

CHAPTER THREE

Genome reconstruction: a puzzle with a billion pieces

Phillip E. C. Compeau and Pavel A. Pevzner

While we can read a book one letter at a time, biologists still lack the ability to read a DNA sequence one nucleotide at a time. Instead, they can identify short fragments (approximately 100 nucleotides long) called *reads*; however, they do not know where these reads are located within the genome. Thus, assembling a genome from reads is like putting together a giant puzzle with a billion pieces, a formidable mathematical problem. We introduce some of the fascinating history underlying both the mathematical and the biological sides of DNA sequencing.



1 Introduction to DNA sequencing

1.1 DNA sequencing and the overlap puzzle

Imagine that every copy of a newspaper has been stacked inside a wooden chest. Now imagine that chest being detonated. We will ask you to further suspend your disbelief and assume that the newspapers are not all incinerated, as would assuredly happen in real life, but rather that they explode cartoonishly into tiny pieces of confetti (Figure 3.1). We will concern ourselves only with the immediate journalistic problem at hand: what did the newspaper say?

This “newspaper problem” becomes intellectually stimulating when we realize that it does not simply reduce to gluing the remnants of newspaper as we would fit together the disjoint pieces of a jigsaw puzzle. One reason why this is the case is that we

Bioinformatics for Biologists, ed. P. Pevzner and R. Shamir. Published by Cambridge University Press.
© Cambridge University Press 2011.

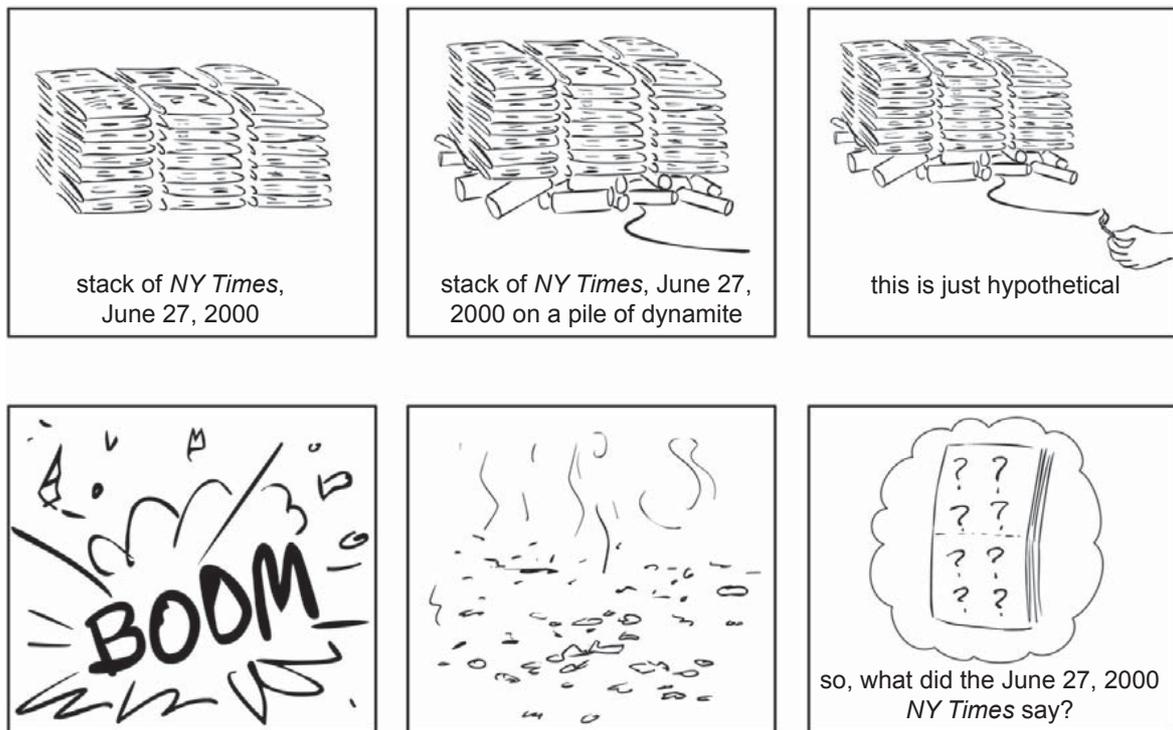


Figure 3.1 The exploding newspapers.

have probably lost some information from each copy (the content that was blown to smithereens). However, we can also see that because the chest contained many identical copies of the same newspaper, different shreds of paper may overlap and therefore contain some of the same information. The newspaper problem therefore induces what we will call an *overlap puzzle*.

We reiterate that our analogy of exploding newspapers is far-fetched, but the newspaper problem nevertheless captures the essence of fragment assembly in DNA sequencing. The technology for “reading” an entire genome nucleotide by nucleotide, like reading a newspaper one letter at a time, remains unknown. At the same time, researchers can indirectly interpret short sequences of DNA, which are referred to as *reads*; the most popular modern technology produces reads that are only 100 nucleotides long (Figure 3.2). The idea behind DNA sequencing, then, is to generate many reads from multiple copies of the same genome, which results in a giant overlap puzzle. For instance, a three billion-nucleotide mammalian genome requires an overlap puzzle with a billion (overlapping) pieces, the largest such puzzle ever assembled.

The problem of genome sequencing therefore reduces to *read generation* (a biological problem) and *fragment assembly* (an algorithmic problem). Read generation

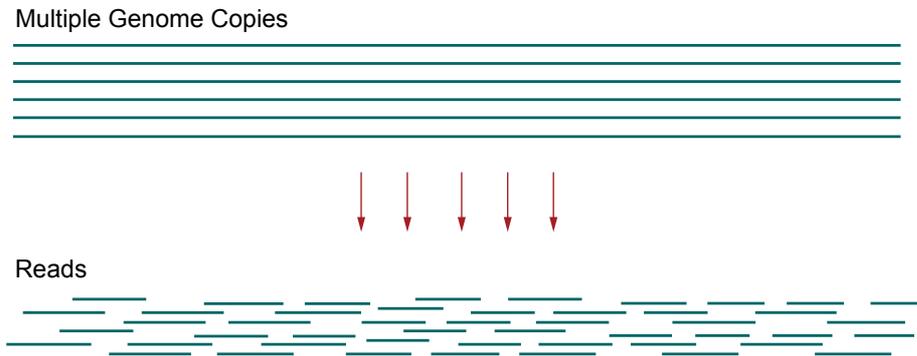


Figure 3.2 In DNA sequencing, multiple (typically more than a billion) copies of a genome are broken in random locations to generate much shorter reads.

has its own long and tangled history that dates to the 1970s, when Walter Gilbert and Fred Sanger won the Nobel Prize for inventing the first read generation technology. In the early 1990s, modern DNA sequencing machines hit the market and the era of high-throughput DNA sequencing began. In 2000, a few hundred such machines working around the clock for over a year eventually generated enough reads to enable the fragment assembly of the human genome, which was completed within a few months by some of the world’s most powerful supercomputers.

1.2 Complications of fragment assembly

Although we shall discuss read generation in some detail at the end of the chapter, our primary target is the computational problem of fragment assembly, or using the generated reads to infer the original genome.

We begin by noting that although we have seen that both the newspaper problem and fragment assembly reduce to solving an overlap puzzle, fragment assembly is substantially more difficult for several reasons, and not simply because of the sheer scale of reconstructing a genome from a billion reads. First, keep in mind that a newspaper is written in some understood language, whose rules will provide us with context clues as to how different shreds of paper may or may not be connected, regardless of whether these shreds overlap (see Figure 3.3a). Yet the rules for the “language” of DNA still mostly elude biologists, and so it is practically impossible to determine how two non-overlapping reads might be connected.

A second complication of fragment assembly is that the underlying nucleotide “alphabet” for DNA contains only four letters: A, T, G, and C. Working with a small

- (a) ...oodie , app... 2" 1c
...e have not yet named any suspects, alt
...e murder occurred at approximately 5:2...
...ation is welc... e ca
- (b) ...e world's mos...
...zone as a contributing facto...
...mentalists ha...ed low levels of oz...
...t the... as a cr
- (c)
- 
- (d)
- TAGGCCATGTCAGATG
CATGTCAGATGCGTAG

Figure 3.3 Complications of fragment assembly. (a) In the newspaper assembly problem, we can see that even though these two shreds do not overlap they are nevertheless probably connected, because we know that "murder" and "suspect" are highly correlated words. (b) In the newspaper problem, "oz" and "zone" are likely the remnants of "ozone," and we can connect these two shreds even though they overlap in just one letter. In the DNA assembly problem, with only four letters in the underlying alphabet, such clues are not available. (c) Repeated regions complicate assembly, as demonstrated by the Triazzle[®]. Note that every frog in the Triazzle appears at least three times. (d) DNA sequencing machines are not perfect. Here, the red 'T' was incorrectly sequenced and should be a 'C'; this mistake of only one nucleotide may cause these two reads to be interpreted as overlapping when they are not.

alphabet actually complicates the reconstruction of the original sequence, because we will observe a greater amount of fragment overlap that is purely attributable to randomness. See Figure 3.3b.

Third, any DNA sequence contains a significant number of “conserved regions,” or information that is repeated many times with minor changes. For example, the approximately 300-nucleotide long *Alu* sequence occurs over a million times in the human genome, with only a few nucleotides changed each time due to insertions, deletions, or substitutions. Therefore, for any one particular fragment, it can become difficult to identify the specific conserved region to which it belongs within the genome. An appropriate illustration of this difficulty is the once-popular Triazzle[®] puzzle. Even though a Triazzle is a jigsaw puzzle with only 16 pieces, it contains identical figures shared by multiple pieces, making a Triazzle much more difficult than an ordinary puzzle. See Figure 3.3c.

