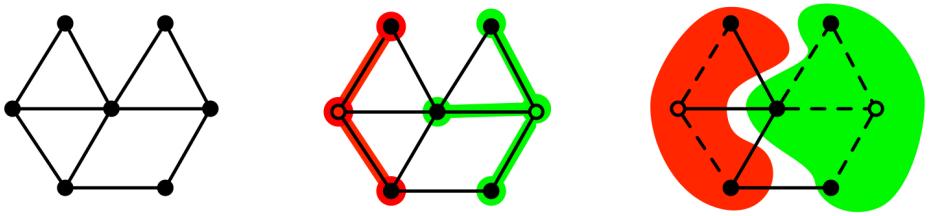


- A partition consisting of the whole graph is a star partition, where the only block is

the graph itself, induced by the star graph.

- A partition where each block is an isolated vertex is a star partition, because each block is a vertex-induced subgraph of a single vertex, which is a star.

Example 6.2. Consider the graph shown below on the left. To partition this graph, we first find two disjoint stars, which are highlighted. Each star induces a block consisting of its vertices and the corresponding edges of the graph. These two blocks form a star partition the graph. Note that in a star partition, a block might not be a star.



A graph

Its two stars (highlighed).

The corresponding star partition. Solid edges are cut edges. Dashed edges are internal edges.

Constructing a Star Partition (Sequential). We can construct a star partition sequentially by iteratively adding stars until the vertices are exhausted as follows.

- Select an arbitrary vertex v from the graph and make v the center of a star.
- Attach as satellites all the neighbors of v in the graph.
- Remove v and its satellites from the graph.

Computing a Star Partition (Parallel). We can construct a star partition in parallel by making local independent decisions for each vertex, and using randomization to break symmetry. One approach proceeds as follows.

- Flip a coin for each vertex.
- If a vertex flips heads, then it becomes the center of a star.
- If a vertex flips tails, then there are two cases.
  - The vertex has a neighbor that flips heads and the vertex selects the neighbor (breaking ties arbitrarily). In this case, the vertex becomes a satellite.
  - The vertex doesn't have a neighbor that flips heads and it becomes a center.

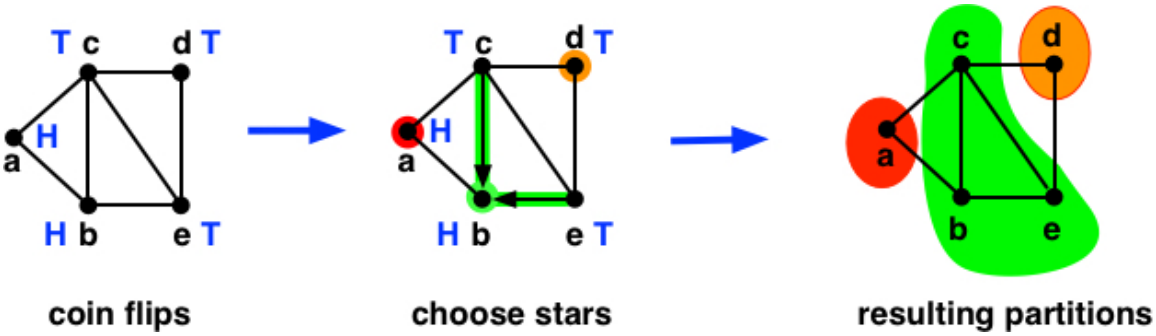
Note that if a vertex doesn't have a neighbor (it is "isolated"), then it will always become a center.

Definition 6.2 (Isolated Vertices)**.** We say that a vertex is ***isolated*** in a graph if it doesn't have a neighbor.

Note. The parallel approach to star partition is not optimal, because it might not always create the smallest number of stars. This is acceptable for us, because we only need to reduce the size of the graph by some constant factor.

Example 6.3 (Randomized Star Partition)**.** The example below illustrates how we may partition a graph using the parallel star partition algorithm described above. Vertices `a` and `b`, which flip heads, become centers. Vertices `c` and `e`, which flipped tails, attempt to become satellites by finding a center among their neighbors, breaking ties arbitrarily. If a vertex does not have a neighbor that is a center (flipped heads), then it becomes a singleton star (e.g., vertex d).

