How to apply de Bruijn graphs to genome assembly

Phillip E C Compeau, Pavel A Pevzner & Glenn Tesler

A mathematical concept known as a de Bruijn graph turns the formidable challenge of assembling a contiguous genome from billions of short sequencing reads into a tractable computational problem.

The development of algorithmic ideas for next-generation sequencing can be traced back 300 years to the Prussian city of Königsberg (present-day Kaliningrad, Russia), where seven bridges joined the four parts of the city located on opposing banks of the Pregel River and two river islands (**Fig. 1a**). At the time, Königsberg's residents enjoyed strolling through their city, and they wondered if every part of the city could be visited by walking across each of the seven bridges exactly once and returning to one's starting location. The solution came in 1735, when the great mathematician Leonhard Euler[1] made a conceptual breakthrough that would solve this 'Bridges of Königsberg problem'. Euler's first insight was to represent each landmass as a point (called a node) and each bridge as a line segment (called an edge) connecting the appropriate two points. This creates a graph—a network of nodes connected by edges (**Fig. 1b**). By describing a procedure for determining whether an arbitrary graph contains a path that visits every edge exactly once and returns to where it started, Euler not only resolved the Bridges of Königsberg problem but also effectively launched the entire branch of mathematics known today as graph theory[2].

Since Euler's original description, the use of graph theory has turned out to have many additional practical applications, most of which have greater scientific importance than the development of walking itineraries. Specifically, Euler's ideas were subsequently adapted by Dutch mathematician Nicolaas de Bruijn to find a cyclic sequence of letters taken from a given alphabet for which every possible word of a certain length (k) appears as a string of consecutive characters in the cyclic sequence exactly once (**Box 1** and **Fig. 2**). Application of the de Bruijn graph has also proven invaluable in the field of molecular biology where researchers are faced with the problem of assembling billions of short sequencing reads into a single genome. In the following article, we describe the problems faced when constructing a genome and how the de Bruijn graph approach can be applied to assemble short-read sequences.

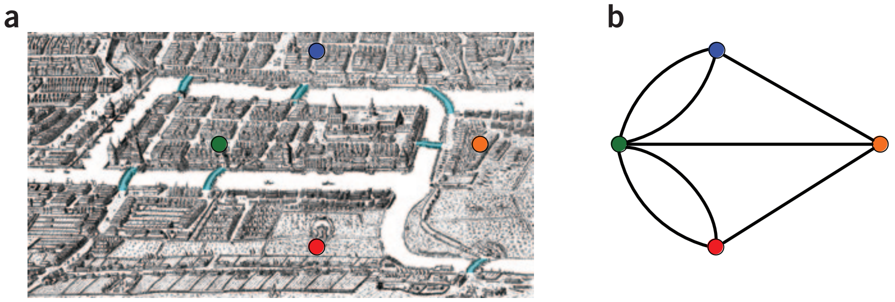

Figure 1 Bridges of Königsberg problem. (**a**) A map of old Königsberg, in which each area of the city is labeled with a different color point. (**b**) The Königsberg Bridge graph, formed by representing each of four land areas as a node and each of the city's seven bridges as an edge.

Phillip E. C. Compeau and Glenn Tesler are in the Department of Mathematics, University of California San Diego, La Jolla, California, USA, and Pavel A. Pevzner is in the Department of Computer Science and Engineering, University of California San Diego, La Jolla, California, USA.
e-mail: ppevzner@ucsd.edu

Problems with alignment-based assembly

To illustrate why graphs are useful for genome assembly, consider a simple example with five very short reads (CGTGCAA, ATGGCGT, CAATGGC, GGCGTGC and TGCAATG) sequenced from a small circular genome, ATGGCGTGCA (**Fig. 3a**). Current next-generation sequencing methods produce reads that vary in length, but the most popular technology generates ~100-nucleotide reads. A straightforward method for assembling reads into longer contiguous sequences—and the one used for assembling the human genome[3,4] in 2001 as well as for all other projects based on Sanger sequencing—uses a graph in which each read is represented by a node and overlap between reads is represented by an arrow (called a 'directed edge') joining two reads. For instance, two nodes representing reads may be connected with a directed edge if the reads overlap by at least five nucleotides (**Fig. 3b**).