Last but not least, modern sequencing machines are not perfect, and the reads they generate often contain errors; thus, reads which do not overlap in the genome may be incorrectly interpreted as overlapping (see Figure 3.3d).

With the pitfalls of DNA sequencing established, we next must introduce a rigorous mathematical framework in order to attack fragment assembly.



2 The mathematics of DNA sequencing

2.1 Historical motivation

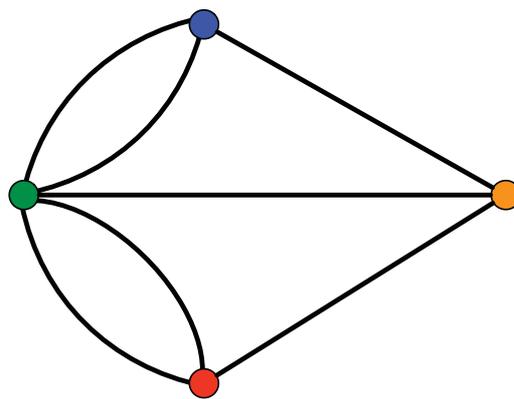
Before we jump headlong into mathematics, let us take two historical detours in order to provide our mathematical discussion with some necessary context. We begin in the eighteenth century and the Prussian city of Königsberg.¹ Königsberg was formed of opposing banks of the Pregel River, as well as two river islands; joining these four parts of the city were seven bridges (see Figure 3.4a). Now, Königsberg’s residents enjoyed taking walks, and they were curious if they could stroll through the city, cross each of the seven bridges exactly once, and return back to their starting point. Their quandary became known as the “Königsberg Bridge Problem,” and it was solved once and for all in 1735 by the great Swiss mathematician Leonhard Euler² (Figure 3.14a). Euler’s result, which we discuss below, is profound because it applies not only to the bridges of Königsberg, but in fact to *any possible* network of bridges.

¹ Present-day Kaliningrad, Russia.

² Pronounced “oiler.”



(a)



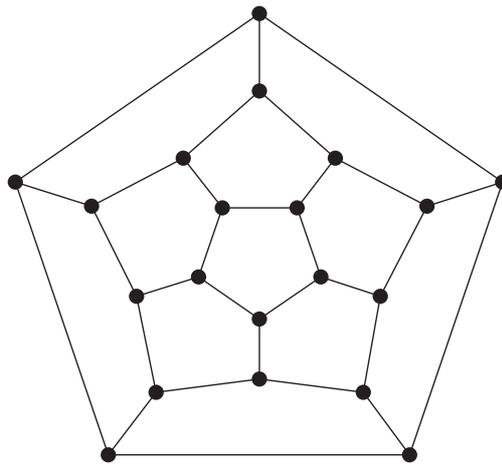
(b)

Figure 3.4 (a) Map of old Königsberg, adapted from Joachim Bering's 1613 illustration. The seven bridges have been highlighted to make them easier to see. (b) The "Königsberg Bridge Graph," formed by compressing each of four land areas to a vertex and representing each of the seven bridges as an edge.

Our second historical detour takes place in Dublin, with the creation in 1857 of the Icosian Game by the Irish mathematician William Hamilton (Figure 3.14b). This "game," which even by contemporary standards could not possibly have been very enjoyable, consisted of a wooden board with 20 pegholes and some lines connecting the holes, as well as 20 numbered pegs (see Figure 3.5a). The game's objective was to



(a)



(b)

Figure 3.5 (a) The Icosian Game, along with (b) the corresponding graph.

place the numbered pegs in the holes in such a way that Peg 1 would be connected by a line on the board to Peg 2, which would in turn be connected by a line to Peg 3, and so on, until finally Peg 20 would be connected by a line back to Peg 1. In other words, if we follow the lines on the board from peg to peg in ascending order, we reach every peg exactly once and then arrive back at our starting peg.

2.2 Graphs

With these two historical asides complete, we are ready to define a “graph” simply as a collection of “vertices” and a collection of “edges,” for which each edge pairs two vertices. The abstractness of this definition may be initially offputting, so we quickly clarify that we can always think about a graph as a network or even a map, in which the vertices are cities and the edges are roads connecting the vertices.

The benefit of providing ourselves with such a general definition is that “graph theory,” or the branch of mathematics concerned with the study of graphs, can be applied to many different types of problems. Applications of graph theory certainly include road and communications networks; however, graph theory also extends to less obvious examples, such as understanding the spread of disease or modeling the webpage connectivity of the internet.

In particular, graph theory applies to both our historical examples. In the Königsberg Bridge Problem, we obtain a graph K by assigning each of the four sectors of the city to a vertex and then connecting two given vertices (sectors) with one edge for every bridge that connects the two sectors (see Figure 3.4b). As for the Icosian Game, we obtain a graph I by representing each peghole by a vertex and then turning the lines that connect pegholes into edges that connect the corresponding vertices (see Figure 3.5b).

2.3 Eulerian and Hamiltonian cycles

Now we will generalize our two historical problems to *all* graphs. So assume that we are given any graph, which we call G , and consider an ant standing on a vertex of G . Just as the residents of Königsberg walk between the different parts of the city via bridges, the ant may walk along edges from vertex to vertex. If the ant returns to where it started, the result of its walk is a “cycle” of G . We will ask two questions about the cycles of G :

- 1 Does there exist a cycle of G in which the ant walks along each *edge* exactly once?
- 2 Does there exist a cycle of G in which the ant travels to every *vertex* exactly once?

Fittingly, Question 1 is called the *Eulerian Cycle Problem* (ECP): note that solving the ECP when our graph is K corresponds to solving the Königsberg Bridge Problem.³ We therefore define an “Eulerian cycle” in a graph G as a cycle of G which traverses every edge in G once and only once.

The second question is called the *Hamiltonian Cycle Problem* (HCP), because when the underlying graph is I , we can solve the HCP by “winning” Hamilton’s Icosian

³ We call your attention to what we mean by “solving” an ECP: because a solution corresponds to a “Yes” or “No” answer to Question 1, the ECP is considered solved when we have provided either an Eulerian cycle in the graph, or definitive proof that no such cycle exists.

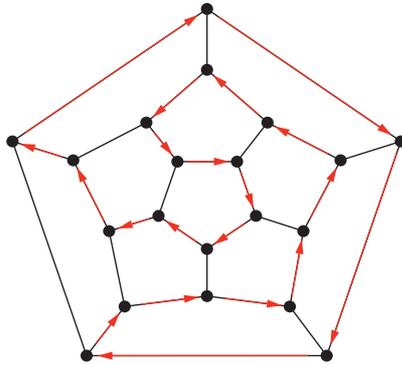


Figure 3.6 A Hamiltonian cycle in the graph I , which provides a solution to Hamilton's Icosian Game.

game (see Figure 3.6). Naturally then, a “Hamiltonian cycle” in a graph G is a cycle of G which travels to each vertex once and only once.

Finally, we define a “connected” graph as one in which an ant standing on any vertex can reach any other vertex by walking through the graph. For our purposes, it only makes sense to study the ECP and HCP for connected graphs. This is because a graph that is not connected automatically contains neither an Eulerian nor a Hamiltonian cycle, in which case the ECP and HCP are both trivial questions. Therefore, every graph in this chapter will be assumed to be connected.

2.4 Euler's Theorem

The decision to extend our historical problems to questions about graphs in general may be confusing, but this decision turns out to be key. While the ECP and HCP are superficially very similar, computer scientists have discovered that the two problems have a fundamentally different algorithmic fate: the ECP can be solved quickly even for huge graphs, while an efficient algorithm for solving the HCP for large graphs remains unknown and may not even exist.

First, we will discuss the ECP. Recall that when we introduced the Königsberg Bridge Problem, we mentioned that Euler's solution could be extended to any possible collection of bridges. What we meant by this was that Euler's solution actually provided a simple condition to solve the ECP for any graph.

Before stating Euler's result, we first need a definition. For a vertex v in a graph G , define the *degree* of v to be the number of edges connecting v to other vertices. For example, for the Königsberg graph K in Figure 3.4b, the top, bottom, and right vertices all have degree 3, while the left vertex (representing the main island of Königsberg) has degree 5. In particular, observe that since a vertex v in K represents a sector of the

city, the degree of v is equal to the number of bridges connecting that sector to other parts of the city.