The resulting star partition has three stars: the star with center `a` (with no satellites), the star with center `b` (with two satellites), and the singleton star `d`. The star partition thus yields three blocks, which are defined by the subgraphs induced by each star.



Algorithm 6.3 (Parallel Star Partition)**.** To specify the star-partition algorithm, we need a source of randomness. We assume that each vertex is given a (potentially infinite) sequence of random and independent coin flips. The i^{th} element of the sequence can be accessed via the function

$$\texttt{heads}\ (v, i) : V \times \mathbb{Z} \to \mathbb{B},$$

which returns `true` if the i^{th} flip on vertex v is heads and false otherwise.

The function *starPartition*, whose pseudo-code is given below, takes as argument a graph and a round number, and returns a graph partition specified by a set of centers and a partition map from all vertices to centers.

The algorithm starts by flipping a coin for each vertex and selecting the edges that point from tails to heads—this gives the set of edges TH. In this set of edges, there can be

multiple edges from the same non-center. Since we want to choose one center for each satellite, we remove duplicates in Line 6, by creating a set of singleton tables and merging them, which selects one center per satellite. This completes the selection of satellites and their centers.

Next, the algorithm determines the set of centers as all the non-satellite vertices. To complete the process, the algorithm maps each center to itself (Line 10). These operations effectively promote unmatched non-centers to centers, forming singleton stars, and matches all centers with themselves. Finally, the algorithm constructs the partition map by uniting the mapping for the satellites and the centers.

```
1   starPartition (G = (V, E), i) =
2     let
3       (* Find the arcs from satellites to centers.  *)
4       TH = {(u, v) ∈ E | ¬heads(u, i) ∧ heads(v, i)}
5       (* Partition map:  satellites map to centers *)
6       P_s = ⋃_{(u,v)∈TH} {u ↦ v}
7       (* Centers are non-satellite vertices *)
8       V_c = V \ domain(P_s)
9       (* Map centers to themselves *)
10      P_c = {u ↦ u : u ∈ V_c}
11    in
12      (V_c, P_s ∪ P_c)
13    end
```

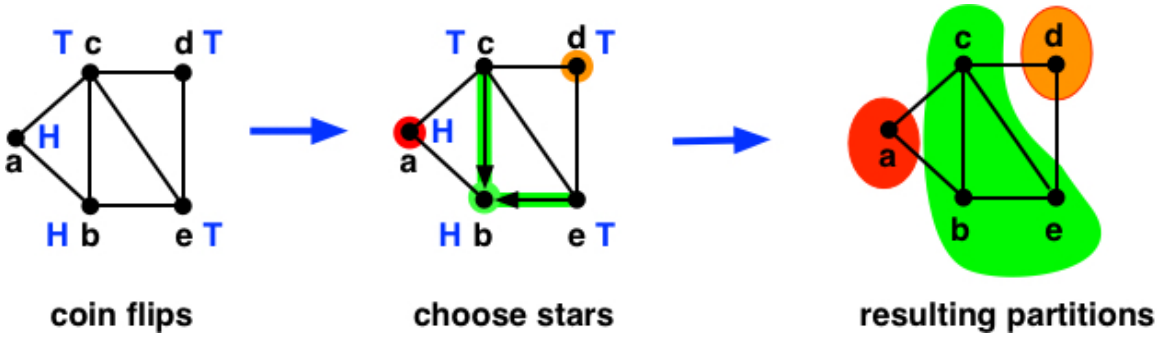
Note. Since most machines don't have true sources of randomness, in practice the function *heads* is usually implemented with a pseudorandom number generator or with a good hash function.

In the algorithm, Line 6 creates a set of singleton tables and merges them. This can be implemented using sets and tables as follows.

```
Set.reduce   (Table.union (lambda (x, y) . x))
             ∅
             {{u ↦ v} : (u, v) ∈ TH}
```

Note that we supply to the *union* operation a function that selects the first of the two possibilities; this is an arbitrary choice and we could have favored the second.

Example 6.4. Consider the star partition illustrated below.