Visualizing an ant walking along the edges of this graph provides an aid for understanding a broad class of algorithms used to derive insights from graphs. In the case of genome assembly, the ant's path traces a series of overlapping reads, and thus represents a candidate assembly. Specifically, if the ant follows the path ATGGCGT → GGCGTGC → CGTGCAA → TGCAATG → CAATGGC →

Box 1 Origin of de Bruijn graphs

In 1946, the Dutch mathematician Nicolaas de Bruijn became interested in the 'superstring problem'[12]: find a shortest circular 'superstring' that contains all possible 'substrings' of length k (k-mers) over a given alphabet. There exist n^k k-mers in an alphabet containing n symbols: for example, given the alphabet comprising A, T, G and C, there are $4^3 = 64$ trinucleotides. If our alphabet is instead 0 and 1, then all possible 3-mers are simply given by all eight 3-digit binary numbers: 000, 001, 010, 011, 100, 101, 110, 111. The circular superstring 0001110100 not only contains all 3-mers but also is as short as possible, as it contains each 3-mer exactly once. But how can one construct such a superstring for all k-mers in the case of an arbitrary value of k and an arbitrary alphabet? De Bruijn answered this question by borrowing Euler's solution of the Bridges of Königsberg problem. Briefly, construct a graph B (the original graph called a de Bruijn graph) for which every possible $(k-1)$-mer is assigned to a node; connect one $(k-1)$-mer by a directed edge to a second $(k-1)$-mer if there is some k-mer whose prefix is the former and whose suffix is the latter (**Fig. 2**). Edges of the de Bruijn graph represent all possible k-mers, and thus an Eulerian cycle in B represents a shortest (cyclic) superstring that contains each k-mer exactly once. By checking that the indegree and outdegree of every node in B equals the size of the alphabet, we can verify that B contains an Eulerian cycle. In turn, we can construct an Eulerian cycle using Euler's algorithm, therefore solving the superstring problem. It should now be apparent why the 'de Bruijn graph' construction described in the main text, which does not use all possible k-mers as edges but rather only those generated from our reads, is also named in honor of de Bruijn.

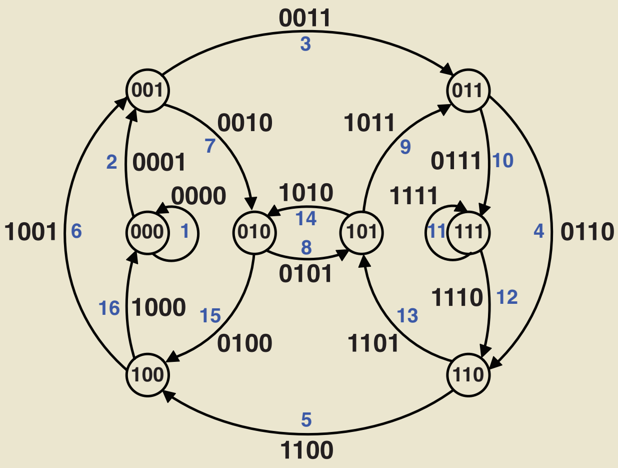

Figure 2 De Bruijn graph. The de Bruijn graph B for $k = 4$ and a two-character alphabet composed of the digits 0 and 1. This graph has an Eulerian cycle because each node has indegree and outdegree equal to 2. Following the blue numbered edges in order from 1 to 16 traces an Eulerian cycle 0000, 0001, 0011, 0110, 1100, 1001, 0010, 0101, 1011, 0111, 1111, 1110, 1101, 1010, 0100, 1000. Recording the first character (in boldface) of each edge label spells the cyclic superstring **0000110010111101**.