Theorem (Euler's Theorem I). *An equivalent condition to a graph G having an Eulerian cycle is that the degree of every vertex of G is even.*

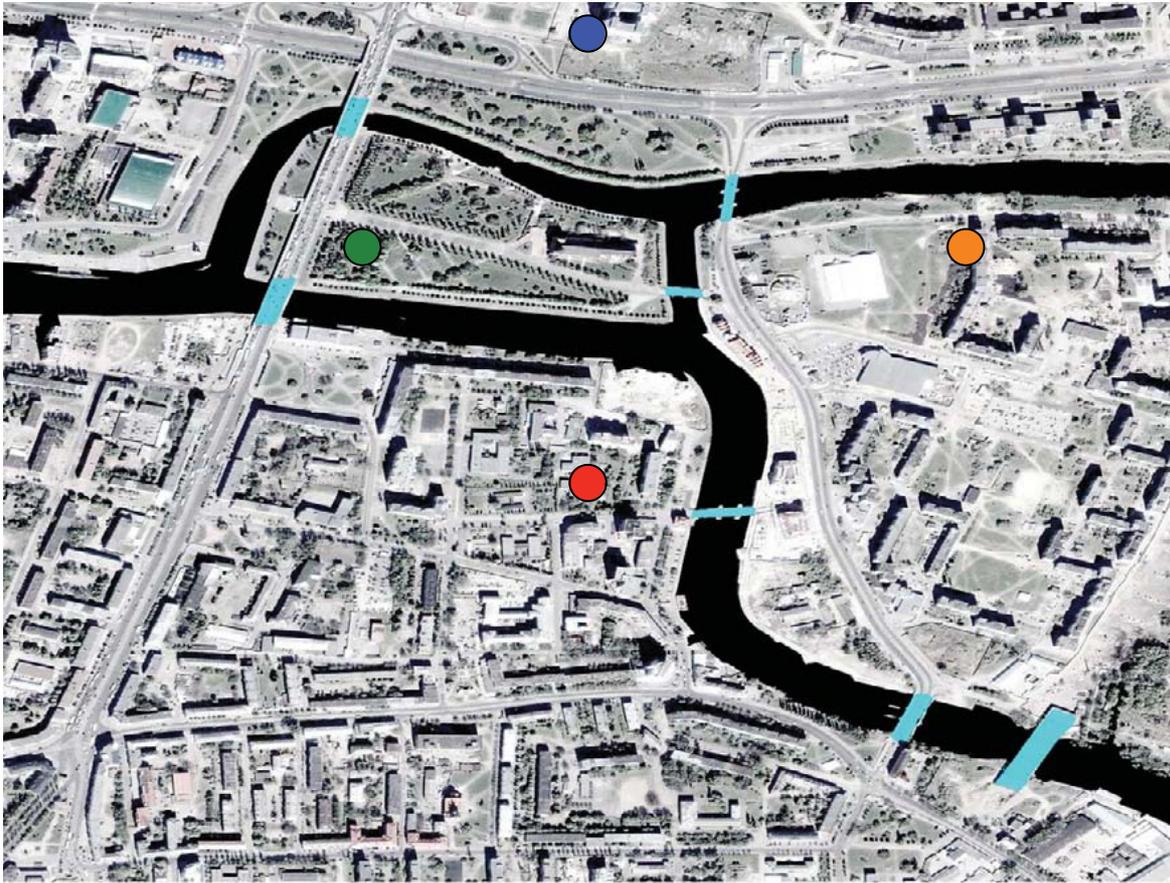
We call your attention to what two conditions being “equivalent” really means. In a sense, it means that if one is true, then the other is necessarily true as well (and vice versa). In the case of Euler's Theorem, the equivalence of the degree condition and the cycle condition is profound because it implies that for a given graph G , we can determine if G has an Eulerian cycle *without ever having to draw any cycles*. Instead, we simply need to check the degree of every vertex, a relatively simple computational task (even for a large graph).

Let us notice that Euler's Theorem immediately solves the Königsberg Bridge Problem. We have seen above that it is not the case that every vertex of K has even degree. Therefore, K does not contain an Eulerian cycle, and so we conclude that the walk for which the citizens of Königsberg had yearned does not exist.

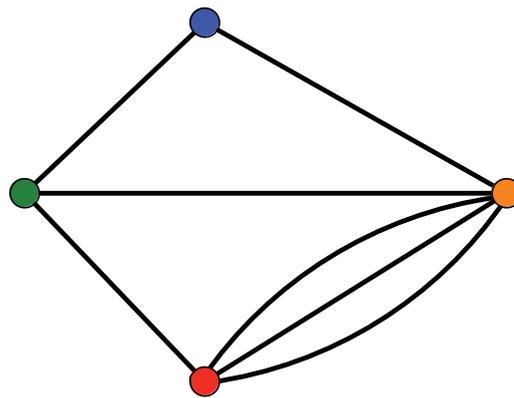
Since the eighteenth century, much has changed in the layout of Königsberg, and it just so happens that the same graph drawn today for the present-day city of Kaliningrad still does not contain an Eulerian cycle (see Figure 3.7); however, this graph does contain an *Eulerian path*, which means that a denizen of Kaliningrad can cross every bridge exactly once, but cannot do so and return to where he started. Thus, the citizens of Kaliningrad finally achieved at least a small part of the goal set by the citizens of Königsberg. Yet it is also worth noting that strolling around Kaliningrad is not as pleasant as it would have been in 1735, since the beautiful old Königsberg was ravaged by the combination of Allied bombing in 1944 and dreadful Soviet architecture in the years following World War II.

2.5 Euler's Theorem for directed graphs

We need a slightly reworked statement of Euler's Theorem in order to handle the impending application of graph theory to fragment assembly. So first assume that we instead have a “directed graph,” which is simply a graph in which all edges are provided with an orientation, so that an edge connecting v to w is not the same as an edge connecting w to v . We might like to think of a directed graph as a network in which all the edges are “one-way streets,” in which case our original undirected graph is a network in which all the edges are “two-way streets.” Accordingly, an Eulerian cycle in a directed graph G is simply an Eulerian cycle which always travels down the streets in the correct direction. A Hamiltonian cycle in G is defined analogously. See Figure 3.8.



(a)



(b)

Figure 3.7 (a) Satellite map of present-day Kaliningrad, with its bridges highlighted. (b) The graph for “Kaliningrad Bridge Problem.” Here is a challenge question: where could the city council of Kaliningrad construct new bridges so that the resulting graph will contain an Eulerian cycle?

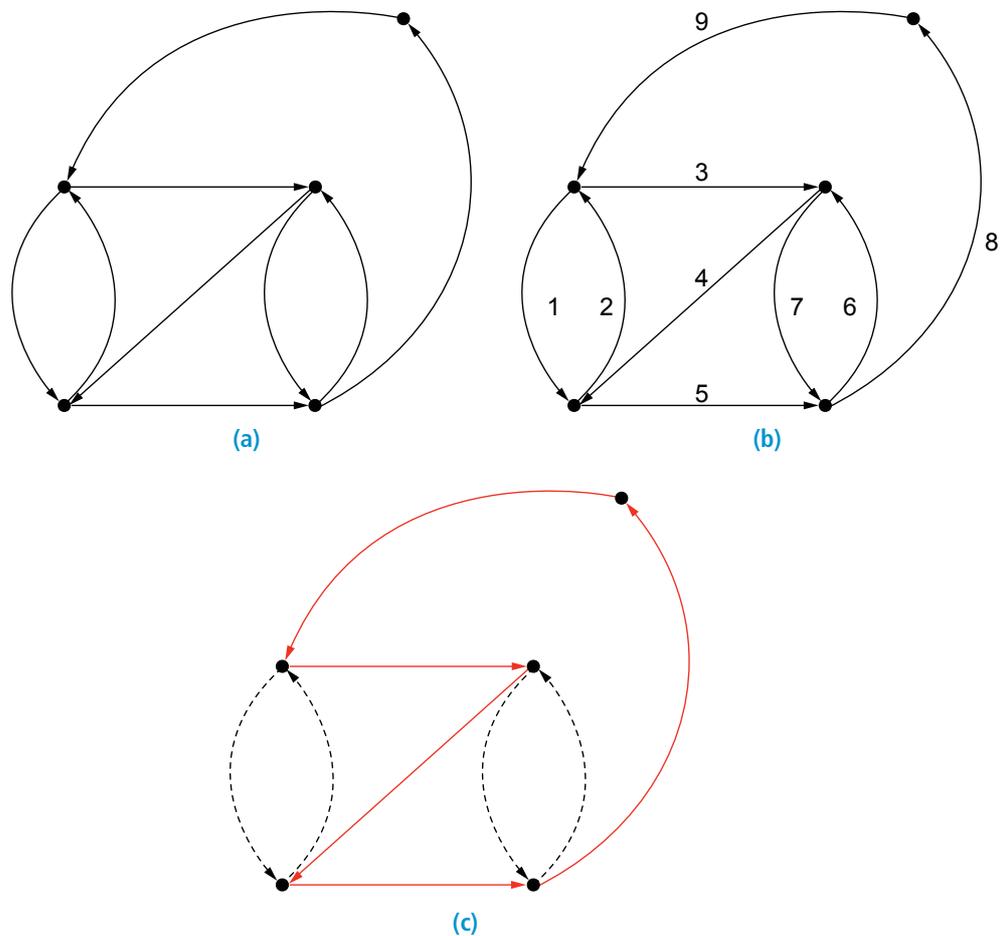


Figure 3.8 (a) A basic example of a directed graph. The arrows provide the orientations of the edges, so that we can see the directions of the “one-way streets.” (b) An illustration of an Eulerian cycle in the directed graph. The edges of the graph are numbered to indicate their order in the cycle. (c) An illustration of a Hamiltonian cycle (red edges) in the directed graph.

For any vertex v in a directed graph G , we define the “indegree” of v as the number of edges leading into v and the “outdegree” of v as the number of edges leading out from v . We are now ready to state the application of Euler’s result to directed graphs.

Theorem (Euler’s Theorem II). *An equivalent condition to a directed graph G having an Eulerian cycle is that for every vertex v in G , the indegree and outdegree of v are equal.*

A proof of Euler’s Theorem is provided at the end of the chapter, as well as a discussion of how we can find an Eulerian cycle “quickly” in the parlance of computers. The key point is that we do not have to test every possible cycle in a directed graph

G in order to determine whether G contains an Eulerian cycle. We need only find the indegree and outdegree of each vertex. If for each vertex, the indegree and outdegree match, then finding an Eulerian cycle will be easy; on the other hand, if there is *any* vertex for which the indegree and outdegree do not match, then we know that finding an Eulerian cycle is impossible.

2.6 Tractable vs. intractable problems

Inspired by Euler's Theorem, we should wonder whether there exists such a simple result governing a quick solution of the HCP. Yet although it is easy to win the Icosian Game, a solution to the HCP for an *arbitrary* graph has remained hidden.