The star-partition algorithm proceeds on this example as follows. First, it computes

$$TH = \{(\mathtt{c},\mathtt{a}),(\mathtt{c},\mathtt{b}),(\mathtt{e},\mathtt{b})\}\,,$$

as the edges from satellites to centers. Now, it converts each edge into a singleton table, and merges all the tables into one table, which is going to become a part of the partition map:

$$P_s = \{\mathtt{c} \mapsto \mathtt{b}, \mathtt{e} \mapsto \mathtt{b}\}\,.$$

Note that the edge $(\mathtt{c},\mathtt{a})$ has been removed since when uniting the tables, we select only one element for each key in the domain. Now for all remaining vertices $V' = V \backslash domain(P) = \{\mathtt{a},\mathtt{b},\mathtt{d}\}$ we map them to themselves, giving:

$$P_c = \{\mathtt{a} \mapsto \mathtt{a}, \mathtt{b} \mapsto \mathtt{b}, \mathtt{d} \mapsto \mathtt{d}\}\,.$$

The vertices in P' are the centers. Finally we merge P and P' to obtain the partition map

$$P_s \cup P_c = \{\mathtt{a} \mapsto \mathtt{a}, \mathtt{b} \mapsto \mathtt{b}, \mathtt{c} \mapsto \mathtt{b}, \mathtt{d} \mapsto \mathtt{d}, \mathtt{e} \mapsto \mathtt{b}\}\,.$$

Implementation. Suppose that we are given an enumerable graph with n vertices and m edges. We can represent the graph using an edge set representation and represent the sets with sequences. This means that we have a sequence of vertices and a sequence of edges.

This representation enables a relatively clean implementation of the star-partition algorithm , as shown by the pseudo-code below. The implementation follows the pseudo-code for the algorithm but is able to compute the satellites and centers compactly by using a sequence *inject* operation. The implementation first constructs a vertex sequence V' where each vertex maps to itself. It then constructs a sequence TH of "updates" from vertices that flip heads into tails, and inject TH into the sequence of vertices V'. The resulting sequence P maps each vertex that flipped tails to a center, if the vertex has a neighbor that flipped heads. The sequence P ensures that a vertex that has flipped heads remains unaffected by the injection, e.g., if vertex i has flipped heads, then $P[i] = i$. We can thus compute set of centers by filtering over P and use the sequence P to represent the partition map for

satellites and centers jointly.

$$
\begin{array}{l}
\mathit{starPartition}\ (G = (V, E), i) = \\
\quad \texttt{let} \\
\quad\quad V' = \langle\, i : 0 \leq i < |V| \,\rangle \\
\quad\quad TH = \langle\, (u, v) \in E \mid \neg \texttt{heads}\ (u, i), \texttt{heads}\ (v, i) \,\rangle \\
\quad\quad P = \mathit{Seq.inject}\ V'\ TH \\
\quad\quad V_C = \mathit{Seq.filter}\ (\texttt{lambda}\ i.\ P[i] = i)\ \mathit{Pin}(V_C, P) \mathit{end}
\end{array}
$$

1.1 Analysis of Star Partition

Theorem 6.1 (Cost of Star Partition)**.** Based on the array-based cost specification for sequences, the cost of $\mathit{starPartition}$ is

$$O(n + m)$$

work and

$$O(\lg n)$$

span for a graph with n vertices and m edges.

Exercise 6.1. Prove the theorem.

Number of Satellites. Let us also bound the number of satellites found by $\mathit{starPartition}$. Note first that there is a one-to-one mapping between the satellites and the set P_s computed by the algorithm. The following lemma establishes that on a graph with $n_\bullet$ non-isolated vertices, the number of satellites is at least $n_\bullet/4$ in expectation. As we will see this means that we can use star partition to perform graph contraction with logarithmic span.

Lemma 6.2 (Number of Satelites)**.** For a graph G with $n_\bullet$ non-isolated vertices, the expected number of satellites in a call to $\mathit{starPartition}\ (G, i)$ with any i is at least $n_\bullet/4$.

