

Phase transition in the computational complexity of the shortest common superstring and genome assembly

L. A. Fernandez ^{1,2}, V. Martin-Mayor,^{1,2} and D. Yllanes ^{3,2}

¹*Departamento de Física Teórica, Universidad Complutense, 28040 Madrid, Spain*

²*Instituto de Biocomputación y Física de Sistemas Complejos (BIFI), 50018 Zaragoza, Spain*

³*Chan Zuckerberg Biohub — SF, 499 Illinois Street, San Francisco, California 94158, USA*



(Received 17 April 2023; accepted 11 December 2023; published 24 January 2024)

Genome assembly, the process of reconstructing a long genetic sequence by aligning and merging short fragments, or reads, is known to be NP-hard, either as a version of the shortest common superstring problem or in a Hamiltonian-cycle formulation. That is, the computing time is believed to grow exponentially with the problem size in the worst case. Despite this fact, high-throughput technologies and modern algorithms currently allow bioinformaticians to handle datasets of billions of reads. Using methods from statistical mechanics, we address this conundrum by demonstrating the existence of a phase transition in the computational complexity of the problem and showing that practical instances always fall in the “easy” phase (solvable by polynomial-time algorithms). In addition, we propose a Markov-chain Monte Carlo method that outperforms common deterministic algorithms in the hard regime.

DOI: [10.1103/PhysRevE.109.014133](https://doi.org/10.1103/PhysRevE.109.014133)

I. INTRODUCTION

Sequence assembly is one of the fundamental problems in bioinformatics. Since an organism’s whole genetic material cannot be read in one go, current technologies build on strategies where the genome (or a portion of it, such as a chromosome) is randomly fragmented in shorter reads, which then have to be ordered and merged to reconstruct the original sequence. In a naive formulation, one would look for the shortest sequence that contains all of the individual reads, or shortest common superstring (SCS). This is, in principle, a formidable task, since the SCS belongs to the so-called NP-complete class of problems [1,2], for which no efficient algorithms are known (or believed) to exist. More precisely, NP denotes a large family of problems that are verifiable in polynomial time, meaning that potential solutions can be checked in a time that grows at most as a power of the size of the input. A problem is then termed NP-complete if it is in NP and at least as hard as any problem in NP. This is a relevant notion because it is believed, but not proven, that not all NP-complete tasks can be solved in polynomial time. See, e.g., Refs. [3–5] for more precise definitions and examples.

As hinted above, the formulation of genome assembly as an SCS problem is not quite correct. This is because our assumption of parsimony is not true: most genomes contain *repeats*, multiple identical stretches of DNA, which the SCS would collapse. A formulation of the assembly problem that takes this issue into account can be made using de Bruijn graphs [6], but the task can still be proven to be NP-hard by reduction from SCS [7]. Alternative approaches to assembly have been proposed, for instance the string-graph representation [8], but this model has also been shown to be NP-hard, by reduction from the Hamiltonian-cycle problem [7].

In short, no polynomial-time algorithms are known (or even believed) to exist to solve the sequence-assembly problem in its general formulation. Despite this fact, with current high-throughput sequencing techniques and assembly algorithms, datasets of billions of reads are regularly assembled (at least at the contig level) [9–14]. This achievement is in stark contrast with the state of the art for the traveling salesman, a closely related NP-complete problem, for which managing as few as 10^4 “cities” is exceedingly difficult and the largest instance solved to date featured 120 000 locations [15,16].

The way out of this apparent contradiction is the general notion that, while in the worst case an NP-complete problem takes exponential time to solve, typical instances might be much easier. This observation could explain the, at least apparent, success of heuristic methods but it needs to be formalized: what is a “typical” instance and how likely is the worst-case scenario? As a path towards answering those questions, it has been observed that in several problems in computational biology, small ranges of parameters cover all the interesting cases. The question is, then, whether we can identify the right variables and whether the problem can be solved in polynomial time for fixed values of these parameters [17]. This parametric complexity paradigm has been applied to genome assembly, either using statistical analyses or analytical methods, suggesting that, in some relevant limits, the problem can indeed be solved correctly with polynomial algorithms [18–20].

In this work we present an alternative approach to this question, based on the methods of statistical mechanics. The applicability of the models of statistical physics to the issue of NP-completeness has been conjectured since the 1980s [21,22] and is now well understood [23]. More to the point, phase diagrams for complexity can be defined for some problems. For instance, in a seminal work [24] all the constraints

in the problem can be satisfied only when a certain parameter is smaller than its critical value, and the computationally hard problems arise only in the neighbourhood of the phase boundary.

We show that a similar result can be obtained for the SCS. Yet, we depart from the paradigm of Ref. [24] in the sense that computationally hard problems become the rule, rather than the exception, in one of our two phases, not just at the critical point. It is worth emphasizing that, unlike assembly, the SCS problem *always* has a well-defined solution, which might not be unique (unfortunately, in some regions of parameter space this solution is very hard to find). Instead, the assembly problem is well posed only if the whole genome is covered. It is a classic result [25] that, if L is the length of our (portion of) genome and ℓ_{frag} the length of the reads, to ensure that the whole genome is covered with probability $1 - \epsilon$, the number N_{frag} of reads must satisfy $N_{\text{frag}} = (L/\ell_{\text{frag}}) \log(N_{\text{frag}}/\epsilon)$. Therefore, in practical applications one is interested in oversampling the genome (the oversampling ratio is termed coverage). Our main result in this respect is that the regime of full coverage corresponds precisely to the easily solvable phase of the SCS problem. Therefore, whenever the assembly problem is well posed, the corresponding solution can be found in polynomial time. A final note of warning is in order: we shall always use assembly language (genomes and reads, rather than superstrings and strings) to unify the nomenclature. The remainder of this paper is organized as follows. In Sec. II we define the two problems considered here, the SCS and the assembly. In Sec. III we discuss our data collection and computational approaches. The analysis of the phase transition is presented in Sec. IV. An alternative algorithm for the hard phase of the SCS is presented in Sec. V. Our conclusions are given in Sec. VI. We provide additional details on the employed algorithms in the Appendices.

