Towards Computation, Space, and Data Efficiency in de novo DNA Assembly: A Novel Algorithmic Framework

Ka Kit Lam and Nihar B. Shah
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
{kklam, nihar}@eecs.berkeley.edu

Abstract—We consider the problem of de novo DNA sequencing from shotgun data, wherein an underlying (unknown) DNA sequence is to be reconstructed from several short substrings of the sequence. We propose a de novo assembly algorithm which requires only the minimum amount of data and is efficient with respect to space and computation. We design the algorithm from an information-theoretic perspective of using minimum amount of data. The key idea to achieve space and computational efficiency is to break the procedure into two phases, an online and an offline phase. We remark that this can serve as an evidence of the feasibility of using an information-theoretic perspective to guide practical algorithmic design in DNA sequencing. Preliminary work on extending this algorithmic framework to more realistic settings is also reported.

I. INTRODUCTION

DNA sequencing is an important process that contributes greatly to the understanding and advancement of medicine and biology. The advent of ‘next-generation’ sequencing technologies has drastically reduced the costs and the time associated to DNA sequencing in the last ten years [2]. Under these next-generation technologies, reads (i.e., substrings) of the underlying DNA sequence are first obtained, from which the underlying sequence is then inferred computationally. Due to the massive amounts of data generated, there is a great need to address the computational aspects of the sequencing process [3], [4]. Thus, algorithm design for assembly of such reads for DNA sequencing remains an important algorithmic challenge.

In this report, we consider the problem of designing efficient algorithms for de novo DNA assembly from shot-gun data. We present a novel algorithm for de novo DNA sequencing that is efficient in terms of both (storage-)space and computations, and furthermore, minimizes the number of reads required for recovering the underlying DNA sequence.

The approach we take towards algorithm design is based on an information-theoretic perspective [5]. This model, which we shall refer to as the ‘i.i.d. model’, is a simple generative model for the DNA sequence and the reading process. This model, per se, is not a very accurate description of real DNA sequences, but provides good insights with respect to this problem. In particular, this model has previously been employed to analyse the fundamental requirements of a sequencing process (e.g., the number of reads required for correct reconstruction) and to analytically compare the performance various existing algorithms. In this paper, we first design an efficient sequencing algorithm based on this model, and subsequently use this as a building block towards de novo assembly in more realistic scenarios.

The contributions of this work thus are:

1) construction of a space, time and data efficient de novo sequencing algorithm based on the i.i.d. model,
2) construction of a framework towards building efficient de novo assembly algorithms for real data, and
3) providing evidence of the feasibility of a information-theoretic perspective to building practical algorithms for DNA sequencing (as opposed to using it for the purposes of analysis alone).

The key idea behind this algorithm, that helps it retain both space and computational efficiency, is to divide the task into two phases: an online phase and an offline phase. Current algorithms (e.g., [3], [6]–[9]) operate only under an offline phase, wherein they require all the reads at their disposal at the beginning of the execution of the algorithm. Such an approach leads to a considerably high storage space requirement since all the reads are required to be stored, and are then processed jointly. On the other hand, the online phase of our algorithm processes and combines reads on the fly, thereby offering a significant reduction in the storage space requirements. We provide additional algorithmic novelties to ensure that the two phases are executed in an efficient manner.

We note that while a lower computational complexity directly relates to lowering the sequencing time, a low space requirement can also contribute significantly to this cause. A lower storage requirement essentially enables the storage of all the data (i.e., all the necessary information required for assembly) in the faster but limited main memory. This results in a faster sequencing as compared to algorithms that have a larger storage footprint, and thus mandate either buying a greater amount of (expensive) main memory, or store and read part of the data from the (slower) disks.

The remainder of the report is organized as follows. Section II describes the i.i.d. generative model of [5], along with the fundamental bounds on the various parameters for exact recovery. Section III presents a space-and-computation-efficient and data-requirement-optimal assembly algorithm under the i.i.d. generative model. Section IV presents an analysis of
this algorithm. Section V provides preliminary thoughts on extending this framework to handle realistic data. Conclusions are drawn in Section VI.

II. THE BASIC GENERATIVE MODEL

A. Notation

Assume that the underlying sequence is haploid, and is $G$ bases long. Denote the sequence by a $G$-length vector $\vec{x}_G$, with each of its elements being $A$, $C$, $G$, or $T$. We have at our disposal, $N$ reads (substrings) of $\vec{x}_G$, each of length $L$, and the goal is to reconstruct $\vec{x}_G$ from these $N$ reads.