ATGGCGT, its walk induces a 'Hamiltonian cycle' in our graph, which is a path that travels to every node exactly once and ends at the starting node, meaning that each read will be included once in the assembly. The circular genome ATGGCGTGCA, which is computed by concatenating the first two nucleotides in each read in such a Hamiltonian cycle, contains all five reads and thus reconstructs the original genome (although we may have to 'wrap around' the genome, for example, to locate CAATGGC in ATGGCGTGCA).

Modern assemblers usually work with strings of a particular length k (k-mers), which are shorter than entire reads (see **Box 2** for an explanation of why researchers prefer k-mers to reads). For example, a 100-nucleotide read may be divided into 46 overlapping 55-mers. The Hamiltonian cycle approach can be generalized to make use of k-mers by constructing a graph as follows. First, from a set of reads, make a node for every k-mer appearing as a consecutive substring of one of these reads (e.g., in **Fig. 3**, ATG, TGG, GGC, GCG, CGT, GTG, TGC, GCA, CAA and AAT). Second, given a k-mer, define its 'suffix' as the string formed by all its nucleotides except the first one and its 'prefix' as the string formed by all of its nucleotides except the last one. Connect one k-mer to another using a directed edge if the suffix of the former equals the prefix of the latter—that is, if the two k-mers completely overlap except for one nucleotide at each end (**Fig. 3c**). Third, look for a Hamiltonian cycle, which represents a candidate genome because it visits each detected k-mer; moreover, that path will also have minimal length because a Hamiltonian cycle travels to each k-mer exactly once.

This method, however, is not as easy to implement as it might seem. Imagine attempting to create a similar graph for a single run of an Illumina (San Diego) sequencer that generates many reads. A million (10^6) reads will require a trillion (10^{12}) pairwise alignments. A billion (10^9) reads necessitate a quintillion (10^{18}) alignments. What's more, there is no known efficient algorithm for finding a Hamiltonian cycle in a large graph with millions (let alone billions) of nodes. The Hamiltonian cycle approach[5,6] was feasible for sequencing the first microbial genome[7] in 1995 and the human genome in 2001, as well as for all other projects based on Sanger sequencing. Even so, the computational burden of this approach was so large that most next-generation sequencing projects have abandoned it.

And here is where genome sequencing faces the limits of modern computer science: the computational problem of finding a Hamiltonian cycle belongs to a class of problems that are collectively called *NP*-Complete (see ref. 2 for further background). To this day, some of the world's top computer scientists have worked to find an efficient solution to any *NP*-Complete problem, with no success. What makes their failure doubly frustrating is that no one has even been able to prove that *NP*-Complete problems are intractable; efficient solutions to these problems may actually exist, but such solutions have not yet been discovered.

Scalable assembly with de Bruijn graphs

As noted in the previous section, finding a cycle that visits all nodes of a graph exactly once (called the Hamiltonian cycle problem) is a difficult computational problem; however, as we will soon see, finding a cycle that visits all edges of a graph exactly once is much easier. This algorithmic contrast has motivated computer scientists to cast DNA fragment assembly as such a problem. Instead of assigning each k-mer contained in some read to a node, we will now assign each such k-mer to an edge. This allows the construction of a 'de Bruijn graph' as follows. First, form a node for every distinct prefix or suffix of a k-mer, meaning that a given sequence of length k–1