Proof. For any vertex v, let H_v be the event that a vertex v comes up heads, T_v that it comes up tails, and R_v that $v \in \mathit{domain}(P)$ (i.e, it is a satellite). Consider any non-isolated vertex $v \in V(G)$. By definition, we know that a non-isolated vertex v has at least one neighbor u. So, we know that $T_v \wedge H_u$ implies R_v, since if v is a tail and u is a head v must either join u's star or some other star. Therefore, $\mathbf{P}\left[R_v\right] \geq \mathbf{P}\left[T_v\right]\mathbf{P}\left[H_u\right] = 1/4$. By the linearity of expectation, the expected number of satellites is

$$
\begin{aligned}
\mathbf{E}\left[\sum_{v : v \text{ non-isolated}} \mathbb{I}\left\{R_v\right\}\right] &= \sum_{v : v \text{ non-isolated}} \mathbf{E}\left[\mathbb{I}\left\{R_v\right\}\right] \\
&\geq n_\bullet/4.
\end{aligned}
$$

The final inequality follows because we have $n_\bullet$ non-isolated vertices and because the expectation of an indicator random variable is equal to the probability that it takes the value 1. □

2 Star Contraction

Definition 6.4 (Star Contraction)**.** ***Star contraction*** is an instance of graph contraction that uses star partitions to contract the graph.

Algorithm 6.5 (Star Contraction)**.** The pseudo-code below gives a higher-order star-contraction algorithm. The algorithm takes as arguments the graph G and two functions:

- $base$ function specifies the computation in the base case, and
- $expand$ function computes the result for the larger graph from the quotient graph.

In the base case, the graph contains no edges and the function $base$ is called on vertex set.

In the recursive case, the graph is partitioned by a call to star partition (Line 6), which returns the set of (centers) super-vertices V' and P the partition map mapping every $v \in V$ to a $v' \in V'$. The set V' defines the super-vertices of the quotient graph. Line 7 computes the edges of the quotient graph by routing the end-points of each edge in E to the corresponding super-vertices in V' as specified by partition-map P. Note that the filter $P[u] \neq P[v]$. removes self edges. The algorithm then recurs on the quotient graph (V', E'). The algorithm then computes the result for the whole graph by calling the function $expand$ on the result of the recursive call R.

```
1   starContract base expand (G = (V, E)) =
2     if |E| = 0 then
3       base (V)
4     else
5       let
6         (V', P) = starPartition (V, E)
7         E' = {(P[u], P[v]) : (u, v) ∈ E | P[u] ≠ P[v]}
8         R = starContract base expand (V', E')
9       in
10        expand (V, E, V', P, R)
11      end
```

Theorem 6.3 (Work and Span of Star Contraction)**.** For a graph $G = (V, E)$, we can contract the graph into a number of isolated vertices in $O\left((|V| + |E|) \lg |V|\right)$ work and $O(\lg^2 |V|)$ span.

For the proof, we will consider work and span separately and assume that

- function $base$ has constant span and linear work in the number of vertices passed as argument, and
- function $expand$ has linear work and logarithmic span in the number of vertices and edges at the corresponding step of the contraction.

Span of Star Contraction. Let $n_\bullet$ be the number of non-isolated vertices. In star contraction, once a vertex becomes isolated, it remains isolated until the final round, since contraction only removes edges. Let $n'_\bullet$ denote the number of non-isolated vertices after one round of star contraction. We can write the following recurrence for the span of star contraction.

$$S(n_\bullet) = \begin{cases} S(n'_\bullet) + O(\lg n) & \text{if} \quad n_\bullet > 0 \\ 1 & \text{otherwise.} \end{cases}$$

Observe that $n'_\bullet = n_\bullet - X$, where X is the number of satellites (as defined earlier in the lemma about $starPartition$), which are removed at a step of contraction. Since $\mathbf{E}[X] = n_\bullet/4$, $\mathbf{E}[n'_\bullet] = 3n/4$. This is a familiar recurrence, which we know solves to $O(\lg^2 n_\bullet)$, and thus $O(\lg^2 n)$, in expectation.

Work of Star Contraction. For work, we would like to show that the overall work is linear, because we might hope that the graph size is reduced by a constant fraction on each round. Unfortunately, this is not the case. Although we have shown that star contraction can remove a constant fraction of the non-isolated vertices in one round, we have not bounded the number of edges removed.

Because removing a satellite also removes the edge that attaches it to its star's center, each round removes at least as many edges as vertices. But this does not help us bound the number of edges removed by a linear function of m, because there can be as many an n^2 edges in the graph.