II. THE SCS AND SEQUENCE ASSEMBLY

As explained in the introduction, there are two different, yet related problems:

(i) *Shortest common superstring (SCS)*. Given N_{frag} sequences of ℓ_{frag} letters taken from a common alphabet, find the shortest sequence of letters that contains every one of the N_{frag} fragments.

(ii) *Ex novo genome reconstruction*. Read N_{frag} fragments randomly chopped from a piece of genome. For simplicity, we shall assume that each fragment contains the same number of letters ℓ_{frag} . Our problem is reconstructing the original genome from these reads.

Under favourable circumstances on the ensemble of reads, the solution of the SCS problem is also the solution of the assembly problem. Our main emphasis will be in the combinatorial optimisation problem, namely the SCS. Reading errors are a real complication in assembly, but effective methods are known to handle them. Since errors do not add to the exponential (in genome size) hardness of the problem, we shall ignore them.

We study the SCS in its formulation as an asymmetric travelling-salesman problem, where one tries to find the permutation of fragments that has the maximum overlap between consecutive segments and, therefore, the minimal total

length of the resulting superstring once overlapping segments are collapsed. For instance, the SCS of the strings TTGAA, AGTTG is AGTTGAA. Our reads are taken from a circular genome of length L base pairs (we use the natural four-letter alphabet A, C, G, and T). For our main study we choose all bases in the alphabet randomly (independent picks with uniform probability), but we have also checked that our main results extend to a natural genome, namely that of the swinepox virus.

A naive approach to an assembly problem emphasizes the covering fraction:

$$W = \frac{N_{\text{frag}} \ell_{\text{frag}}}{L}. \quad (1)$$

$W < 1$ implies that the SCS is shorter than the genome (obviously, successful assembly is impossible under these circumstances). In typical instances of assembly $W \gg 1$ ($W \sim 1000$ is not uncommon with high-throughput techniques).

Given a set of reads, obtaining the SCS is NP-hard. Since we are taking our fragments from a known long string, however, we always have a candidate solution, as we now explain. Since the genome is known to us beforehand, we can exactly locate the position in the genome of every fragment. If we order these starting points in increasing order, we obtain by merging overlaps a candidate solution for the SCS. We name ℓ_{ordered} the length of the candidate solution, which often turns out to be the exact solution ($\ell_{\text{ordered}} = L$) when $W \gg 1$. This is the common situation in applications. Yet, when $W < 1$, ℓ_{ordered} is guaranteed to be smaller than L . Furthermore, the ordered sequence may actually be a very bad solution for SCS problem, when $W \ll 1$. Indeed, in the limit $W \rightarrow 0$, nothing distinguishes the ordered solution from a random ordering. In the intermediate regime, $W \sim 1$, the ordered sequence is a good guess for the SCS (and for assembly).

For a given algorithm, a run whose resulting superstring length ℓ is $\ell \leq \ell_{\text{ordered}}$ will be considered successful. In practice, in the $W \gg 1$ region, one never finds $\ell < \ell_{\text{ordered}} = L$ and the original genome coincides with the SCS.

III. CREATING AND SAMPLING THE DATA SET

The main classifying feature for our simulations is the number of fragments, N_{frag} . For every N_{frag} we create a set of N_{chro} synthetic circular chromosomes. As discussed above, we have considered both random chromosomes—in which each letter is extracted with uniform probability randomly from the four-letter alphabet—or extracted reads from the genome of the swinepox virus (downloaded from GenBank, Accession No. NC_003389).

To generate the fragments, we proceed as follows. We independently and randomly generate N_{frag} integers uniformly distributed in $\{1, 2, \dots, L\}$ (L is the length of the *circular* chromosome). Each integer is regarded as the starting point of a fragment of length ℓ_{frag} .

We have analyzed our data using two algorithms: a version of the greedy algorithm, which we name *Glotón*, and *Velvet* [26], a commonly used program for genome assembly based on de Bruijn graphs (see Appendix A for a description of the *Glotón* algorithm and Appendix B for details on our simulation parameters with *Velvet*). It will be important

that *Glotion* has a stochastic component, while *Velvet* is deterministic.

We attempt to reconstruct the chromosome from this set of reads N_{attempts} times (we generate just one set of reads for each chromosome; we set $N_{\text{attempts}} = 1$ for *Velvet*). The success probability for a given chromosome is the fraction of the N_{attempts} assembly attempts that meet our success criterion $\ell \leq \ell_{\text{ordered}}$. Specifically, let $\ell_{i,j}$ be the length obtained on the j th attempt for the i th chromosome. We have for *Glotion*

$$p_{\text{success}}^{(i)} = \frac{1}{N_{\text{attempts}}} \sum_{j=1}^{N_{\text{attempts}}} \mathbf{1}(\ell_{i,j} \leq \ell_{\text{ordered}}), \quad (2)$$

where $\mathbf{1}$ is the indicator function. For the deterministic *Velvet*, we adopt a slightly different definition, see Appendix B, such that $p_{\text{success}}^{(i)} = 0, 1$. From now on, we shall refer to $p_{\text{success}}^{(i)}$ as the individual success probability. The total success probability is just the average over the N_{chro} chromosomes of the individual success rates:

$$p_{\text{success}} = \frac{1}{N_{\text{chro}}} \sum_{i=1}^{N_{\text{chro}}} p_{\text{success}}^{(i)}. \quad (3)$$

We compute in a similar way the variance—or covariance—of the individual success probabilities.