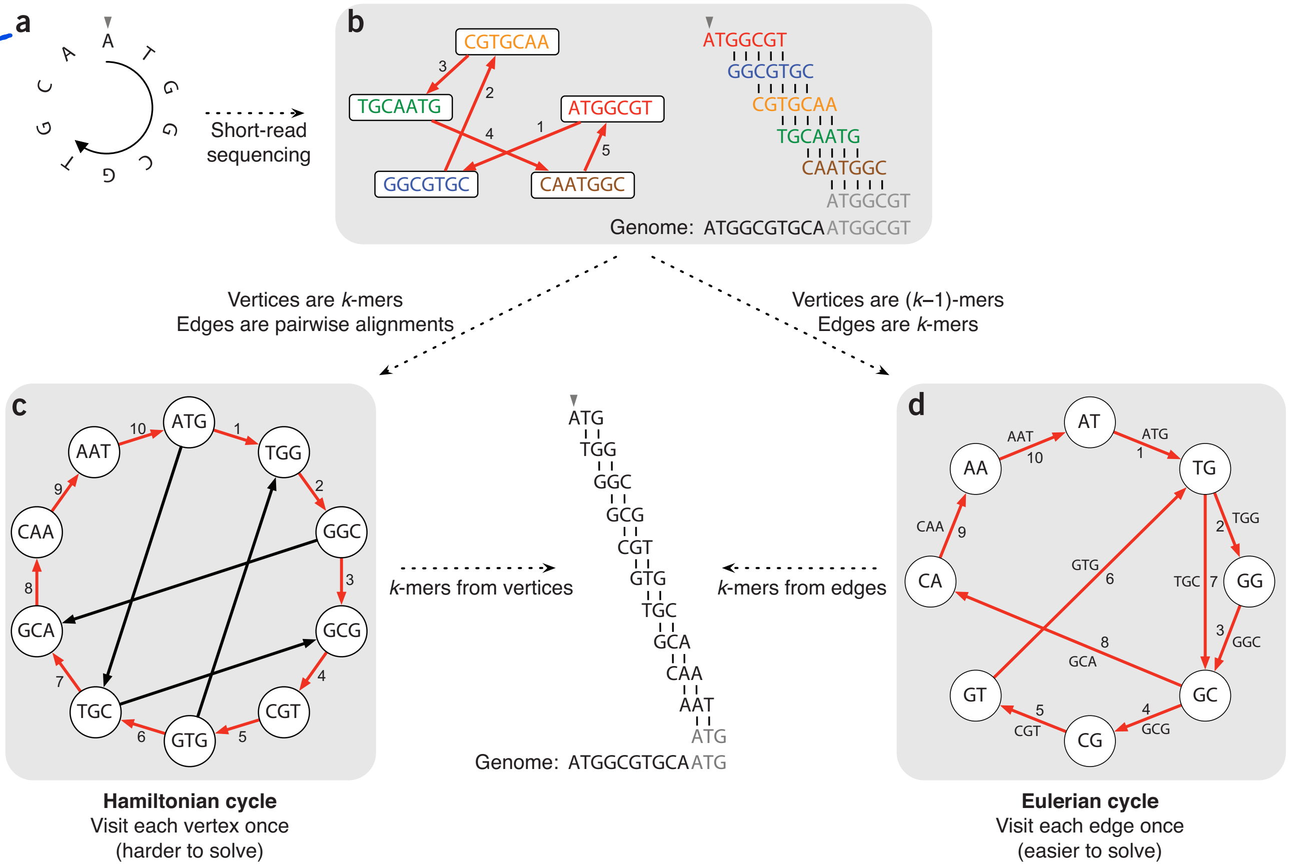

Figure 3 Two strategies for genome assembly: from Hamiltonian cycles to Eulerian cycles. (**a**) An example small circular genome. (**b**) In traditional Sanger sequencing algorithms, reads were represented as nodes in a graph, and edges represented alignments between reads. Walking along a Hamiltonian cycle by following the edges in numerical order allows one to reconstruct the circular genome by combining alignments between successive reads. At the end of the cycle, the sequence wraps around to the start of the genome. The repeated part of the sequence is grayed out in the alignment diagram. (**c**) An alternative assembly technique first splits reads into all possible *k*-mers: with *k* = 3, ATGGCGT comprises ATG, TGG, GGC, GCG and CGT. Following a Hamiltonian cycle (indicated by red edges) allows one to reconstruct the genome by forming an alignment in which each successive *k*-mer (from successive nodes) is shifted by one position. This procedure recovers the genome but does not scale well to large graphs. (**d**) Modern short-read assembly algorithms construct a de Bruijn graph by representing all *k*-mer prefixes and suffixes as nodes and then drawing edges that represent *k*-mers having a particular prefix and suffix. For example, the *k*-mer edge ATG has prefix AT and suffix TG. Finding an Eulerian cycle allows one to reconstruct the genome by forming an alignment in which each successive *k*-mer (from successive edges) is shifted by one position. This generates the same cyclic genome sequence without performing the computationally expensive task of finding a Hamiltonian cycle.