To bound the work, we will consider non-isolated and isolated vertices separately. Let $n'_\bullet$ denote the number of non-isolated vertices after one round of star contraction. For the non-isolated vertices, we have the following work recurrence:

$$W(n_\bullet, m) \leq \begin{cases} W(n'_\bullet, m) + O(n_\bullet + m) & \text{if} \quad n_\bullet > 1 \\ 1 & \text{otherwise.} \end{cases}$$

This recursion solves to

$$\mathbf{E}[W(n_\bullet, m)] = O(n_\bullet + m \lg n_\bullet) = O(n + m \lg n).$$

To bound the work on isolated vertices, we note that there at most n of them at each round and thus, the additional work is $O(n \lg n)$.

We thus conclude that the total work is

$$O((n + m) \lg n).$$

Note. Consider as an example a star contraction where n and m have the following values

in each round.

round	vertices	edges
1	n	m
2	$n/2$	$m - n/2$
3	$n/4$	$m - 3n/4$
4	$n/8$	$m - 7n/8$

It is clear that the number of edges does not drop below $m - n$, so if there are $m > 2n$ edges to start with, the overall work will be $O(m \lg n)$.

Chapter 7

Graph Connectivity

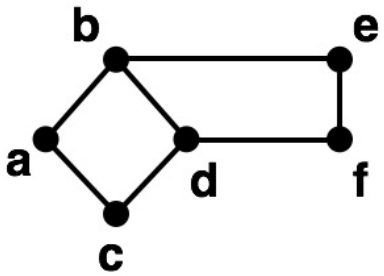
This chapter presents a parallel graph connectivity algorithm that uses graph contraction (more specifically star contraction).

1 Preliminaries

Definition 7.1 (The Graph Connectivity Problem)**.** Given an undirected graph $G = (V, E)$, the ***graph-connectivity problem*** requires finding all of the connected components of G by specifying the set of vertices in each component.

Assumption (Graph Representation). Throughout this chapter, we use an edge-set representation for graphs, where every edge is represented as a pair of vertices, in both orders. This is effectively equivalent to a directed graph representation of undirected graphs with two arcs per edge. As usual we use n and m to denote the number of vertices and edges respectively.

Example 7.1. The edge-set representation of an undirected graph is shown below.



$$
\begin{aligned}
V &= \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}, \texttt{e}, \texttt{f}\} \\
E &= \{(\texttt{a}, \texttt{b}), (\texttt{b}, \texttt{a}), (\texttt{b}, \texttt{d}), (\texttt{b}, \texttt{e}), (\texttt{e}, \texttt{b}), (\texttt{d}, \texttt{b}), (\texttt{d}, \texttt{f}), (\texttt{a}, \texttt{c}), \\
&\quad (\texttt{c}, \texttt{a}), (\texttt{c}, \texttt{d}), (\texttt{d}, \texttt{c}), (\texttt{d}, \texttt{f}), (\texttt{f}, \texttt{d}), (\texttt{e}, \texttt{f}), (\texttt{f}, \texttt{e})\}
\end{aligned}
$$

2 Algorithms for Connectivity

Sequential Algorithms for Connectivity. The graph connectivity problem can be solved by using graph search as follows.

- Start at any vertex and find, using DFS or BFS, all vertices reachable from that vertex and mark them visited. This creates the first connected component.
- Select another vertex, and if it has not already been visited, then search from that vertex to create the second component. Repeat until all the vertices are considered.

This approach leads to perfectly sensible sequential algorithms for graph connectivity, but the resulting algorithms have large span and are therefore poor parallel algorithms.

DFS is a purely sequential algorithm and has span $\Omega(m)$. BFS can yield some parallelism but still the span of BFS is lower bounded by the diameter of a component, which can be as large as $n-1$, e.g., a "chain" of n vertices has diameter $n-1$. Even when the diameter of the graph is small, the span can be high, because we have to iterate over the components sequentially. The span can is lower bounded by the number of components, which can be large.

Algorithm 7.2 (Component Count)**.** The pseudo-code below shows a graph-contraction based algorithm for determining the number of connected components in a graph. The call to *starPartition* (Algorithm 6.3) on Line 6 returns the set of (centers) super-vertices V' and a table P mapping every $v \in V$ to a $v' \in V'$.