The key challenge is that while we are guided by Euler's Theorem in solving the ECP, an analogous simple condition for the HCP remains unknown. Of course, you could always employ the method of "brute force" to solve the HCP, in which you have a computer explore all walks through the graph and report back if it finds a Hamiltonian cycle. This method is simple enough to understand, yet think about a huge graph that does not contain a Hamiltonian cycle. For this graph, the computer would have to test *every* walk through the graph before reporting back that no Hamiltonian cycle exists. The cataclysmic problem with this method is that for the average graph on just a thousand vertices, there are more walks through the graph than there are atoms in the universe!

The HCP was one of the first algorithmic problems that eluded all attempts to solve it by some of the world's most brilliant researchers. After years of fruitless effort, computer scientists began to wonder whether the HCP is *intractable*, or in other words that their failure to find a quick algorithm was not attributable to a lack of cleverness, but rather because an efficient algorithm for solving the HCP simply does not exist. Moreover, in the 1970s, computer scientists discovered thousands more algorithmic problems with the same fate as the HCP: while they are superficially simple, no one has been able to find efficient algorithms for solving them. A large subset of these problems, along with the HCP, are now collectively known as "*NP-complete*."

What has only exacerbated the frustration caused by the failure to find a simplifying condition for the HCP is that while all the NP-complete problems are different, they turn out to be *equivalent* to each other: if you find a fast algorithm for one of them, you will be able to automatically find a fast algorithm for all of them! The problem of efficiently solving NP-complete problems (or finally proving that they are intractable) is so fundamental to both computer science and mathematics that it was named on the list of "Millennium Problems" by the Clay Mathematics Institute in the year 2000: find an efficient algorithm for *any* NP-complete problem, or show that *any* NP-complete

problem is in fact intractable, and this institute will award you a prize of one million dollars.

Henceforth, we will simply think of the ECP as “easy” and the HCP as “difficult.” Keep this distinction between the two problems in mind, as it will shortly become critical.



3 From Euler and Hamilton to genome assembly

3.1 Genome assembly as a Hamiltonian cycle problem

Equipped with all the mathematics that we need, we return to fragment assembly. Having generated all our reads, we will henceforth make three simplifying assumptions about the problem at hand in order to streamline our work:

- 1 The genome we are reconstructing is cyclic.
- 2 Every read has the same length l (a string of l nucleotides is called an “ l -mer”).
- 3 All possible substrings of length l occurring in our genome have been generated as reads.
- 4 The reads have been generated without any errors.

It turns out that we can relax each of these assumptions, but the resulting solution to fragment assembly winds up being far more technical than what is suitable for this text.

In the early days of DNA sequencing, the following idea for fragment assembly was proposed. Construct a graph H by forming a vertex for every read (l -mer); we connect l -mer R_1 to l -mer R_2 by a directed edge if the string formed by the final $l - 1$ characters of R_1 (called the *suffix* of R_1) matches the string formed by the first $l - 1$ characters of R_2 (called the *prefix* of R_2). For instance, in the case $l = 5$, we would connect GGCAT to GCATC by a directed edge, but not vice versa. An example of such a graph H is provided in Figure 3.9a.

Now, consider a cycle in H . It will begin with an l -mer R_1 , and then proceed along a directed edge to a different l -mer R_2 ; let us think of walking along this edge as beginning with R_1 and tacking on the lone non-overlapping character from R_2 in order to form a “superstring” S of length $l + 1$. To continue our above example, if we walk from GGCAT to GCATC, then our superstring S will be GGCATC. Observe that the first l characters of S will be R_1 , and the final l characters of S will be R_2 . At each new vertex that we reach, we append one new character to S and notice that the final l characters of our superstring will represent the read at the present vertex. At the end of the cycle, our (cyclic) superstring S will therefore contain every l -mer that we reached

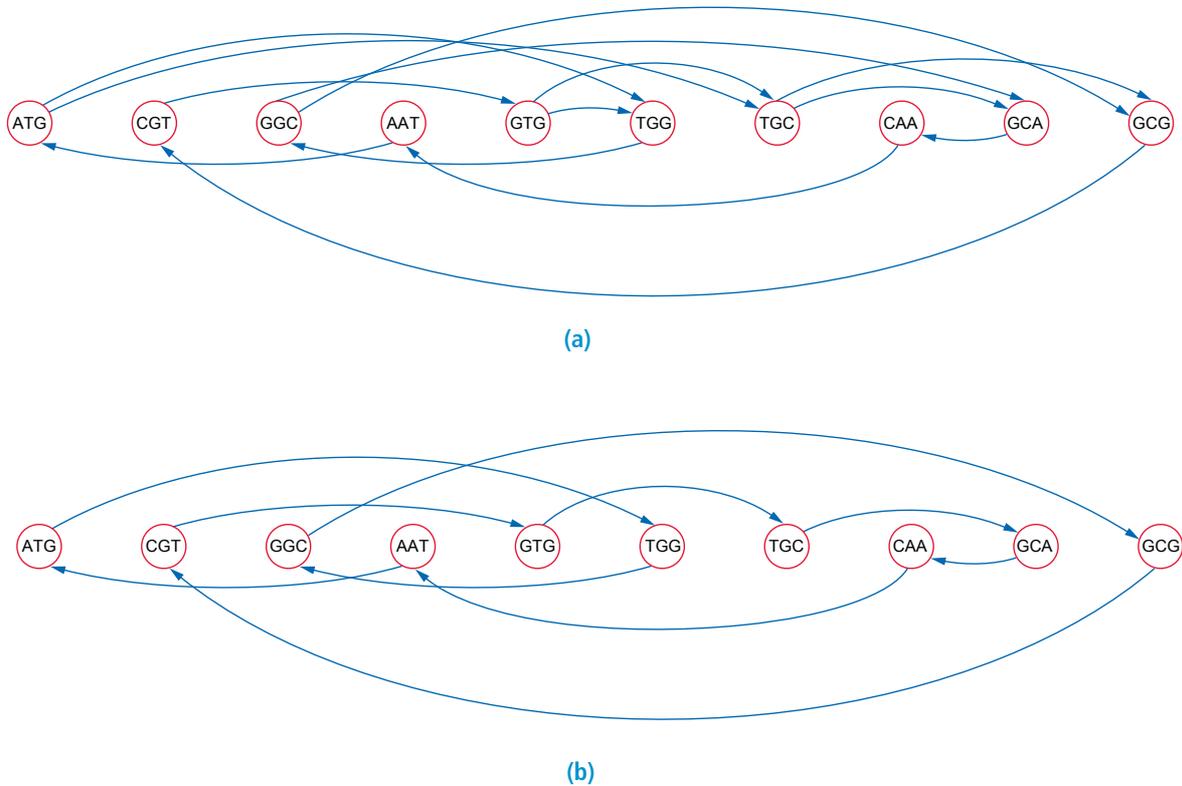


Figure 3.9 (a) The graph H for the set of 3-mers ATG, CGT, GGC, AAT, GTG, TGG, TGC, CAA, GCA, and GCG. (b) A Hamiltonian cycle in H . What is the cyclic “superstring” DNA sequence corresponding to this Hamiltonian cycle?

along the way. Extending this reasoning, a *Hamiltonian* cycle in H , which travels to every vertex in H , must correspond to a superstring of nucleotides which contains every one of our l -mers. Furthermore, every substring of length l in S will correspond to an l -mer, so S is as short as possible and therefore provides us with a candidate DNA sequence! See Figure 3.9b.

The problem with this method is that although it is elegant, it nevertheless rests upon solving the HCP, so that it is impractical unless our graph H is small. Therefore, this method is unsuitable for the graph obtained from a genome, which may have billions of vertices.

3.2 Fragment assembly as an Eulerian cycle problem

Yet all is not lost. Instead of assigning each read to a vertex, let us make the admittedly counterintuitive decision to assign each read to an *edge*. To this end, consider all prefixes and suffixes of all reads. Note that different reads may share suffixes and

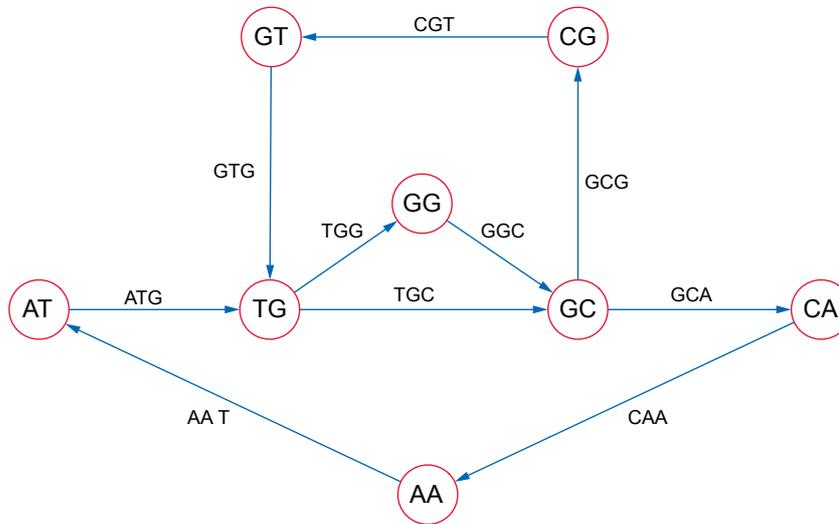


Figure 3.10 The graph E for the same set of 3-mers as in Figure 3.9. Can you find an Eulerian cycle in E ? What is the “superstring” DNA sequence corresponding to your Eulerian cycle?

prefixes; for example, reads CAGC and CAGT of length 4 share the prefix CAG. We construct a graph E with each *distinct* prefix or suffix represented by a vertex; connect an $(l - 1)$ -mer A to an $(l - 1)$ -mer B via a directed edge if there exists a read whose prefix is A and whose suffix is B . See Figure 3.10 for an example using the same set of reads from Figure 3.9.