(e.g., AT, TG, GG, GC, CG, GT, CA and AA) can appear only once as a node of the graph. Then, connect node *x* to node *y* with a directed edge if some *k*-mer (e.g., ATG) has prefix *x* (e.g., AT) and suffix *y* (e.g., TG), and label the edge with this *k*-mer (**Fig. 3d**; in **Box 3**, we describe how this approach was originally discussed in the context of sequencing by hybridization).

Now imagine an ant that follows a different strategy: instead of visiting every node of the graph (as before), it now attempts to visit every edge of the graph exactly once. Sound familiar? This is exactly the kind of path that would solve the Bridges of Königsberg problem and is called an Eulerian cycle. As it visits all edges of the de Bruijn graph, which represent all possible *k*-mers, this new ant also spells out a candidate genome; for each edge that the ant traverses, one records the first nucleotide of the *k*-mer assigned to that edge.

Euler considered graphs for which there exists a path between every two nodes (called connected graphs). He proved that a connected graph with undirected edges contains an Eulerian cycle exactly when every node in the graph has an even number of edges touching it. For the Königsberg Bridge graph (**Fig. 1b**), this is not the case because each of the four nodes has an odd number of edges touching it and so the desired stroll through the city does not exist.

The case of directed graphs (that is, graphs with directed edges) is similar. For any node in a directed graph, define its indegree as the number of edges leading into it and its outdegree as the number of edges leaving it. A graph in which indegrees are equal to outdegrees for all nodes is called 'balanced'. Euler's theorem states that a connected directed graph has an Eulerian cycle if and only if it is balanced. In particular, Euler's theorem implies that our de Bruijn graph contains an Eulerian cycle as long as we have located all *k*-mers present in the genome. Indeed, in this case, for any node, both its indegree and outdegree represent the number of times the (*k*–1)-mer assigned to that node occurs in the genome.

To see why Euler's theorem must be true, first note that a graph that contains an Eulerian cycle is balanced because every time an ant traversing an Eulerian cycle passes through a particular vertex, it enters on one edge of the cycle and exits on the next edge. This pairs up all the edges touching each vertex, showing that half the edges touching the vertex lead into it and half lead out from it. It is a bit harder to see the converse—that every connected balanced graph contains an Eulerian cycle. To prove this fact, Euler sent an ant to randomly explore the graph under a single constraint: the ant cannot traverse a previously traversed edge. Sooner or later, the ant must get stuck at a certain node (with all outgoing edges previously traversed) and Euler noticed that because the graph is balanced, this node where the ant gets stuck is exactly the vertex where it started, no matter how the ant traveled through the graph. This implies that the ant has completed a cycle; if this cycle happens to traverse all edges, then the ant has found an Eulerian cycle! If this cycle does not traverse all of the edges, Euler sent another ant to randomly traverse unexplored edges and thereby to trace a second cycle in the graph. Euler further showed that the two cycles discovered by the two ants can be combined

Box 2 Practical strategies for applying de Bruijn graphs

In practice, attempting to apply de Bruijn graphs to experimental data is not a straightforward procedure. We describe some key computational techniques that have been devised to address the practical challenges introduced by errors and quirks in current sequencing technologies, as well as to resolve the complexities created by repeat-rich genomes. For instance, the astute observer will have noticed that the de Bruijn method for fragment assembly relies upon four hidden assumptions that do not hold for next-generation sequencing. We took for granted that we can generate all *k*-mers present in the genome, that all *k*-mers are error free, that each *k*-mer appears at most once in the genome and that the genome consists of a single circular chromosome. For example, Illumina technology, which generates 100-nucleotide long reads, may miss some 100-mers present in the genome (even if the read coverage is high) and the 100-mers that it does generate typically have errors.