We have set $N_{\text{attempts}} = 100$ for the *Glotion* and *segment-swap* algorithms (described below) and, as we said above, $N_{\text{attempts}} = 1$ for *Velvet*. We use $N_{\text{chro}} = 10\,000$ for *Glotion* (the only exception is in Fig. 3, where $N_{\text{chro}} = 100\,000$ for $N_{\text{frag}} \leq 800$). However, we have contented ourselves with $N_{\text{chro}} = 1000$ for the costlier *Velvet* and *segment-swap* algorithms (in Fig. 5, for $N_{\text{frag}} = 800$ and *segment-swap*, we use $N_{\text{chro}} = 100$).

IV. THE SUCCESS PROBABILITY AND A PHASE TRANSITION IN THE COMPLEXITY

We want to characterize the hardness of the problem in terms of the success probability p_{success} for a run of a simple algorithm (i.e., one that ends in polynomial time). Here we consider two, namely *Glotion*, and *Velvet*.

It is our goal to understand quantitatively the behavior of p_{success} as a function of L , ℓ_{frag} and N_{frag} . Ideally, one would be able to simplify the three-variable function $p_{\text{success}} = f(L, N_{\text{frag}}, \ell_{\text{frag}})$ into a function of a single *scaling* variable x . A first attempt, shown in Fig. 1 (top), plots p_{success} as a function of W for $\ell_{\text{frag}} = 100$ and various values of N_{frag} for the *Glotion* (see Sec. III for more details on these simulations). We can see two regimes: for large W , this algorithm always succeeds, while for small W it always fails. Recall that, in the large- W regime, success effectively means reconstructing the original genome, while in the small- W regime it means finding a good approximation to the SCS given by ℓ_{ordered} .

Comparing the different curves in Fig. 1 (top), we see that W is not a good scaling variable, since a clear dependence on N_{frag} remains. A more natural candidate is the maximum distance d_{max} between the starting points of reads consecutive in the original genome. Notice that the genome is fully covered by the N_{frag} fragments if and only if $d_{\text{max}} \leq \ell_{\text{frag}}$. For a given

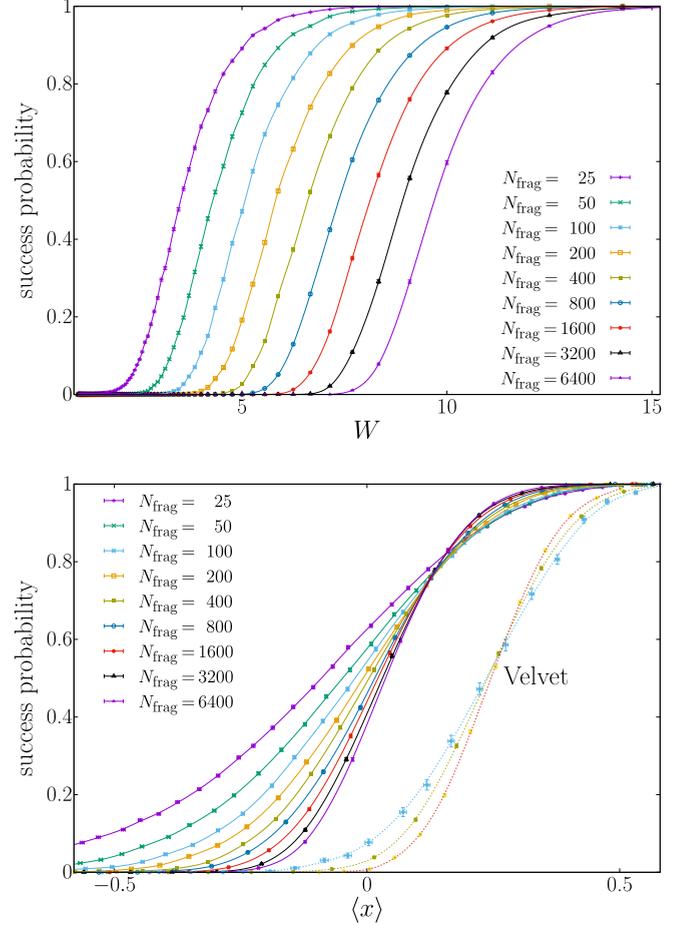


FIG. 1. Performance of common algorithms for the shortest common superstring problem. (Top) Probability of finding a successful solution (see text) using a greedy algorithm as a function of the coverage W , Eq. (1), for several values of the number of fragments (reads) N_{frag} and for fragment length $\ell_{\text{frag}} = 100$. For large coverage values, the algorithm always succeeds. (Bottom) In terms of the correct scaling variable x , Eq. (4), based on the ratio between the average maximum distance between fragments and ℓ_{frag} , the p_{success} curves for different N_{frag} cross, which we interpret as the onset of a phase transition at some critical x_c . The value of x_c is algorithm dependent, but the qualitative behavior is the same for more sophisticated methods. As a demonstration, we also show the results using *Velvet*, which employs an algorithm based on de Bruijn graphs.

realisation of the problem, we define

$$x = 1 - d_{\text{max}}/\ell_{\text{frag}}. \quad (4)$$

We can also define $\langle x \rangle$ as the ensemble average of x for all possible genomes of length L and all possible choppings with N_{frag} and ℓ_{frag} . It can be shown that

$$\langle d_{\text{max}} \rangle \sim \log N_{\text{frag}}/W, \quad (5)$$

for instance by discretization of the continuum calculation in [27]. In each of the different curves of Fig. 1 N_{frag} and ℓ_{frag} are fixed and $\langle x \rangle$ (or W) are varied by changing L .