Throughout the paper, we shall use the terms underlying sequence and underlying genome interchangeably.

B. The i.i.d., Error-free Model [5]

The underlying DNA sequence $\vec{x}_G$ and the set of reads are assumed to be generated in the following manner. Each entry of the underlying sequence $\vec{x}_G$ is independently and identically distributed on the set of four alphabets $\{A, C, G, T\}$, with probabilities $\{p_1, p_2, p_3, p_4\}$ respectively. Here, $\min_{i=1}^{4} p_i \geq 0$ and $\sum_{i=1}^{4} p_i = 1$. The values of $\{p_1, p_2, p_3, p_4\}$ may or may not be known – our algorithm can handle both the cases. The $N$ reads are obtained by noiseless sampling from the long underlying sequence $\vec{x}_G$. The starting positions of the reads are unknown, and are assumed to be uniformly (and independently) distributed across the entire sequence $\vec{x}_G$ of length $G$. Moreover, for simplicity of exposition, we assume that the reads may also be wrapped around $\vec{x}_G$, i.e., if the starting position of a read lies in the last $(L-1)$ bases of $\vec{x}_G$, then the $L$ bases of the read consist of the last few bases of $\vec{x}_G$ followed by the first few bases of $\vec{x}_G$. Again, this condition of wrapping around is not fundamental to our algorithm, and only aids in the analysis. The reads are assumed to be free of errors or missing data. The setting is illustrated via an example in Fig. 1. The goal is to exactly recover the underlying sequence $\vec{x}_G$.

C. Fundamental Bounds under the i.i.d. Model [5]

Motahari et al. [5] previously showed that for exact recovery of $\vec{x}_G$, there exists a tradeoff between the number of reads $N$ and the length $L$ of each read. In particular, in an asymptotic setting, they derived a precise characterization of the values of $(N, L)$ under which exact reconstruction is feasible, as described below.

The scheme for assembly provided in [5] is a greedy one. Under this scheme, the maximum overlap between every pair of reads is computed first. Pairs of reads are then merged in the decreasing order of overlaps. For this algorithm to fail, there are two possible sources of errors:

(a) Lack of coverage: There exists one or more bases in $\vec{x}_G$ that are not covered by any of the $N$ reads.

(b) Confounding due to repeats: Two reads, which do not overlap in $\vec{x}_G$, coincidentally have an overlap that is larger than their overlaps with the reads that are their true neighbours.

For instance, for the sequence depicted in Fig. 1a, the set of reads $\{ACTGA, TGACA, CAGTG, AGTGG\}$ would cause an error due to lack of coverage since the last position is not covered by any of the reads. On the other hand, the reads $\{ACTGA, ACAGT, AGTGG, GAAGT\}$ would confound the algorithm due to the repeats: the algorithm would append $GAAGT$ at the end of $ACTGA$ (due to the overlap ‘GA’), while the correct merge would have been to append $ACAGT$ at the end of $ACTGA$ (they have an overlap of only ‘A’).

It is shown in [5] that under the greedy algorithm, when

\[ L \geq \frac{2}{H_2(p)} \ln G, \quad \text{and} \tag{1} \]

\[ N \geq \frac{G \ln G}{L}, \quad \text{and} \tag{2} \]

the probability of error decays exponentially with an increase in $G$. Here, $H_2(p)$ is the Renyi entropy of order 2 for the distribution $\{p_1, p_2, p_3, p_4\}$, and is given by

\[ H_2(p) = -\ln \sum_{i=1}^{4} p_i^2. \tag{3} \]
When conditions (1) and (2) are satisfied, the probability of error under the greedy algorithm goes to zero asymptotically as $G \to \infty$. It is also shown in [5] that the conditions (1) and (2) are necessary for any algorithm to be able to correctly reconstruct the sequence $\vec{x}$.

The greedy algorithm of [5] described above is highly suboptimal in terms of space and computation efficiency. In particular, since it stores all the reads, it requires a space of $\Theta(NL) = \Theta(G \ln^2 G)$. Furthermore, the algorithm needs to make pairwise overlap computation for each pair of reads, thus resulting in a computation complexity lower bounded by $\Theta(N^2L) = \Theta(G^2 \ln^2 G)$. In the next section, we describe our algorithm that is efficient in terms of storage space and computation requirements, and is also optimal with respect to the number of reads required.