Here, then, is the critical question: what does a cycle in E represent? Once again, imagine that you are an ant starting at some vertex of E and that you walk along a directed edge to another vertex. As with H , the result is the creation of a superstring S by tacking on the non-overlapping characters from the second vertex to those of the first. However, in this case S is just the read representing the edge connecting the two vertices. Note that in Figure 3.10, we have labeled each edge with the appropriate 3-mer.

This process repeats itself as the ant walks through E ; with each new edge, we append one additional nucleotide to the superstring S , but we also gain one additional read. Therefore, an Eulerian cycle in E will induce a (cyclic) superstring S that contains all our reads with maximum overlap, and so S is also a candidate DNA sequence. Yet in contrast to our above graph H , we have no computational troubles: by Euler’s Theorem, the ECP is easy to solve. Hence we have reduced fragment assembly to an easily solved computational problem!

Nevertheless, the reduction of fragment assembly to solving the ECP on our graph E carries one vital concern: how do we know from the start that E even *contains* an Eulerian cycle? After all, E was constructed with no thought as to whether it might

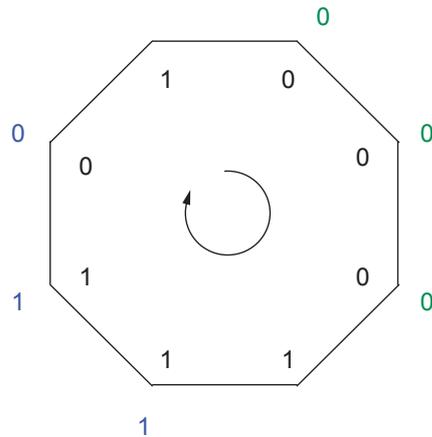


Figure 3.11 The minimal superstring problem. Here we show the circular superstring 00011101 along with illustrations of the location of the 3-digit binary numbers 000 and 110. Note that we can locate all 3-digit binary numbers in the superstring with no repeats, so 00011101 is as short as possible.

have an Eulerian cycle; if it does not, then the construction of E was simply nonsense, and the process of creating a superstring by concatenating nucleotides as we progress through E will not result in a candidate DNA sequence. In order to resolve this potential quagmire, we will tell a third and final mathematical tale.

3.3 De Bruijn graphs

In 1946, the Dutch mathematician Nicolaas de Bruijn⁴ (see Figure 3.14c) was interested in the problem of designing a circular superstring of minimal length that contains all possible l -digit binary numbers as substrings. For example, the circular string 00011101 contains all 3-digit binary numbers: 000, 001, 010, 011, 100, 101, 110, and 111. It is easy to see that 00011101 is the shortest such superstring, because it does not contain any “extra” digits, meaning that each 3-digit substring of 00011101 is the unique occurrence of one of the 3-digit binary numbers listed above. See Figure 3.11.

De Bruijn analyzed a specific class of graphs, defined as follows. Consider an alphabet of n characters, as well as some fixed number l . Form all n^{l-1} possible “words” of length $l - 1$, where a word is just a string of $l - 1$ letters from our alphabet.⁵ De Bruijn constructed a graph $B(n, l)$ (now known as the *de Bruijn graph*⁶) whose vertices

⁴ In contrast to Euler, the anglophone will find the pronunciation of “de Bruijn” very difficult: it is similar to “brine,” except with a slight ‘r’ sound between the ‘i’ and the ‘n.’

⁵ There are n^{l-1} such words because there are n choices for the first letter, n choices for the second letter, and so on. Since there are $l - 1$ letters to choose, we wind up with n^{l-1} total possibilities.

⁶ This nomenclature is a bit cruel to the British mathematician I. J. Good, who independently discovered de Bruijn graphs.

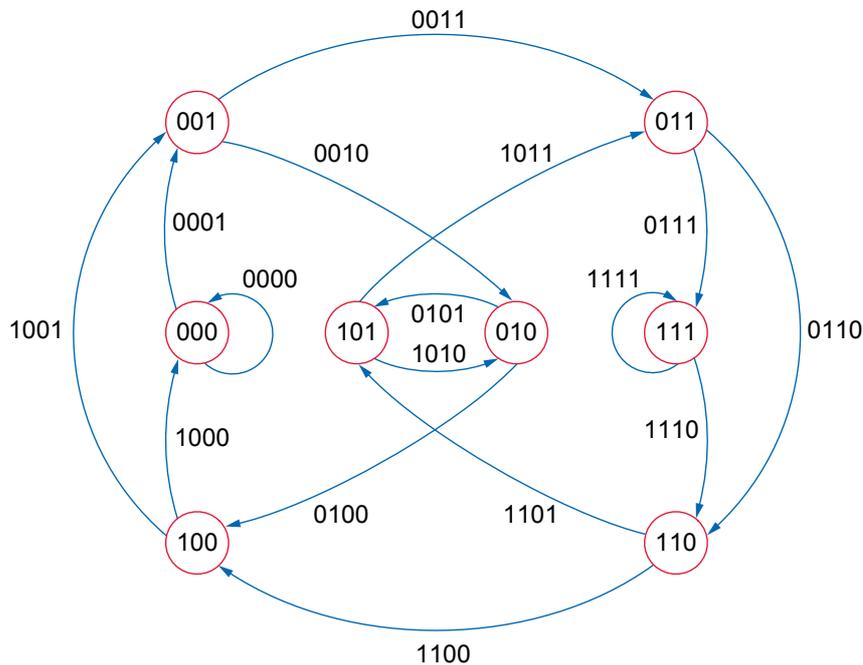


Figure 3.12 The de Bruijn graph $B(2, 4)$, where our 2-character “alphabet” is composed of just the digits 0 and 1. Observe that by Euler’s Theorem, this graph must have an Eulerian cycle; we will find such a cycle for this graph in Figure 3.19.

are all n^{l-1} words of length $l - 1$; a directed edge connects word w_1 to word w_2 if there exists an l -letter word W whose prefix is w_1 and whose suffix is w_2 . See Figure 3.12.

The crucial property shared by all de Bruijn graphs is that every one of them will always contain an Eulerian cycle. For example, in Figure 3.12 we can see that there are two edges entering every vertex and two edges leaving every vertex of $B(2, 4)$, implying that it has an Eulerian cycle. To see why the same is true for *any* de Bruijn graph $B(n, l)$, consider a vertex w corresponding to a word of length $l - 1$. There exist n words of length l whose prefix is w (each such word is obtained by adding one of n letters to the end of w) and thus the outdegree of each vertex in $B(n, l)$ is n . Similarly, there exist n words of length l whose suffix is w (each such word is obtained by adding one of n letters to the beginning of w) and thus the indegree of each vertex in $B(n, l)$ is also n . Hence every vertex of $B(n, l)$ has indegree and outdegree both equal to n , and so Euler’s Theorem implies that $B(n, l)$ must have an Eulerian cycle.

The biological connection arises when we realize that our graph E above will be contained in the de Bruijn graph $B(4, l)$, because whereas the vertices of E are all $(l - 1)$ -mers occurring as prefixes or suffixes of our reads, the vertices of $B(4, l)$ are

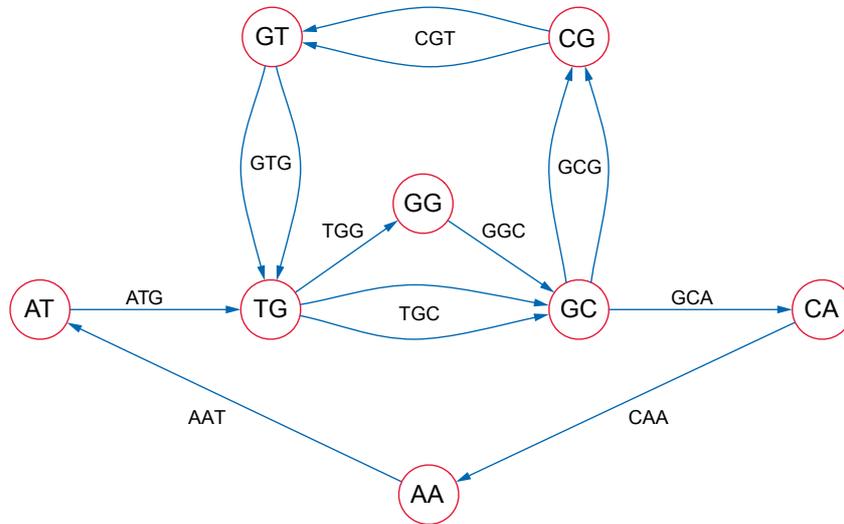


Figure 3.13 This more general version of the graph from Figure 3.10 allows for the case that the same read occurs in more than one location in the genome. The good news is that this generalization does not make the problem any more difficult to solve: an Eulerian cycle in this graph will still correspond to a candidate DNA sequence.

all possible $(l - 1)$ -mers. Furthermore, it can be demonstrated that E itself has an Eulerian cycle!

3.4 Read multiplicities and further complications

Imagine for a moment that our genome is ATGCATGC. Then we will obtain four reads of length 3: ATG, TGC, GCA, and CAT; however, this might lead us to reconstruct the genome as ATGC. The problem is that each of these reads actually occurs *twice* in the original genome. Therefore, we will need to adjust genome reconstruction so that we not only find all l -mers occurring as reads, but we also find how many times each such l -mer occurs in the genome, called its “ l -mer multiplicity.” The good news is that we can still handle fragment assembly in the case l -mer multiplicities are known.