Plotting p_{success} as a function of $\langle x \rangle$, we can see that the curves for different N_{frag} cross at some x_c , while their shape approaches a step function as N_{frag} increases. This is the

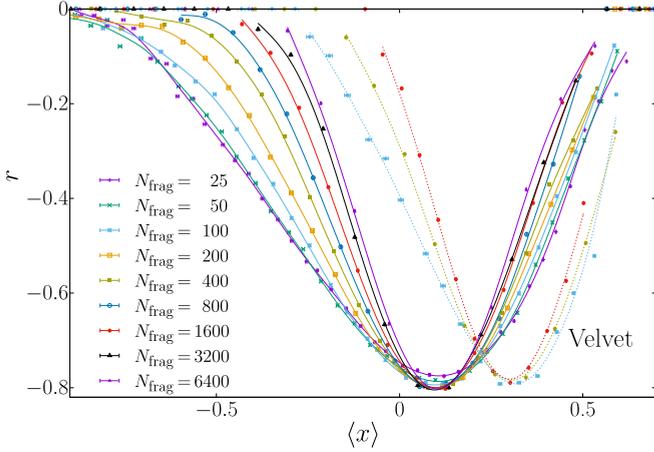


FIG. 2. Location of the critical point. The critical point can be determined by looking for the value of $\langle x \rangle$ where fluctuations are largest. We plot the correlation coefficient between the scaling variable x and the success probability for single realisations of the N_{frag} reads. The absolute value of r has a maximum at the critical point x_c .

characteristic finite-size scaling behaviour at a phase transition [28]. Interestingly enough, the *Glotion* and *Velvet* algorithms have the same behavior, just with a shift in x_c . Again, this is typical of phase transitions, where the critical point depends on the details of the model, but the scaling behavior is universal. We thus propose that, in the large- N_{frag} limit, the SCS problem experiences a phase transition that separates a large- x where polynomial-time algorithms always succeed from a small- x regime where they always fail. All practical applications of genome assembly are in the (very) large- x regime, which explains how it is routinely possible to solve problems that are, in principle, NP-hard with millions of fragments.

We can study the critical regime more quantitatively by looking not just at means but at fluctuations. In particular, we plot in Fig. 2 the correlation coefficient r between x , as computed for a particular set of fragments, and the individual success probability (2) of a polynomial algorithm for that particular set. Away from the critical point, r is very small but in the critical regime a strong (anti)correlation is observed. In fact, the minimum of r as a function of $\langle x \rangle$ is probably the best way of locating x_c . Notice that x_c is lower for *Glotion* than for *Velvet*, which is perhaps unsurprising, since the latter was not designed as an SCS solver (see note in Appendix B).

Finally, notice that, to have a real phase transition, p_{success} should not just be very small, but actually go to zero in the hard phase as N_{frag} increases. With our numerical data, see Fig. 3, we can see that the results are compatible with a power-law decay. However, for $\langle x \rangle$ significantly larger than x_c , the success probability is compatible with 1 even for finite N_{frag} .

The role of the fragment length and comparison with a natural genome

Thus far, we have always used $\ell_{\text{frag}} = 100$ but, given a number of fragments N_{frag} , two parameters remain: the genome length L and ℓ_{frag} or, equivalently, $\langle x \rangle$ and ℓ_{frag} . The beauty of the choice of variables $\langle x \rangle$ and ℓ_{frag} is that the ℓ_{frag}

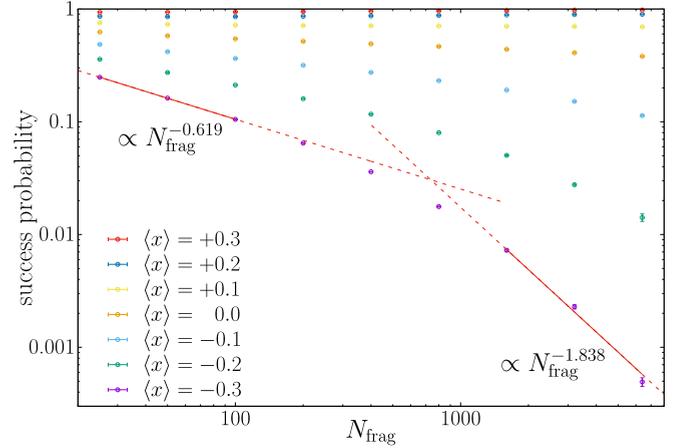


FIG. 3. The success probability goes to zero in the hard phase. If the behavior shown in Fig. 1 corresponds to a phase transition, p_{success} should tend to zero as $N_{\text{frag}} \rightarrow \infty$ for $\langle x \rangle < x_c$. This figure shows that indeed p_{success} decays at least as fast as power law in $1/N_{\text{frag}}$ in the hard phase, while in the easy phase our results are already compatible with $p_{\text{success}} = 1$ for finite sizes.

dependence is residual and vanishes quite fast as ℓ_{frag} grows, as can be seen in Fig. 4. More precisely,

$$p_{\text{success}} \simeq f[\langle x \rangle + A \log N_{\text{frag}} / (N_{\text{frag}} \ell_{\text{frag}})], \quad (6)$$

where A is an algorithm-dependent constant [29]. That is, ℓ_{frag} acts as a scaling correction.