Generating (nearly) all *k*-mers present in the genome. Reads of 100-mers generated by Illumina technology capture only a small fraction of 100-mers from the genome (even for samples sequenced to high coverage), thus violating the key assumption of de Bruijn graphs. However, if one breaks these reads into shorter *k*-mers, the resulting *k*-mers often represent nearly all *k*-mers from the genome for sufficiently small *k*. For example, de Bruijn graph–based assemblers may break every 100-nucleotide read into 46 overlapping 55-mers and further assemble the resulting 55-mers. Even if some 100-mers occurring in the genome are not generated as reads, this ‘read breaking’ procedure[13] ensures that nearly all 55-mers appearing in the genome are detected. In the example shown in **Figure 3**, the five reads do not account for all 7-mer substrings of the genome. But they do contain all 3-mers present in the genome, and this is sufficient to reconstruct the genome.

Handling errors in reads. Each error in a read creates a ‘bulge’ in the de Bruijn graph (**Supplementary Fig. 1**), complicating assembly. To make matters even worse, in a genome with inexact repeats (e.g., two regions differing by a single nucleotide or other small variation), reads from the two repeat copies will also generate bulges in the de Bruijn graph. An approach for ‘error correcting’ reads, in which errors are resolved before even beginning assembly, was proposed[14] in 2001, and it is now commonly applied. An approach for removing bulges from de Bruijn graphs was outlined[15] in 2004 and, with some variations, is used in most existing short-read assemblers (e.g., EULER-SR[16], Velvet[17], ALLPATHS[18], ABySS[19] and SOAPdenovo[20]). These and other recently developed tools introduced many new algorithmic and software engineering ideas in assembly algorithms and paved the way toward assembling large (e.g., mammalian) genomes with next-generation sequencing data (see refs. 21,22 for a detailed comparison of these and other assemblers).

Handling DNA repeats. Imagine sequencing 3-base reads from the cyclic genome, ATGCATGC. This should yield the four 3-mers: ATG, TGC, GCA and CAT. The present definition of de Bruijn graphs, however, would lead us to reconstruct the genome as ATGC. The problem is that each of the 3-mers actually occurs twice in the original genome. Therefore, we will need to adjust genome reconstruction so that we not only find all *k*-mers occurring in the genome, but we also find how many times each such *k*-mer appears, which is called its ‘*k*-mer multiplicity’. The good news is that we can still handle fragment assembly in the case when *k*-mer multiplicities are known. We simply use the same method to construct the de Bruijn graph, except that if the multiplicity of a *k*-mer is *m*, we will connect its prefix to its suffix using *m* directed edges (instead of just one). Extending the example in **Figure 3**, if we discover during read generation that each of the four 3-mers TGC, GCG, CGT and GTG has multiplicity of two, and that each of the six 3-mers ATG, TGG, GGC, GCA, CAA and AAT has multiplicity of one, we create the graph shown in **Supplementary Figure 2**. Furthermore, the graph resulting from adding multiplicity edges is balanced (and therefore contains an Eulerian cycle), as both the indegree and outdegree of a node (representing a (k–1)-mer) equals the number of times this (k–1)-mer appears in the genome.