We simply use the same graph E , except that if the multiplicity of an l -mer is k , we will connect its prefix to its suffix via k edges (instead of just one). Continuing our ongoing example from Figure 3.10, if during read generation we discover that each of the four 3-mers TGC, GCG, CGT, and GTG has multiplicity 2, and that each of the six 3-mers ATG, TGG, GGC, GCA, CAA, and AAT has multiplicity 1, we create the graph shown in Figure 3.13. In general, it is easy to see that the graph resulting from adding multiplicity edges is Eulerian, as both the indegree and outdegree of a vertex

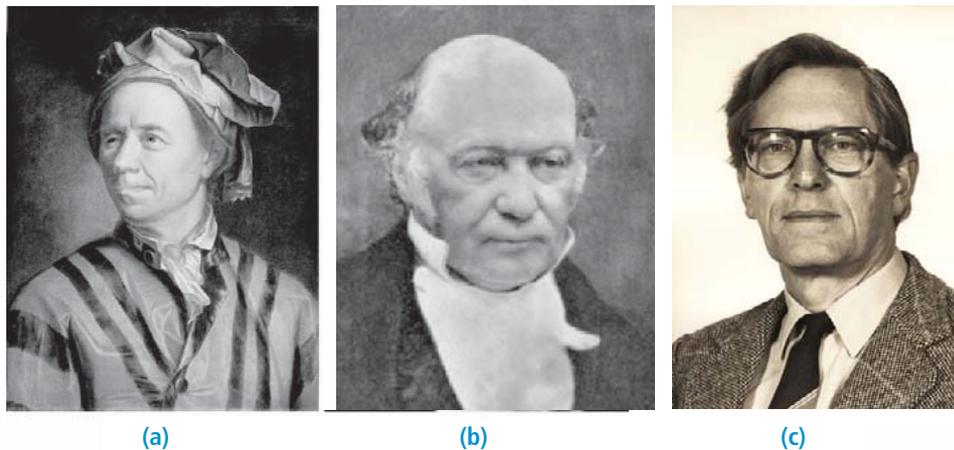


Figure 3.14 The three mathematicians. (a) Leonhard Euler. (b) William Hamilton. (c) Nicolaas de Bruijn.

(represented by an $(l - 1)$ -mer) equals the number of times this $(l - 1)$ -mer appears in the genome.

In practice, information about the exact multiplicities of $(l - 1)$ -mers in the genome may be difficult to obtain, even with modern sequencing technologies. However, computer scientists have recently found a way to reconstruct the genome even when this information is unavailable. Furthermore, DNA sequencing machines are prone to errors, our reads will have varying lengths, and so on. However, with every variation to fragment assembly, it has proven fruitful to apply some cousin of de Bruijn graphs in order to transform a question involving Hamiltonian cycles into a different question about Eulerian cycles.



4

A short history of read generation

4.1 The tale of three biologists: DNA chips

While Euler, Hamilton, and de Bruijn could not possibly meet each other, their mathematical fates got intricately criss-crossed. In 1988, three other Europeans would find their fates intertwined (Figure 3.15). Radoje Drmanac (Serbia), Andrey Mirzabekov (Russia), and Edwin Southern (UK) simultaneously and independently developed the futuristic and at the time completely implausible method of *DNA chips* as a proposal for read generation. None of these three biologists knew of the work of Euler, Hamilton, and de Bruijn; none could have possibly imagined that the implications of his own



Figure 3.15 The three biologists. (a) Radoje Drmanac. (b) Andrey Mirzabekov. (c) Edwin Southern.

experimental research would eventually bring him face to face with these giants of mathematics.

In 1977 Fred Sanger and colleagues sequenced the first virus, the tiny 5,375 nucleotide long bacteriophage ϕ X174. However, while biologists in the late 1980s were routinely sequencing viruses containing hundreds of thousands of nucleotides, the idea of sequencing bacterial (let alone human) genomes seemed preposterous, both experimentally and computationally. Drmanac, Mirzabekov, and Southern realized that one main problem with the original DNA sequencing technology developed in the 1970s is the fact that it is not cost-effective for larger genomes. Indeed, generating a single read in the late 1980s cost more than a dollar, and thus sequencing a mammalian genome would have been a billion-dollar enterprise.⁷ Due to such a high cost, it was infeasible to generate all l -mers from a genome, one of our conditions for the successful application of the Eulerian approach. DNA chips were therefore invented with the goal of cheaply generating *all* l -mers from a genome, albeit with a smaller read length l than the original DNA sequencing technology. For example, whereas traditional sequencing techniques generated reads containing approximately 500 nucleotides, the inventors of DNA arrays aimed at producing reads with around 15 nucleotides.

DNA chips work as follows. One first synthesizes all 4^l possible l -mers (i.e. all DNA fragments of length l) and attaches them to a *DNA array*, which is a grid on which each l -mer is assigned a unique location. We next take an (unknown) DNA fragment,

⁷ Even in 2000, when the cost of read generation reduced substantially, sequencing the human genome still cost a few hundred million dollars.

AAA	AGA	CAA	CGA	GAA	GGA	TAA	TGA
AAC	AGC	CAC	CGC	GAC	GGC	TAC	TGC
AAG	AGG	CAG	CGG	GAG	GGG	TAG	TGG
AAT	AGT	CAT	CGT	GAT	GGT	TAT	TGT
ACA	ATA	CCA	CTA	GCA	GTA	TCA	TTA
ACC	ATC	CCC	CTC	GCC	GTC	TCC	TTC
ACG	ATG	CCG	CTG	GCG	GTG	TCG	TTG
ACT	ATT	CCT	CTT	GCT	GTT	TCT	TTT

Figure 3.16 A schematic of the DNA array containing all possible 3-mers. Ten fluorescently labeled 3-mers represent complements of the 10 3-mers from Figures 3.9 and 3.10. In order to obtain our reads from this array, we simply take the complements of the highlighted 3-mers. For example, CAC is highlighted, which means that GTG (the complement of CAC) is one of our reads. Note that this DNA array provides no information regarding *l*-mer multiplicities.

fluorescently label it, and apply a solution containing this fluorescently labeled DNA to the DNA array. The upshot is that the nucleotides in the DNA fragment will hybridize (bond) to their complements on the array (A will bond to T, and C to G). All we need to do is use spectroscopy to analyze which sites on the array emit the greatest fluorescence; the complement of the *l*-mer corresponding to such a site on the array must therefore be one of our reads. See Figure 3.16 for an illustration of the DNA array for our recurring set of reads.

At first, almost no one believed that the idea of DNA arrays would work, because both the biochemical problem of synthesizing millions of short DNA fragments and the mathematical problem of sequence reconstruction appeared too complicated. In 1988, *Science* magazine wrote that given the amount of work required to synthesize a DNA array, “using DNA arrays for sequencing would simply be substituting one horrendous task for another.” It turned out that *Science* was wrong: in the mid 1990s, a number of startup companies perfected technologies for designing large DNA arrays. However,

DNA arrays ultimately failed to realize the dream that motivated their inventors. Arrays are incapable of sequencing DNA, because the fidelity of DNA hybridization with the array is too low and because the value of l is too small.

Yet the failure of DNA arrays was a spectacular one: while the original goal (DNA sequencing) was out of reach for the moment, two new unexpected applications of DNA arrays emerged. Today, arrays are used to measure gene expression, as well as to analyze genetic variations. These new applications transformed DNA arrays into a multi-billion dollar industry that included Hyseq (founded by Radoje Drmanac) and Oxford Gene Technology (founded by Sir Edwin Southern).

4.2 Recent revolution in DNA sequencing

After founding Hyseq, Radoje Drmanac did not abandon his dream of inventing an alternative DNA sequencing technology. In 2005 he founded *Complete Genomics*, which recently developed the technology to generate (nearly) all l -mers from a genome, thus at last enabling the method of Eulerian assembly. While his *nanoball arrays* technology is quite different from the DNA chip technology he proposed in 1988, one can still recognize the intellectual legacy of DNA chips in nanoball arrays, a testament that good ideas do not die even if they fail. Moreover, a number of other companies, including *Illumina* and *Life Technologies*, are competing with Complete Genomics by using their own technologies to generate (nearly) all l -mers from a genome. While DNA arrays failed to generate accurate reads even 15 nucleotides long, the next generation sequencing technologies generate reads of length 25 nucleotides and longer (and producing hundreds of millions such reads in a single experiment). These developments in *next-generation sequencing technologies* in the last five years have revolutionized genomics, and biologists are presently preparing to assemble the genomes of *all* the mammals on Earth (Figure 3.17) ... while still relying on the grand idea that Leonhard Euler developed in 1735.



5

Proof of Euler's Theorem

We now will prove Euler's Theorem. First, let us restate his result for the case of undirected graphs, which we may recall are graphs for which the edges are "two-way streets."