Taking our reads from a random genome or from a real one seems to make no difference. Indeed, our results for the success probability for the *Glotion* using reads sampled from

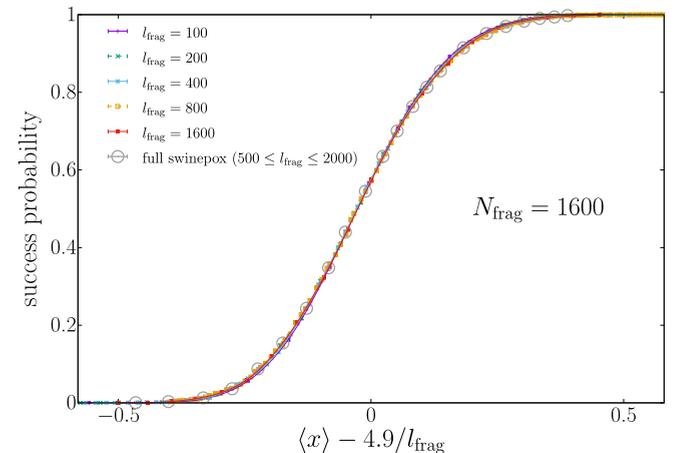


FIG. 4. Varying the fragment length hardly makes any difference. Our previous results have always considered $\ell_{\text{frag}} = 100$. It turns out that the dependence in this parameter is residual and rapidly vanishes as ℓ_{frag} grows, according to Eq. (6). That is, the curves of p_{success} as a function of $\langle x \rangle$ can be collapsed if we subtract the scaling correction caused by finite ℓ_{frag} . We also show that the results for a natural genome (namely that of the swinepox virus) are indistinguishable from those for random sequences. In this case, since L is fixed, we have a single value of p_{success} for each ℓ_{frag} , all of which fall on the rescaled curve.

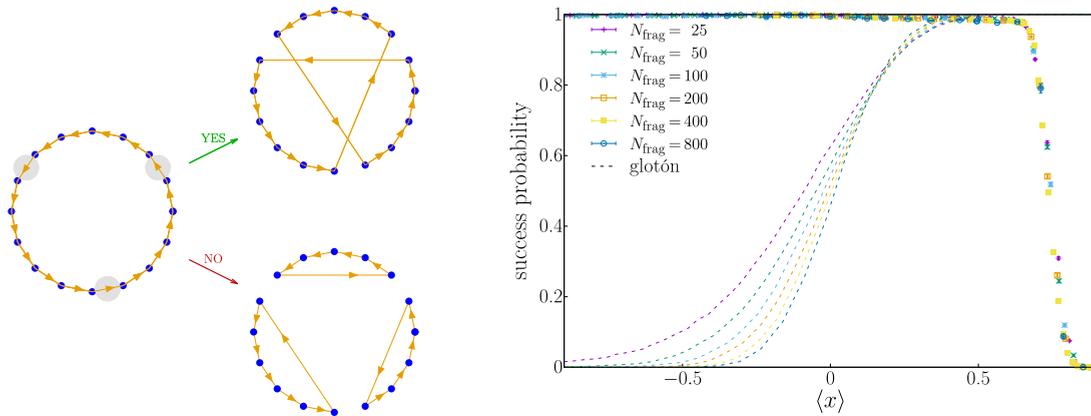


FIG. 5. A Monte Carlo algorithm for the hard cases. We propose a segment-swap Markov-chain Monte Carlo algorithm (sketched in the left panel) that outperforms common deterministic methods in the hard regime (see right panel). We represent the permutation of the reads as an ordered sequence of fragments (the arrows indicate the sense in which the sequence should be toured). The elementary move of the algorithm is composed of the following three steps. First, choose randomly three independent pairs of consecutive fragments (depicted with grey circles in the plot), $\dots \rightarrow \alpha_1 \rightarrow \alpha_2 \rightarrow \dots \rightarrow \beta_1 \rightarrow \beta_2 \rightarrow \dots \rightarrow \gamma_1 \rightarrow \gamma_2 \rightarrow \dots$. Mind the pair ordering: when one tours the circular sequence starting from fragment α_1 , fragments γ_1 and γ_2 are not found earlier than β_1 and β_2 (the choices $\alpha_2 = \beta_1$ and/or $\beta_2 = \gamma_1$ are acceptable). Second, consider the rewired sequence $\dots \rightarrow \alpha_1 \rightarrow \beta_2 \rightarrow \dots \rightarrow \gamma_1 \rightarrow \alpha_2 \rightarrow \dots \rightarrow \beta_1 \rightarrow \gamma_2 \rightarrow \dots$ (there is an unacceptable reconnection—indicated by *NO* in the figure—that would split the sequence into three disconnected cycles). Third step: If the new cycle is not longer than the original one, the segment swap is accepted. As we show in the right panel, segment swap is more effective than other algorithms for $-1 < x < 0.5$. For $x < 0$ the SCS problem no longer corresponds to a full assembly, since there are gaps between reads. The segment-swap algorithm, however, always finds superstrings that satisfy our success criterion ($\ell \leq \ell_{\text{ordered}}$).

the genome of the swinepox virus nicely fall onto the same scaled curve obtained for the random genome.

V. A BETTER ALGORITHM FOR THE HARD PHASE: THE SEGMENT-SWAP

We have seen that, while the SCS problem is NP-hard, it becomes solvable for polynomial-time algorithms in the large- $\langle x \rangle$ regime. For $\langle x \rangle < x_c$, however, common methods always fail even to find a good approximation to the SCS (provided in our success criterion by ℓ_{ordered}). For moderately negative values of x we have found a Markov-chain Monte Carlo algorithm that is both powerful and relatively simple.