III. MAIN ALGORITHM

The proposed algorithmic framework operates as follows. Depending on the (stochastic) model for the process, define a similarity metric between any two reads. Also find a threshold such that under the model considered, there is a vanishing probability of two non-adjacent reads having a similarity greater than that threshold. The first phase of the algorithm is an online phase, where two reads are merged whenever their similarity crosses the threshold. This phase thus allows the algorithm to require a small storage space, since all the reads now are not required to be stored separately. The second phase of the algorithm is an offline phase, where the remaining reads are merged in a greedy-yet efficient manner, exploiting the knowledge that no two of the remaining contigs gave a similarity greater than the threshold. Finally, depending on the underlying model of errors, run a third ‘consensus’ phase that performs a consensus operation to obtain a final result from the scaffold construction (this third phase is not required in the absence of errors, or if missing data is the only form of errors). We now make this framework concrete by applying it to an i.i.d. model in the absence of errors.

A. Parameters

Our algorithm requires the read lengths to satisfy

$$L \geq \frac{4}{H_2(p)} \ln G.$$  

(4)

For any $L$ satisfying this condition, our algorithm is optimal with respect to the number of reads required for successful exact reconstruction, i.e., operates with

$$N = \frac{G \ln G}{L}.$$  

(5)

Note that the value of $L$ required by our algorithm (4) is only a factor of 2 away from the lower bound (1). This may not be of a great concern in general, since for example, substituting $G = 10^{10}$ and assuming $p_i = 0.25 \ \forall \ i$, we get the requirement to be $L \geq 67$ which is certainly feasible with today’s sequencing technologies. On the other hand, the optimality of $N$ (see (2) and (5)) is perhaps a significant advantage offered by this algorithm since this allows the sequencing to be carried out much faster.

**Figure 3.** An example of MergedContig and K-mers data structures. Each row represents one distinct merged-contig/K-mer respectively. The entries of MergedContig are split into consecutive K-length segments represented via curly braces. These K-length segments are stored in the K-mers table in a sorted manner. Also depicted (through indices 1 to 10) is the one-to-one correspondence between the entries of the two tables, which are implemented via pointers both ways.

**Figure 4.** Implementation details of the MergedContig data structure. Each contig in the table is stored in the form of a linked list with each element having a pointer to the immediately succeeding length K sequence in that contig. Pointers to the beginning and end of each contig are stored separately. The set of contigs to an i.i.d. model in the absence of errors.

Associated to our algorithm, we define an additional parameter $K$ as

$$K = \frac{2}{H_2(p)} \ln G.$$  

(6)

This parameter shall be extensively employed in the algorithm.

1The parameter $K$ may also be chosen to have any higher value than 6, and this will continue to possess theoretical correctness guarantees provided in Section IV. The space and computation complexity of the algorithm remains the same (in an order sense) as long as $K$ is chosen to be $\Theta(\ln G)$. 
Algorithm 1 Main assembly algorithm

1) **Phase 1: online**

   Execute the following for every arrival of a new read \( r \) (of length \( L \)).
   
   a) Segment \( r \) into K-mers, i.e., all consecutive \( K \)-length subsequences (a total of \( L - K + 1 \) of them)
   
   b) Search each of these \((L - K + 1)\) subsequences in the K-mers table, and for each exact match, obtain the corresponding parent entry in “Merged-contig”. There will be at-most two such parent entries in “Merged-contig” (see Fig. 6).
   
   i) If there are no matching entries, then run subroutine Combine\((r', \text{null}, \text{null}, \text{null})\).
   
   ii) If there is exactly one matching entry \( y \) in “Merged-contig”, and if \( y \) completely subsumes \( r \), then do nothing. Such a matching is depicted in Fig. 6b.
   
   iii) If there is exactly one matching entry \( y \) in “Merged-contig”, and if this entry has a partial overlap with \( r \), then (see Fig. 6a)
   
      A) if prefix of \( r \) overlaps with suffix of \( y \), run subroutine Combine\((r', y, \text{new}, \text{existing})\).
      
      B) if suffix of \( r \) overlaps with prefix of \( y \), run subroutine Combine\((y, r', \text{existing}, \text{new})\).
   
   iv) If there are two matching entries, then \( r \) will have partial overlaps with both, as depicted in Fig. 6c. Suppose the suffix of \( r \) overlaps with the prefix of \( y_1 \) and the prefix of \( r \) overlaps with the suffix of \( y_2 \), then run subroutine Combine\((y_1, y_2, \text{existing}, \text{existing})\).