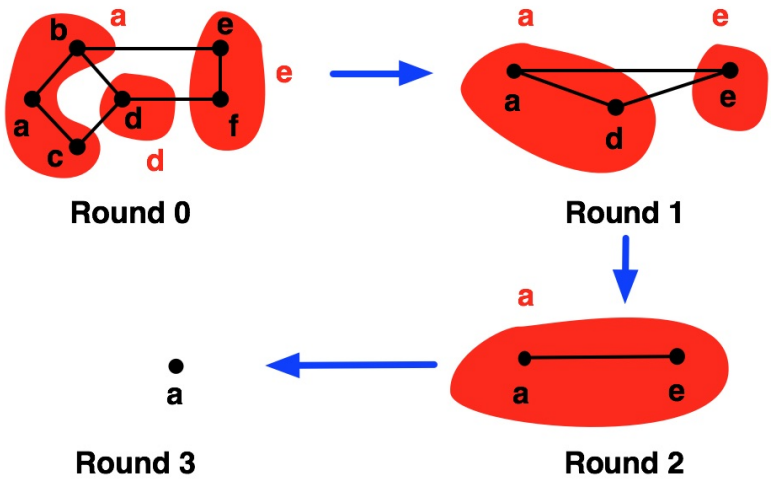
The set V' defines the super-vertices of the quotient graph. Line 7 completes the computation of the quotient graph.

- It computes the edges of the quotient graph by routing the end points of each edge to the corresponding super-vertices in V', which is specified by the table P;
- It removes all self edges via the filter $P[u] \neq P[v]$.

The algorithm then recursively computes the number of connected components in the quotient graph. Recursion bottoms out when the graph contains no edges, where the number of components is equal to the number of vertices.

```
1   countComponents (G = (V, E)) =
2     if |E| = 0 then
3       |V|
4     else
5       let
6         (V', P) = starPartition (V, E)
7         E' = {(P[u], P[v]) : (u, v) ∈ E | P[u] ≠ P[v]}
8         R = countComponents (V', E')
9       in
10        R
11      end
```

Example 7.2. Consider an execution of *countComponents* that contracts the graph as follows.



The values of V', P, and E' after each round of the contraction is shown below.

$$\begin{array}{lll} & V' & = \{\mathtt{a},\mathtt{d},\mathtt{e}\} \\ \text{Round } 1 & P' & = \{\mathtt{a}\mapsto\mathtt{a},\mathtt{b}\mapsto\mathtt{a},\mathtt{c}\mapsto\mathtt{a},\mathtt{d}\mapsto\mathtt{d},\mathtt{e}\mapsto\mathtt{e},\mathtt{f}\mapsto\mathtt{e}\} \\ & E' & = \{(\mathtt{a},\mathtt{e}),(\mathtt{e},\mathtt{a}),(\mathtt{a},\mathtt{d}),(\mathtt{d},\mathtt{a}),(\mathtt{d},\mathtt{e}),(\mathtt{e},\mathtt{d})\} \\ & V' & = \{\mathtt{a},\mathtt{e}\} \\ \text{Round } 2 & P' & = \{\mathtt{a}\mapsto\mathtt{a},\mathtt{d}\mapsto\mathtt{a},\mathtt{e}\mapsto\mathtt{e}\} \\ & E' & = \{(\mathtt{a},\mathtt{e}),(\mathtt{e},\mathtt{a})\} \\ & V' & = \{\mathtt{a}\} \\ \text{Round } 3 & P' & = \{\mathtt{a}\mapsto\mathtt{a},\mathtt{e}\mapsto\mathtt{a}\} \\ & E' & = \{\} \end{array}$$

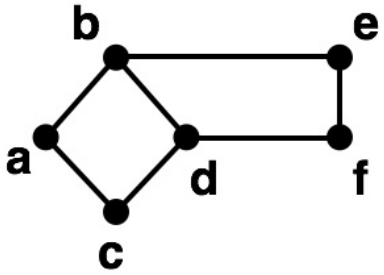
Exercise 7.1. Express `countComponents` in terms of higher order function `starContract` (Algorithm 6.5) by specifying the functions `base` and `expand`.

Computing Components. We can modify algorithm for counting components to compute the components themselves. The idea is to construct recursively a mapping from vertices to their components. The algorithm below implements this idea.