In practice, information about the multiplicities of *k*-mers in the genome may be difficult to obtain with existing sequencing technologies. However, computer scientists have found ways to reconstruct the genome, even when these data are unavailable. One such technique involves ‘paired reads’. Reads are typically generated in pairs by sequencing both ends of a long fragment of DNA whose length can be estimated well. If one read maps at or before the entrance to a repeat in the graph, and the other maps at or after the exit, the read pair may be used to determine the correct traversal through the graph.

Handling multiple and linear chromosomes. We have discussed examples in which the genome consists of one circular chromosome. If instead the chromosome is linear, then we will need to search for an Eulerian path, instead of an Eulerian cycle; an Eulerian path is not required to end at the node where it begins. If there are multiple linear chromosomes, then we will have one path for each chromosome. Euler’s work can be adapted to handle these complexities.

Handling unsequenced regions. Regions that are not sequenced and sequencing errors may further break the chromosomes into contigs (a sequenced contiguous region of DNA) and gaps (unsequenced regions), with one path for each contig. Increasing the value of *k* will tend to reduce the number of bulges and give longer contigs in places with high coverage but some errors. Even so, it will also tend to break contigs in regions that have low coverage. Successive contigs along a chromosome may have overlaps of fewer than *k* nucleotides, or they may have gaps between them. The correct order and orientation of the contigs, and the approximate sizes of the gaps, is determined in the scaffolding phase of assembly. This phase uses additional information, including paired reads, to determine the order of contigs.

Box 3 Sequencing by hybridization

Few people remember that the idea of DNA sequencing using short reads dates back to the late 1980s. In fact, the very first short-read sequencing approach, sequencing by hybridization[23–25], aimed to achieve genome assembly in this fashion. Sequencing by hybridization, proposed in 1988, is based on building a microarray containing every possible oligonucleotide of length *k*. After hybridization of such an array with an unknown genome, one would get information about all *k*-mers present in the genome. De Bruijn graphs were first brought to bioinformatics in 1989 as a method to assemble *k*-mers generated by sequencing by hybridization[26]; this method is very similar to the key algorithmic step of today's short-read assemblers.

DNA arrays ultimately failed to realize the dream (DNA sequencing) that motivated their inventors because the fidelity of DNA hybridization with the array turned out to be too low and the value of *k* was too small. Yet the failure of DNA arrays was short-lived. Although their original goal (DNA sequencing) was still out of reach, two new unexpected applications emerged: measuring gene expression and analyzing genetic variations. Today, these applications have become so ubiquitous that most people have forgotten that the original goal of the inventors of DNA arrays was to sequence the human genome!

into a single cycle. If this larger, combined cycle contains all of the edges in the graph, then the two ants have together found an Eulerian cycle! If not, Euler's method keeps recruiting additional ants until all of the graph's edges have been explored, at which point the ants' cycles can be combined to form an Eulerian cycle.

On modern computers, this algorithm can efficiently find Eulerian cycles in huge balanced graphs having billions of nodes, thus avoiding the quagmire of *NP*-Completeness. Therefore, simply recasting our original problem into a slightly different framework converts genome assembly into a tractable computational problem.

The time required to run a computer implementation of Euler's algorithm is roughly proportional to the number of edges in the de Bruijn graph. In the Hamiltonian approach, the time is potentially a lot larger, because of the large number of pairwise alignments needed to construct the graph and the *NP*-Completeness of finding a Hamiltonian cycle. A more detailed comparison of these approaches is given in reference 8.

Practical matters

De Bruijn graphs are not a cure-all. Throughout our exposition, we have made several simplifying assumptions, which require much work to formally validate. Yet for every apparent complication to sequence assembly, it has proven fruitful to apply some cousin of de Bruijn graphs to transform a question involving Hamiltonian cycles into a different question regarding Eulerian cycles (**Box 2, Supplementary Figs. 1** and **2**). Moreover, analogs of de Bruijn graphs have been useful in many other bioinformatics problems, including antibody sequencing[9], synteny block reconstruction[10] and RNA assembly[11]. In each of these applications, the de Bruijn graph represents the experimental data in a manner that leads to a tractable computational problem.