Theorem (Euler's Theorem I). *An equivalent condition to a graph G having an Eulerian cycle is that the degree of every vertex of G is even.*

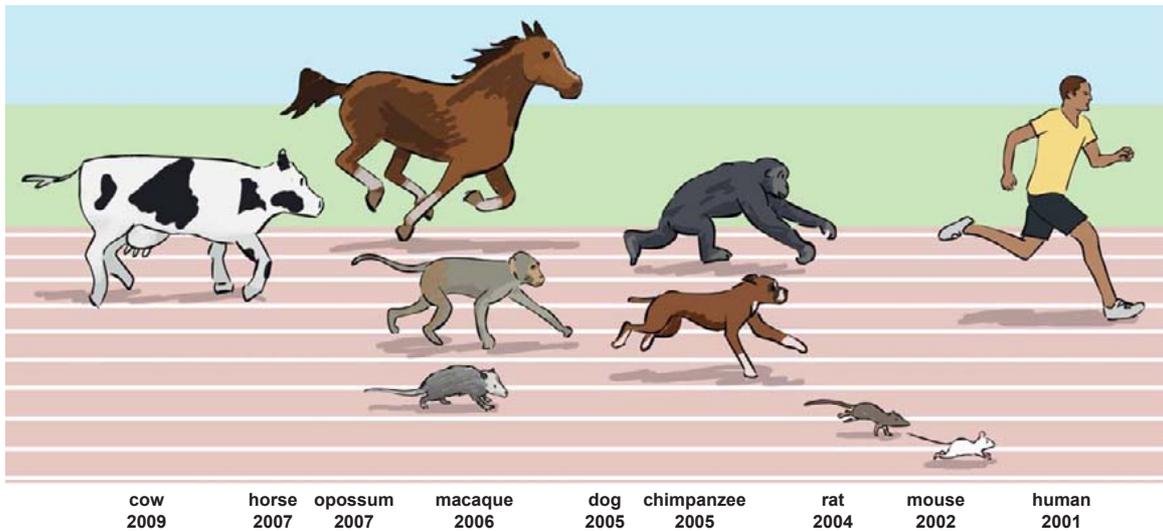


Figure 3.17 At the moment, only nine mammals have had their genomes sequenced: human, mouse, rat, dog, chimpanzee, macaque, opossum, horse, and cow. This is all about change.

We shall only prove the second version of Euler’s Theorem for directed graphs (in which the edges are “one-way streets”), which is ultimately more relevant to the themes of this chapter. We urge you to read through the proof we provide carefully, and then see if you can prove Euler’s Theorem I for yourself. Do not be terrified. The overall structure of the two proofs is identical, except for a few details. Simply follow the proof of Euler’s Theorem II and fit in the appropriate details for undirected graphs.

Here, then, is the restatement of Euler’s Theorem for directed graphs.

Theorem (Euler’s Theorem II). *An equivalent condition to a directed graph G having an Eulerian cycle is that for every vertex v in G , the indegree and outdegree of v are equal.*

Recall that two conditions being “equivalent” means that if one is true, then the other must be true. In this specific instance, our equivalent conditions are as follows for a given directed graph G :

- 1 G has an Eulerian cycle.
- 2 Each vertex of G has equal indegree and outdegree.

So in order to prove that these two conditions are equivalent, we simply need to demonstrate two statements. First, we need to show that if (1) is true for a directed graph G , then so is (2). Second, we must show that if (2) is true for a directed graph G , then so is (1). If these two statements hold, then there is no way that we can have a

directed graph for which condition (1) is true and condition (2) is false, or vice versa. In other words, our two conditions above will be equivalent.

Proof First we will show that if condition (1) is true, then so is condition (2). So assume that we are given a directed graph G which contains an Eulerian cycle; our aim is to show that each vertex of G has equal indegree and outdegree. Every time we enter a vertex in the Eulerian cycle of G , we leave it via a different edge. If a vertex v is used k times throughout the course of the cycle, then we enter v via a total of k edges and leave v via a total of k edges. All $2k$ edges are distinct, because since our cycle is Eulerian, no edge can be used more than once. Furthermore, these $2k$ edges constitute all edges touching this vertex, since an Eulerian cycle uses every edge in G . Therefore the indegree and outdegree of v are both equal to k . We can iterate this argument on every vertex in G to obtain that every vertex in G has equal indegree and outdegree, as needed.

Conversely, we need to show that if condition (2) is true, then so is condition (1). So assume that we are given a directed graph G for which each vertex has indegree equal to its outdegree. We will actually *form* an Eulerian cycle in G by the following procedure. Choose any vertex v in G , and choose any edge leaving v . Travel down this edge to the next vertex. Continue this process of choosing any unused edge to walk down, creating what is called a “random walk,” while making sure only that we never use the same edge twice. Eventually, we will reach our original vertex v , creating a cycle which we call C_1 . We should be suspicious of why a random walk in G is guaranteed to produce a cycle; this fact is ensured by the assumed condition that every vertex has equal indegree and outdegree, so that every time we arrive at a vertex, we must be able to find an unused edge leaving it (i.e. we cannot get “stuck” along our walk).

Now, once we have formed our cycle C_1 , there are two possibilities for it. Either C_1 is an Eulerian cycle, in which case we are finished, or C_1 is not Eulerian. In the latter case, remove C_1 from G to form a new graph H . Because every vertex of C_1 (a cycle) must have indegree equal to its outdegree, condition (2) must also hold for every vertex in H . Since G is connected, we are guaranteed to have some vertex w in H that contains edges in both H and C_1 . So since condition (2) holds for H , we can start at w and form an arbitrary cycle C_2 in H via a random walk in H .

We now have two cycles, C_1 and C_2 , which do not share any edges but which both pass through w . We can therefore consolidate C_1 and C_2 to form a single “supercycle,” which we call C . See Figure 3.18 for a brief illustration of how we form C .

In turn, we test if C is Eulerian, and if not we can iterate the above procedure indefinitely. If at any step our supercycle C becomes an Eulerian cycle, then we are

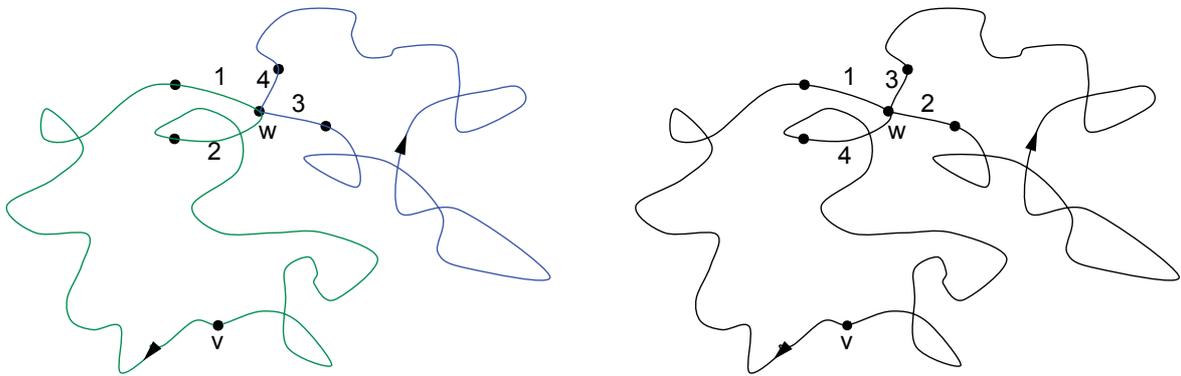


Figure 3.18 Cycle consolidation. If we have two cycles passing through the same vertex w , then we can combine them into a single cycle simply by changing the order in which we choose edges leaving w .

finished. The only concern is that C might never become Eulerian. However, this is impossible: there are only finitely many edges in the original graph G , so that since we remove some edges at each step, eventually we must reach a step at which we run out of edges. When we consolidate cycles at this step, our supercycle will use every edge in G without using any edges more than once, which is precisely the definition of an Eulerian cycle in G . Therefore G has an Eulerian cycle, which is what we set out to show.

The brilliant facet of this proof (as well as the proof of Euler’s Theorem I) is that it serves as an example of what mathematicians call a “constructive proof,” or a proof that not only proves the desired result, but also delivers us with a very precise method for actually *constructing* what we need, which in this case is an Eulerian cycle. Therefore, if we are given a graph and asked to find an Eulerian cycle in it, we can easily test to see if each vertex has indegree equal to its outdegree (or if the degree of each vertex is even, as in the case of undirected graphs). If this condition fails, then the graph contains no Eulerian cycle; if it holds, we simply follow the idea outlined in the proof and form an arbitrary sequence of cycles that do not share any edges, combining the cycles into a single “supercycle” at each step, and iterating this process until an Eulerian cycle is inevitably obtained.

Let us conclude by illustrating the power of our constructive proof. In Figure 3.19, we apply Euler’s Theorem to find an Eulerian cycle in the de Bruijn graph from Figure 3.12. Keep in mind that the same method will work for genome graphs containing billions of edges. At last, we have definitively solved our giant puzzle!