Algorithm 7.3 (Graph Connectivity)**.** The algorithm below computes the connected components of the input graph G and returns a tuple consisting of 1) a representative for each component, and 2) a mapping from the vertices in the graph to the representative of their component.

```
1   connectedComponents (G = (V, E)) =
2     if |E| = 0 then
3       (V, {v ↦ v : v ∈ V})
4     else
5       let
6         (V', P) = starPartition (V, E)
7         E' = {(P[u], P[v]) : (u, v) ∈ E | P[u] ≠ P[v]}
8         (V'', C) = connectedComponents (V', E')
9       in
10        (V'', {u ↦ C[v] : (u ↦ v) ∈ P})
11      end
```

Example 7.3. Applying `connectedComponents` to the following graph

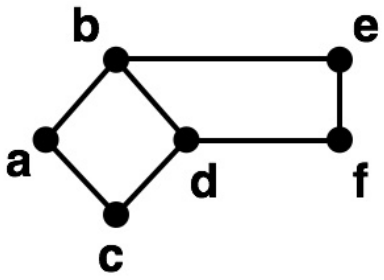


might return:

$$(\{\mathtt{a}\},\ \{\mathtt{a} \mapsto \mathtt{a}, \mathtt{b} \mapsto \mathtt{a}, \mathtt{c} \mapsto \mathtt{a}, \mathtt{d} \mapsto \mathtt{a}, \mathtt{e} \mapsto \mathtt{a}, \mathtt{f} \mapsto \mathtt{a}\})$$

This is because there is a single component and all vertices will map to that component label. In this case `a` was picked as the representative, but any of the initial vertices is a valid representative, in which case all vertices would map to it.

Example 7.4. Consider the following graph.



Suppose that `starPartition` returns:

$$\begin{aligned} V' &= \{\mathtt{a}, \mathtt{d}, \mathtt{e}\} \\ P &= \{\mathtt{a} \mapsto \mathtt{a}, \mathtt{b} \mapsto \mathtt{a}, \mathtt{c} \mapsto \mathtt{a}, \mathtt{d} \mapsto \mathtt{d}, \mathtt{e} \mapsto \mathtt{e}, \mathtt{f} \mapsto \mathtt{e}\} . \end{aligned}$$

This pairing corresponds to the case where a, d and e are chosen an centers.

Because the graph is connected, the recursive call to `connectedComponents` (V', E') will map all vertices in V' to the same vertex. Lets say this vertex is `a` giving:

$$\begin{aligned} V'' &= \{\mathtt{a}\} \\ P' &= \{\mathtt{a} \mapsto \mathtt{a}, \mathtt{d} \mapsto \mathtt{a}, \mathtt{e} \mapsto \mathtt{a}\} . \end{aligned}$$

Now $\{u \mapsto P'[v] : (u \mapsto v) \in P\}$ will for each vertex-super-vertex pair in P, look up what that super-vertex got mapped to in the recursive call. For example, vertex `f` maps to vertex `e` in P so we look up `e` in P', which gives us `a` so we know that `f` is in the component `a`. Overall the result is:

$$\{\mathtt{a} \mapsto \mathtt{a}, \mathtt{b} \mapsto \mathtt{a}, \mathtt{c} \mapsto \mathtt{a}, \mathtt{d} \mapsto \mathtt{a}, \mathtt{e} \mapsto \mathtt{a}, \mathtt{f} \mapsto \mathtt{a}\} .$$

Note. The only differences between the algorithm for counting components and the algorithm for computing the components is the base case, and "expansion step" (Definition 4.4) on Line 10 of Algorithm 7.3.

In the base case instead of returning the size of V returns all vertices in V along with a mapping from each one to itself. This is a valid answer since if there are no edges each vertex is its own component. In the inductive case, before returning from the recursion, Line 10 updates the mapping P from vertices to super-vertices by looking up the component that the super-vertex belongs to, which is given by C. This involves looking up $C[u]$ for every $(u \mapsto v) \in P$. If we view a mapping as a function, then the expansion step is equivalent to function composition, i.e., $C \circ P$.

Exercise 7.2. Express `countComponents` in terms of higher order function `starContract` (Algorithm 6.5) by specifying the functions `base` and `expand`.

Exercise 7.3. What is the work and span of the algorithm for counting components ? Explain your choice of the representation for the graph. What happens if you choose a different representation?

Exercise 7.4. What is the work and span of the algorithm for computing the components ? Explain your choice of the representation for the graph. What happens if you choose a different representation?