The method is sketched in Fig. 5 (left). The segment ordering is actually a circular sequence where we randomly choose three cutting points. There are two ways of reconnecting the three resulting fragments, one of which generates several cycles and can be discarded. The other reconnection generates a single cycle and is potentially a new permutation of the reads that effects nonlocal changes (one would need $\sim N_{\text{frag}}$ transpositions of neighboring reads to generate a single segment-swap move). We accept the new configuration only if its total length is not larger than in the previous step. This is the acceptance criterion of a Metropolis algorithm at zero temperature [see, e.g., Ref. [30]]. As for the stopping condition, note that there $N_{\text{triplets}} = N_{\text{frag}}! / [(N_{\text{frag}} - 3)! 3!]$ possible choices for the cutting points. Whenever the length of the sequence has not decreased for $2N_{\text{triplets}}$ consecutive iterations (where the cutting points are chosen randomly with uniform probability), we check explicitly that none of the N_{triplets} possible moves would decrease the total length. If this is the case, the run is stopped.

In spite of its simplicity, this segment-swap method is very successful in the $-1 < \langle x \rangle < 0.5$ region and, in particular, in the negative $\langle x \rangle$ region where both *Glotion* and *Velvet* fail. The segment-swap method can be generalized by including a fictive temperature [31] and parallel tempering [32]. In this way, one would have a candidate algorithm for treating the $x \rightarrow -\infty$ limit of completely independent reads. Notice that, as x grows more negative, our variational solution ℓ_{ordered} grows worse as an upper bound on the length of the actual SCS. In these cases, the segment-swap algorithm finds solutions with $\ell < \ell_{\text{ordered}}$, but, at least in the simple $T = 0$ version shown in Fig. 5, we cannot be sure that these solutions are the actual SCS.

The reader could worry about completeness: is the segment-swap method capable of reaching all possible configurations? In fact, the transposition of consecutive fragments is a particular case of the segment-swap move. Indeed, take a subsequence $\dots \rightarrow A \rightarrow B \rightarrow C \rightarrow D \rightarrow \dots$ and let us imagine that one randomly selects the pairs $A \rightarrow B$, $B \rightarrow C$ and $C \rightarrow D$ as the ones to be reconnected. In the language of Fig. 5, one would say $\alpha_1 = A$, $\alpha_2 = \beta_1 = B$, $\beta_2 = \gamma_1 = C$, and $\gamma_2 = D$. With this choice, the segment swap results in the transposition of fragments B and C : $\dots \rightarrow A \rightarrow C \rightarrow B \rightarrow D \rightarrow \dots$. Now, since an arbitrary permutation may be obtained from an ordered sequence of transpositions of consecutive fragments, a finite-temperature version of the segment-swap method is ergodic. Our zero-temperature version of the algorithm never accepts a move that increases the total sequence length but, as we said above, this lack of ergodicity causes no problem in the region $-1 < \langle x \rangle < 0.5$ (see Fig. 5). A plausible explanation is that the lack of ergodicity of the zero-temperature dynamics induces another algorithmic phase transition located near $\langle x \rangle = 0.5$.

Another technical question regards the best data structure for implementing the segment-swap algorithm. We have chosen a linked list, because the number of operations needed to change the configuration is independent of N_{frag} . A major drawback, however, is that one basically needs to go through the full linked list to assess which one of the two possible fragment reconnections is acceptable. Our solution is checking the resulting length from both reconnections, *before* going through the list. Indeed, in most cases, both reconnections would enlarge the total length and can be rejected without checking the list, which requires $\mathcal{O}(N_{\text{frag}})$ operations. Nevertheless, for larger N_{frag} than we have considered in this work, it might be advisable to implement the segment-swap algorithm with binary-search trees, which have the potential of turning the computational cost down to $\mathcal{O}(\log N_{\text{frag}})$ operations.

VI. CONCLUSION

We have applied the methods of statistical mechanics to characterize the computational complexity of the SCS problem, showing that, in terms of an appropriate scaling variable $\langle x \rangle$, a phase diagram can be constructed. For $\langle x \rangle > x_c$ the problem is in the easy regime, i.e., it is solvable in polynomial time, while below x_c it is exponentially hard. In the language of statistical physics, an *order parameter* can be defined using the probability that a polynomial-time algorithm will find the correct SCS. For a finite system size (in our case set by the number of reads N_{frag}) this probability will increase continuously as a function of the scaling variable (which plays the role of variables like the temperature, magnetic field or pressure in the phase diagrams of physical systems). As N_{frag} grows, the crossover regime grows narrower until, in the $N_{\text{frag}} \rightarrow \infty$ limit, one can speak of a *phase transition*: the computation always succeeds for $\langle x \rangle > x_c$ (and always fails for $\langle x \rangle < x_c$). In this sense, macroscopic physical systems are considered to be in a “thermodynamic limit,” whose behavior is indistinguishable from that of an infinite system. Similarly, while this study has considered small values of N_{frag} to penetrate into the hard regime and to show the scaling behavior, real instances of assembly employ such large N_{frag} that one can properly talk of two distinct phases in its computational complexity.

Provided that the just-mentioned interpretation of our numerical results proves to be correct, the behavior described above will be universal [28], in that the same scaling variable will classify instances of the problem into easy or hard, no matter which polynomial-time algorithm is used [33]. The precise location of the critical point x_c is algorithm dependent, but it will be such that the average maximum distance between reads is of the order of the length of the reads, $\langle d_{\text{max}} \rangle \sim \ell_{\text{frag}}$ (an intuitive result that we have demonstrated by studying two completely different methods). Putting all the above considerations together with the fact that, in modern high-throughput methods, the genome is heavily oversampled (implying $\langle d_{\text{max}} \rangle \ll \ell_{\text{frag}}$) we have our main result: practical instances of the sequence-assembly problem are deep in the easy phase, that is, always solvable in polynomial time. We would have thus achieved a characterisation of the parametric complexity of the problem in the sense proposed in Refs. [17,18].