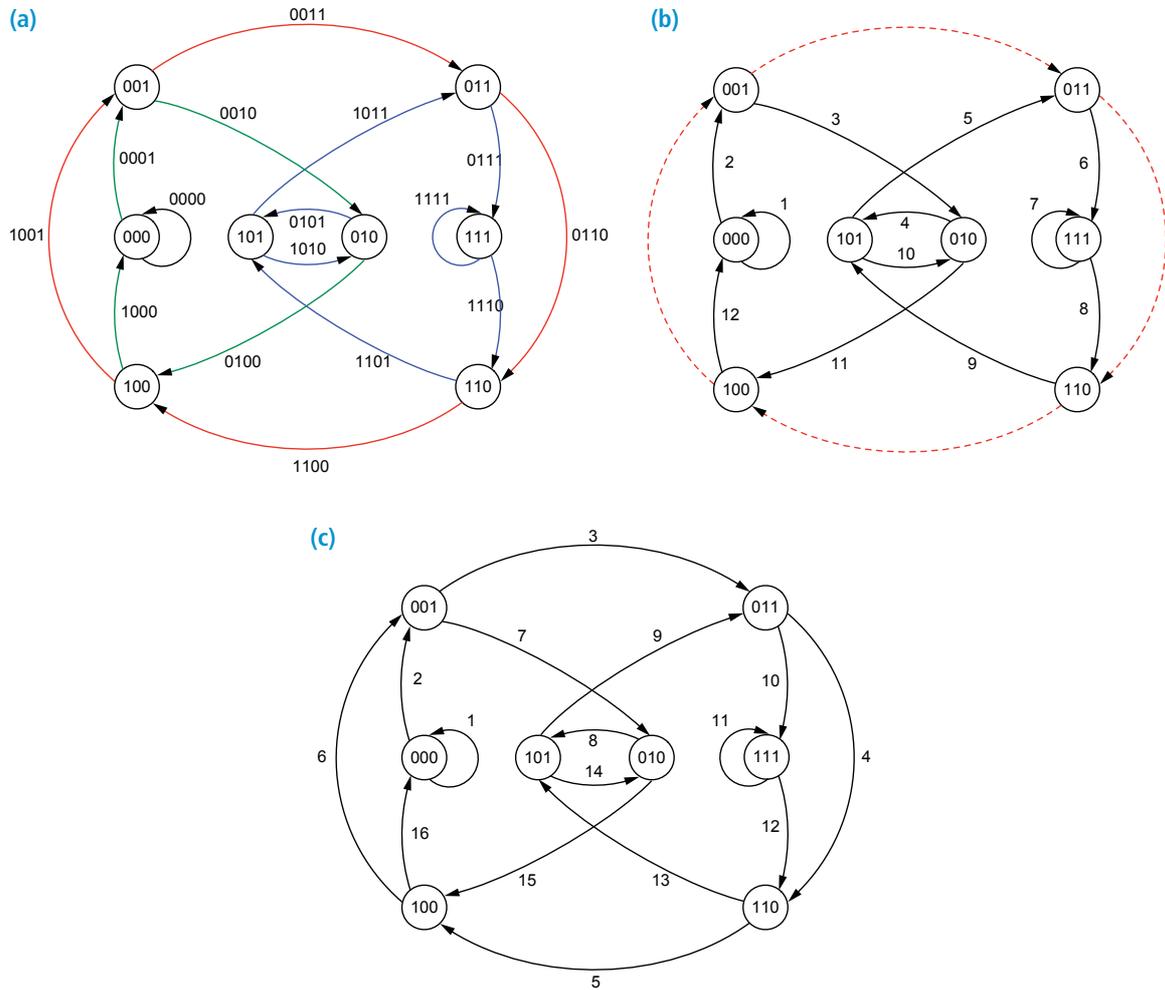


Figure 3.19 Obtaining an Eulerian cycle from a graph in which all vertices have the appropriate degrees. Here, we find an Eulerian cycle in the directed graph $B(2, 3)$ from Figure 3.12. (a) We first find three arbitrary cycles in the graph at hand (here shaded with three different colors). Once we have chosen the green cycle, we remove it from the graph and choose the blue cycle, which we then remove from the graph and choose the red cycle. (b) We next consolidate the green and blue cycles into a single cycle (black). The edge numberings give the order of the edges if we start at vertex 000. Note that the red cycle is dashed to indicate that it is not yet part of our supercycle. (c) Finally, we add the red cycle into our supercycle, which is Eulerian. The edges are renumbered as needed. The resulting Eulerian cycle spells the cyclic superstring 0000110010111101.



DISCUSSION

We have met three mathematicians of three different centuries, Euler, Hamilton, and de Bruijn, spread out across the European continent, each with his own queries. We might be inclined to feel a sense of adventure at their work and how it converged to this singular point in modern biology. Yet the first biologists who worked on DNA sequencing had no idea of how graph theory could be applied to this subject; what's more, the first paper combining the trio's mathematical ideas into fragment assembly was published lifetimes after the deaths of Euler and Hamilton, when de Bruijn was in his seventies. So perhaps we might think of these three men not as adventurers, but instead as lonely wanderers. As is so often the mathematician's curse, each man passionately pursued questions in the abstract mathematical world while having no idea where the answers might one day lead without him in the real world.



NOTES

Euler's solution of the Königsberg Bridge Problem was presented to the Imperial Russian Academy of Sciences in St. Petersburg on August 26, 1735. Euler was the most prolific writer of mathematics of all time: besides graph theory, he first introduced the notation $f(x)$ to represent a function, i for the square root of -1 , and π for the circular constant. Working very hard throughout his entire life, he became blind. In 1735, he lost the use of his right eye. He kept working. In 1766, he lost the use of his left eye and commented: "Now I will have fewer distractions." He kept working. Even after becoming completely blind, he published *hundreds* of papers.

After Euler's work on the Königsberg Bridge Problem, graph theory was forgotten for over a hundred years, but was revived in the second half of the nineteenth century by prominent mathematicians, among them William Hamilton. Graph theory flourished in the twentieth century, when it became an area of mainstream mathematical research.

DNA sequencing methods were invented independently and simultaneously in 1977 by Frederick Sanger and colleagues [1] as well as Walter Gilbert and colleagues [2]. The Hamiltonian cycle approach to DNA sequencing was first outlined in 1984 [3] and further developed by John Kececioglu and Eugene Myers in 1995 [4]. Advances in DNA sequencing led to the sequencing of the entire 1800 kb *H. influenzae* bacterial genome in the mid 1990s. The human genome was sequenced using the Hamiltonian approach in 2001.

DNA arrays were proposed simultaneously and independently in 1988 by Radoje Drmanac and colleagues in Yugoslavia [5], Andrey Mirzabekov and colleagues in Russia [6], and Ed Southern in the UK [7]. The Eulerian approach to DNA arrays was described in [8]. The Eulerian approach to DNA sequencing was described in [9] and further developed in 2001 [10], when hardly anybody believed it could be made practical.

At roughly the same time, Sydney Brenner and colleagues introduced the *Massively Parallel Signature Sequencing* (MPSS) method [11], which brought in the era of next generation sequencing with short reads. Throughout the last decade, MPSS in addition to technologies developed by such companies as Complete Genomics, Illumina, and Life Technologies revolutionized genomics. Next-generation techniques produce rather short reads, which vary in length from 30 to 100 nucleotides and result in a challenging fragment assembly problem. To address this challenge, a number of assembly tools have been developed [12–15], all of which follow the Eulerian approach.



QUESTIONS

- (1) Does the graph I representing the Icosian Game contain an Eulerian cycle? Why or why not?
- (2) Construct the de Bruijn Graph $B(3, 3)$ and find an Eulerian cycle in it.
- (3) Give three Eulerian cycles in the graph of Figure 3.13 along with their corresponding cyclic superstrings.
- (4) From the following set of reads of length 4, use the ideas of this chapter to provide a (cyclic) candidate DNA sequence: AACG, TCGT, GATC (multiplicity 2), TATC, ATCG, CCCG, ATCC (multiplicity 2), CGGA, CCCT, GTAT, CCGA, CTAA, TCCC (multiplicity 2), GGAT, CCTA, TAAC, CGAT, CGTA, ACGG.
- (5) Prove Euler's Theorem I.



REFERENCES

- [1] F. Sanger, S. Nicklen, and A. R. Coulson. DNA sequencing with chain-terminating inhibitors. *Proc. Natl Acad. Sci. U S A*, 74:5463–5467, 1977.
- [2] A. M. Maxam and W. Gilbert. A new method for sequencing DNA. *Proc. Natl Acad. Sci. U S A*, 74:560–564, 1977.

- [3] H. Peltola, H. Soderlund, and E. Ukkonen. SEQAID: A DNA sequence assembling program based on a mathematical model. *Nucl. Acids Res.*, 12:307–321, 1984.
- [4] J. Kececiloglu and E. W. Myers. Combinatorial algorithms for DNA sequence assembly. *Algorithmica*, 13:7–51, 1995.
- [5] R. Drmanac, I. Labat, I. Brukner, and R. Crkvenjakov. Sequencing of megabase plus DNA by hybridization: Theory of the method. *Genomics*, 4:114–128, 1989.
- [6] Y. Lysov, V. Florent'ev, A. Khorlin, K. Khrapko, V. Shik, and A. Mirzabekov. DNA sequencing by hybridization with oligonucleotides. *Dok. Acad. Nauk USSR*, 303:1508–1511, 1988.
- [7] E. Southern. United Kingdom patent application gb8810400. 1988.
- [8] P. A. Pevzner. *l*-tuple DNA sequencing: Computer analysis. *J. Biomol. Struct. Dyn.*, 7:63–73, 1989.
- [9] R. Idury and M. Waterman. A new algorithm for DNA sequence assembly. *J. Comput. Biol.*, 2:291–306, 1995.
- [10] P. A. Pevzner, H. Tang, and M. Waterman. An Eulerian path approach to DNA fragment assembly. *Proc. Natl Acad. Sci. U S A*, 98:9748–9753, 2001.
- [11] S. Brenner, M. Jonson, J. Bridgham, *et al.* Gene expression analysis by massively parallel signature sequencing (MPSS) on microbead arrays. *Nat. Biotech.*, 18:630–634, 2000.
- [12] M. J. Chaisson and P. A. Pevzner. Short read fragment assembly of bacterial genomes. *Genome Res.*, 18:324–330, 2008.
- [13] D. R. Zerbino and E. Birney. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res.*, 18:821–829, 2008.
- [14] J. Butler, I. MacCullum, M. Kieber, *et al.* ALLPATHS: De novo assembly of whole-genome shotgun microreads. *Genome Res.*, 18:810–820, 2008.
- [15] J. T. Simpson, K. Wang, S. D. Jackman, *et al.* ABySS: A parallel assembler for short read sequence data. *Genome Res.*, 19:1117–1123, 2009.