2) **Phase 2: offline**

   This phase merges all remaining contigs in MergedContig.
   
   a) Discard the K-mers table.
   
   b) Create two tables “pre-K-mer” and “suf-K-mer”, both implemented in the same way as “K-mer”, and populate them as follows (note that both these tables will be sorted).
   
   c) For each entry in MergedContig, store its K-length prefix as an entry in “pre-K-mer”.
   
   d) For each entry in MergedContig, store its reversed K-length suffix as an entry in “suf-K-mer”.
   
   e) If table “pre-K-mer” is empty, exit. Else, for the first entry \( y \) of “front-K-mer”, do the following:
   
      i) \( t_y \leftarrow y, s_1 \leftarrow 0, s_2 \leftarrow -1, t_x \leftarrow \text{null} \)
      
      ii) Execute subroutine Matching\((t_y, \text{suf}, s_2)\). If “no match found” then go to (2b iv). Else, let \( t_x \) denote this best match, and let \( s_1 \) denote the overlap between \( t_y \) and \( t_x \), and continue to (2b iii).
      
      iii) Execute subroutine Matching\((t_x, \text{pre}, s_1)\). If “no match found” then go to (2b iv). Else, let \( t_p \) denote this best match, and let \( s_2 \) denote the overlap between \( t_x \) and \( t_p \), and go back to (2b ii).
      
      iv) Merge the contigs corresponding to \( t_p \) and \( t_x \) in “MergedContigs”. Delete \( t_p \) from “pre-K-mer” and \( t_x \) from “suf-K-mer”. Go to (2e).
framework described previously, the distance metric is thus the amount of contiguous overlap and the threshold is $K$.) This phase thus allows the algorithm to require a small storage space, since all the reads now are not required to be stored separately. The comparisons and merges are made efficient by implementing the K-mers data structure as a red-black-tree, which enables fast searching, insertion and deletion. The offline phase merges the remaining set of contigs in a greedy-yet-efficient manner. This phase is made efficient by exploiting the fact that (following the online phase) no pair of contigs will have an overlap larger than $K$.

We also implement the following two subroutines that perform these tasks efficiently. Subroutine Combine() is used in the online phase for merging a new read with existing contigs. Subroutine Matching() is used in the offline phase for finding the best possible match (i.e., one with highest contiguous overlap) of a prefix contig among a set of suffix contigs, or vice versa. The precise algorithms of the two subroutines are provided in Algorithm 2 and Algorithm 3 respectively.

The operation of the online and offline phases of the main assembly algorithm are illustrated via an example in Figs. 7 and 8 respectively. These illustrations are associated to parameters $L = 4$ and $K = 2$, and assume an underlying sequence as $GCGTGGACCC$. The illustration of the online phase considers four newly arriving reads $GCGT$, $ACCC$, $ACCC$, and $GGAC$. The online phase merges these reads with existing contigs in a greedy manner, while the offline phase merges the remaining set of contigs in a greedy-yet-efficient manner.

Algorithm 2 Subroutine : Combine($t$, $u$, preType, sufType)
1) If preType=new and sufType=null then add $t$ as a new entry in table “Merged-contig”. Add the K-length prefix of $t$ and the K-length suffix of $t$ as two new entries in the K-mers table. Exit the subroutine.
2) If preType = existing, delete the K-mer corresponding to the prefix of $t$ from K-mers
3) If sufType = existing, delete the K-mer corresponding to the suffix of $u$ from K-mers
4) Merge the entries $t$ and $u$ in table “Merged-contig” as a single entry.
5) If a part of the merged entry is not included in K-mers (due to steps 2 and 3 of the subroutine, or if $t$ or $u$ is a new read), extract K-length substrings covering this part and add them as new entries into K-mers.

Algorithm 3 Subroutine : Matching($t$, searchTable, s)
1) If searchTable=suf:
a) For the $w = (\frac{L}{2} - 1)$ downto $(s + 1)$
i) Let $v$ be the first $w$ elements of $t$. Reverse $v$. Append $v$ with $(K - w)$ zeros.
ii) Search for $v$ in “suf-K-mer”.
iii) If there is an entry in “suf-K-mer” that matches the first $w$ elements of $v$, and that the parent (in MergedContig) of this entry is different from the parent of $t$, then return this entry and the value of $w$. Exit the subroutine.
b) Return “No match found”
2) If searchTable=pre:
a) For the $w = (\frac{L}{2} - 1)$ downto $(s + 1)$
i) Let $v$ be the last $w$ elements of $t$. Append $v$ with $(K - w)$ zeros.
ii) Search for $v$ in “pre-K-mer”.
iii) If there is an entry in “pre-K-mer” that matches the first $w$ elements of $v$, and that the parent (in MergedContig) of this entry is different from the parent of $t$, then
b) Return “No match found”
GTGG and GGAC (in that order).