The universality of our result should not be taken to mean that all algorithms are equally good or, more to the point, that sophisticated methods based on heuristics and de Bruijn graphs [11,12] are not useful. It does mean that, as pointed out recently in Ref. [34], their usefulness does not reside in their turning an NP-hard problem into a polynomial one. Instead, de Bruijn graphs are useful because of their efficient implementation for very large datasets and their power for dealing with errors in the reads and long repeats in the genomes.

Below x_c , the SCS problem decouples from that of sequence assembly (since the full genome cannot be reconstructed unambiguously) and becomes NP-hard. In this phase, we do not know the real SCS but we can consider a variational upper bound on its length given by ℓ_{ordered} . This bound will be good close to x_c and deviate more and more from the real solution to the SCS as $\langle x \rangle$ decreases. We have explored this regime using a Markov-chain Monte Carlo method that we name segment-swap. We find that, unlike deterministic methods, our segment-swap algorithm always succeeds in finding solutions with $\ell \leq \ell_{\text{ordered}}$ for $-1 \leq \langle x \rangle \leq 0.5$. As considered in this work, just as a proof of concept, the segment-swap method is not ergodic, so there is no assurance that its stopping point corresponds to the actual SCS. This shortcoming could be cured by coupling segment swaps with parallel tempering [32], which, furthermore, provides a self-consistent way of validating the solutions [35,36]. We thus believe that segment-swap Monte Carlo may be considered as a candidate to solve general instances of the SCS and related problems, such as the traveling salesman.

ACKNOWLEDGMENTS

This work was partially supported by Ministerio de Ciencia, Innovación y Universidades (Spain), Agencia Estatal de Investigación (AEI, Spain, 10.13039/501100011033), and European Regional Development Fund (ERDF, A way of making Europe) through Grant PID2022-136374NB-C21. D.Y. was supported by the Chan Zuckerberg Biohub.

APPENDIX A: OUR GREEDY *Glotón* ALGORITHM

Our greedy algorithm solves the SCS problem (very slightly) better than the standard greedy algorithm, of which it is only a slight variation (see, e.g., Ref. [37]). As explained in the main text, we represent the superstring as a permutation of the original reads [our state space is a sequence of reads which are consecutive in the permutation, see Fig. 5 (left)]. The total length of the superstring is $N_{\text{frag}}\ell_{\text{frag}}$ minus the total sum of the overlaps between consecutive reads (therefore, the SCS corresponds to the maximum total overlap between consecutive reads). Our *Glotón* seeks the shortest superstring through the following procedure:

(1) Pick at random the starting fragment of the cycle. This fragment is named the *active* read.

(2) Consider the overlap with the active read of all the still unsorted reads (the *candidates*).

(3) Select the candidate that has the maximum overlap with the active read. If there is more than one choice, we pick randomly (with uniform probability) one candidate of maximum overlap. The chosen candidate is placed in the cycle

right after the active read and is declared to be the *new* active read.

(4) While there are remaining candidates, go back to step 2.

There are two differences with the standard greedy algorithm. First, the randomness in step 3 (some implementations pick the maximum-overlap candidate deterministically) Second, we grow the sequence from just one active fragment (instead, the greedy algorithm allows more than one sequence-growing point).

The most demanding part of the *Glotion* algorithm is the computation of the overlap between fragments. We have sped up this part of the computation by generating a look-up table containing all possible overlaps—there are $N_{\text{frag}}(N_{\text{frag}} - 1)$ possible ordered pairs of reads. This is particularly useful, because we run the *Glotion* algorithm $N_{\text{attempts}} = 100$ times for each given set of N_{frag} reads.

APPENDIX B: A NOTE ON *Velvet*

Velvet is not properly an algorithm for finding the SCS, but instead outputs contigs. These are contiguous segments that can unambiguously be inferred to be part of the original genome. In this case, a successful solution produces a single contig of length ℓ_{ordered} while an ‘unsuccessful’ one might be missing one or more reads or be broken into several contigs. Notice that this is the program working as desired: it interprets that it does not have enough data to reconstruct the full genome and, rather than attempting to find an approximation to an SCS that would not match the original sequence, it produces unambiguous subsequences. Hence, the phase transition acquires a different meaning for *Velvet*: the critical point separates the region where *the fragments database comes from a single contig, for sure*, from the region when *Velvet* reaches the conclusion that *most probably, the database comes from two (or more) contigs*, (which is unjustified whenever $x > 0$).