As new sequencing technologies emerge, the best computational strategies for assembling genomes from reads may change. The factors that influence the choice of algorithms include the quantity of data (measured by read length and coverage); quality of data (including error rates); and genome structure (e.g., GC content and the number and size of repeated regions). Short-read sequencing technologies produce very large numbers of reads, which currently favor the use of de Bruijn graphs. De Bruijn graphs are also well suited to representing genomes with repeats, whereas overlap methods need to mask repeats that are longer than the read length. However, if a future sequencing technology produces high-quality reads with tens of thousands of bases, a smaller number of reads would be needed, and the pendulum could swing back toward favoring overlap-based approaches for assembly.

Note: Supplementary information is available on the Nature Biotechnology website.

ACKNOWLEDGMENTS
This work was supported by grants from Howard Hughes Medical Institute (HHMI grant 52005726), the US National Institutes of Health (NIH grant 3P41RR024851-02S1) and the National Science Foundation (NSF grant DMS-0718810). We are grateful to S. Wasserman for many helpful comments.

COMPETING FINANCIAL INTERESTS
The authors declare no competing financial interests.

1. Euler, L. *Commentarii Academiae Scientiarum Petropolitanae* **8**, 128–140 (1741).
2. Skiena, S. *The Algorithm Design Manual* (Springer, Berlin, 2008).
3. Lander, E. *et al. Nature* **409**, 860–921 (2001).
4. Venter, J.C. *et al. Science* **291**, 1304–1351 (2001).
5. Kececioglu, J. & Myers, E. *Algorithmica* **13**, 7–51 (1995).
6. Adams, M. *et al. Science* **287**, 2185–2195 (2000).
7. Fleischmann, R. *et al. Science* **269**, 496–512 (1995).
8. Schatz, M., Delcher, A. & Salzberg, S. *Genome Res.* **20**, 1165–1173 (2010).
9. Bandeira, N., Pham, V., Pevzner, P., Arnott, D. & Lill, J. *Nat. Biotechnol.* **26**, 1336–1338 (2008).
10. Pham, S. & Pevzner, P.A. *Bioinformatics* **26**, 2509–2516 (2010).
11. Grabherr, M. *et al. Nat. Biotechnol.* **29**, 644–652 (2011).
12. de Bruijn, N. *Proc. Nederl. Akad. Wetensch.* **49**, 758–764 (1946).
13. Idury, R. & Waterman, M. *J. Comput. Biol.* **2**, 291–306 (1995).
14. Pevzner, P.A., Tang, H. & Waterman, M. *Proc. Natl. Acad. Sci. USA* **98**, 9748–9753 (2001).
15. Pevzner, P.A., Tang, H. & Tesler, G. *Genome Res.* **14**, 1786–1796 (2004).
16. Chaisson, M. & Pevzner, P.A. *Genome Res.* **18**, 324–330 (2008).
17. Zerbino, D. & Birney, E. *Genome Res.* **18**, 821–829 (2008).
18. Butler, J. *et al. Genome Res.* **18**, 810–820 (2008).
19. Simpson, J. *et al. Genome Res.* **19**, 1117–1123 (2009).
20. Li, R. *et al. Genome Res.* **20**, 265–272 (2010).
21. Paszkiewicz, K. & Studholme, D. *Brief. Bioinform.* **11**, 457–472 (2010).
22. Miller, J., Koren, S. & Sutton, G. *Genomics* **95**, 315–327 (2010).
23. Drmanac, R., Labat, I., Brukner, I. & Crkvenjakov, R. *Genomics* **4**, 114–128 (1989).
24. Southern, E. United Kingdom patent application gb8810400 (1988).
25. Lysov, Y. *et al. Doklady Academy Nauk USSR* **303**, 1508–1511 (1988).
26. Pevzner, P.A. *J. Biomol. Struct. Dyn.* **7**, 63–73 (1989).