-
- [1] D. Maier, The complexity of some problems on subsequences and supersequences, *J. ACM* **25**, 322 (1978).
 - [2] J. Gallant, D. Maier, and J. Astorer, On finding minimal length superstrings, *J. Comput. Syst. Sci.* **20**, 50 (1980).
 - [3] D. E. Knuth, Postscript about NP-hard problems, *ACM SIGACT News* **6**, 15 (1974).
 - [4] C. Papadimitriou, *Computational Complexity* (Addison-Wesley, Reading, MA, 1994).
 - [5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *An Introduction to Algorithms*, 3rd ed. (MIT Press, Cambridge, MA, 2009).
 - [6] P. A. Pevzner, H. Tang, and M. S. Waterman, An Eulerian path approach to DNA fragment assembly, *Proc. Natl. Acad. Sci. USA* **98**, 9748 (2001).
 - [7] P. Medvedev, K. Georgiou, G. Myers, and M. Brudno, Computability of models for sequence assembly, in *Proceedings of the International Workshop on Algorithms in Bioinformatics* (Springer, Berlin, 2007), pp. 289–301.
 - [8] E. W. Myers, The fragment assembly string graph, *Bioinformatics* **21**, ii79 (2005).
 - [9] M. Schatz, A. Delcher, and S. Salzberg, Assembly of large genomes using second-generation sequencing, *Genome Res.* **20**, 1165 (2010).
 - [10] P. E. C. Compeau, P. A. Pevzner, and G. Tesler, How to apply de Bruijn graphs to genome assembly, *Nat. Biotechnol.* **29**, 987 (2011).
 - [11] J. R. Miller, S. Koren, and G. Sutton, Assembly algorithms for next-generation sequencing data, *Genomics* **95**, 315 (2010).
 - [12] W. Zhang, J. Chen, Y. Yang, Y. Tang, J. Shang, and B. Shen, A practical comparison of de novo genome assembly software tools for next-generation sequencing technologies, *PLoS One* **6**, e17915 (2011).
 - [13] S. El-Metwally, T. Hamza, M. Zakaria, and M. Helmy, Next-generation sequence assembly: Four stages of data processing and computational challenges, *PLoS Comput. Biol.* **9**, e1003345 (2013).
 - [14] S. E. Levy and R. M. Myers, Advancements in next-generation sequencing, *Annu. Rev. Genom. Hum. Genet.* **17**, 95 (2016).
 - [15] D. L. Applegate, *The Traveling Salesman Problem: A Computational Study* (Princeton University Press, Princeton, NJ, 2006).
 - [16] W. Cook, Traveling salesman problem, <https://www.math.uwaterloo.ca/tsp/>.
 - [17] H. L. Bodlaender, R. G. Downey, M. R. Fellows, M. T. Hallett, and H. T. Wareham, Parameterized complexity analysis in computational biology, *Bioinformatics* **11**, 49 (1995).
 - [18] N. Nagarajan and M. Pop, Parametric complexity of sequence assembly: Theory and applications to next generation sequencing, *J. Comput. Biol.* **16**, 897 (2009).
 - [19] C. Kingsford, M. C. Schatz, and M. Pop, Assembly complexity of prokaryotic genomes using short reads, *BMC Bioinf.* **11**, 21 (2010).
 - [20] G. Bresler, M. Bresler, and D. Tse, Optimal assembly for high throughput shotgun sequencing, *BMC Bioinf.* **14**, S18 (2013).
 - [21] Y. Fu and P. W. Anderson, Application of statistical mechanics to NP-complete problems in combinatorial optimisation, *J. Phys. A* **19**, 1605 (1986).
 - [22] M. Mézard, G. Parisi, and M. Virasoro, *Spin-Glass Theory and Beyond* (World Scientific, Singapore, 1987).
 - [23] M. Mézard and A. Montanari, *Information, Physics, and Computation* (OUP Oxford, Oxford, UK, 2009).
 - [24] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky, Determining computational complexity from characteristic ‘‘phase transitions,’’ *Nature (London)* **400**, 133 (1999).
 - [25] E. S. Lander and M. S. Waterman, Genomic mapping by fingerprinting random clones: A mathematical analysis, *Genomics* **2**, 231 (1988).
 - [26] D. R. Zerbino and E. Birney, Velvet: algorithms for *de novo* short read assembly using de Bruijn graphs, *Genome Res.* **18**, 821 (2008).
 - [27] E. Schlemm, Limiting distribution of the maximal distance between random points on a circle: A moments approach, *Stat. Probab. Lett.* **92**, 132 (2014).

- [28] D. J. Amit and V. Martín-Mayor, *Field Theory, the Renormalization Group and Critical Phenomena*, 3rd ed. (World Scientific, Singapore, 2005).
- [29] When relating to the continuum results of Ref. [27], our formulation has discretization errors of order $1/L$. The other natural lengths occurring in the problem are d_{\max} and ℓ_{frag} , hence we expect discretization errors to be controlled by $d_{\max}/\ell_{\text{frag}}$ and d_{\max}/L . But $d_{\max}/\ell_{\text{frag}} = 1 - x$ and $d_{\max}/L \sim \log N_{\text{frag}}/(N_{\text{frag}}\ell_{\text{frag}})$, recall Eq. (5).
- [30] A. D. Sokal, Monte Carlo methods in statistical mechanics: Foundations and new algorithms, in *Functional Integration: Basics and Applications (1996 Cargèse School)*, edited by C. DeWitt-Morette, P. Cartier, and A. Folacci (Plenum, NY, 1997).
- [31] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, Optimization by simulated annealing, *Science* **220**, 671 (1983).
- [32] K. Hukushima and K. Nemoto, Exchange Monte Carlo method and application to spin glass simulations, *J. Phys. Soc. Jpn.* **65**, 1604 (1996).
- [33] In other words, the universality caused by the presence of a phase transition ensures that all polynomial algorithms will fail in the hard phase, i.e., that the problem is not in P in the usual classification of computational complexity.
- [34] P. Medvedev and M. Pop, What do Eulerian and Hamiltonian cycles have to do with genome assembly? *PLoS Comput. Biol.* **17**, e1008928 (2021).
- [35] R. Alvarez Baños, A. Cruz, L. A. Fernandez, J. M. Gil-Narvion, A. Gordillo-Guerrero, M. Guidetti, A. Maiorano, F. Mantovani, E. Marinari, V. Martín-Mayor, J. Monforte-Garcia, A. Muñoz Sudupe, D. Navarro, G. Parisi, S. Perez-Gaviro, J. J. Ruiz-Lorenzo, S. F. Schifano, B. Seoane, A. Tarancon, R. Tripiccione, and D. Yllanes (Janus Collaboration), Nature of the spin-glass phase at experimental length scales, *J. Stat. Mech.* (2010) P06026.
- [36] V. Martín-Mayor and I. Hen, Unraveling quantum annealers using classical hardness, *Sci. Rep.* **5**, 15324 (2015).
- [37] A. Goel, Shortest superstring problem, <https://www.geeksforgeeks.org/shortest-superstring-problem/>.