IV. ANALYSIS

We analyse the performance of our algorithm in this section. Subsection IV-A describes simulations and experiments, while Subsections IV-B, IV-C and IV-D provide a theoretical analysis of the algorithm. To state the theoretical analysis in a nutshell, our algorithm is space-efficient in that it requires only \(O(G)\) main memory and computation-efficient as it requires \(O(G \ln^3 G)\) computations. The algorithm also minimizes the amount of data (i.e., the number of reads) required for successful reconstruction by achieving (5) which is (asymptotically) the minimum number of reads required for successful reconstruction of the sequence.

A. Simulation and Experiments

We have built a prototype of the assembler based on the algorithm provided, and the source code of the prototype has been made available. Using this prototype, we perform experiments to test the performance of the assembly algorithm. In particular, we consider an underlying DNA sequence of length \(G = 1000\), and for various values of the read length \(L\) and coverage \(NL\), we plot the fraction of times the algorithm successfully reconstructed the underlying sequence exactly. 2 We experiment with three different underlying sequences: (a) a synthetic sequence generated from an i.i.d. uniform distribution at each base, (b) a segment of human chromosome 22 and (c) a segment of a bacteria genome Enterobacteria phage. The results are plotted in Fig. IV-A.

We remark that under the i.i.d. model with each base drawn from a uniform distribution, our algorithm requires (in theory, asymptotically) a coverage of at least \(\frac{NL}{G} \geq 6.9078\) to entirely cover the underlying sequence and a read-length threshold of at least \(L \geq 4 \ln \frac{G}{\ln 2} = 39.8631\) for successful reconstruction. The empirical results of Fig. IV-A are fairly close: the genome is recovered correctly a reasonable fraction of times when \(L \geq 60\) and coverage is \(\geq 9x\). We reckon that the observed discrepancy is because the parameter \(G = 1000\), and hence the algorithm is operating in a regime that is far from hitting asymptotic limits. 3 One can also observe from the plots that, as one would expect, the performance of the algorithm improves with an increase in coverage or with an increase in the read lengths.

B. Theoretical Correctness Guarantees

One can see that the algorithm described above will give a correct solution whenever the following three conditions are satisfied:
- coverage: every position in \(\bar{x}_G\) should be covered at-least once
- no-duplication: in \(\bar{x}_G\), no sub-string of length \(K\) should occur more than once

2 For the simulations below, we employ the algorithm as in Algorithm 1. However, the implementations of the individual data-structures have not been fine tuned yet (e.g., the K-mers table is implemented as an array instead of the more efficient red-black-tree). This preserves all the correctness guarantees of the algorithm, but does not provide as efficient a performance in terms of space and computation.

3 We are also simulating the algorithm for much higher values of \(G\), but the simulations are still running at the time of submission.

Figure 7. An example of the online phase of the main assembly algorithm, with \(L = 4\) and \(K = 2\). The four parts (from top to bottom) show the updates to the MergedContig and the K-mers tables under the algorithm upon arrival of four new reads.

Figure 8. An example of the offline phase of the algorithm. The figures depict two merges in this phase: the subfigure above shows one merge and the subfigure below shows the immediate next merge. The tables depicted are the pre-K-mer and the suf-K-mer tables (on the left and right respectively). The MergedContig table of the offline phase is not depicted here. Highest overlap counterparts of the contig being matched are indicated by an arrow.
conditions (4) and (5) guarantee successful reconstruction of \( \tilde{x}_G \). In particular, in the asymptotic setting as \( G \to \infty \), the correctness is guaranteed with probability 1.

We note that our algorithm guarantees the correct output (with probability 1) whenever the three conditions listed above (coverage, no duplication, no confounding) are satisfied. This makes the algorithm robust to the assumptions on \( \tilde{x}_G \) as well as to the starting positions of the \( N \) reads, as long as these three conditions are satisfied.

C. Space Complexity

We first analyze the space complexity (i.e., the storage space requirements) of the algorithm. The data-structures “MergedConting” and K-mers encompass all the storage space requirements under our algorithm.

We begin with an analysis of the MergedConting data structure, which is employed in both the online and offline phases. As we saw previously in Section IV-B, since \( K \geq \frac{2}{\ln 2} \ln G \), all merge operations performed in the online phase are correct (with a high probability). Furthermore, observe that since \( L = 2K \), any position in the underlying sequence \( \tilde{x}_G \) can be a part of at most two contigs in MergedConting (otherwise two of the contigs in MergedConting containing this position would have had an overlap of at least \( K \), and would have been merged previously with each other). It follows that in MergedConting, every position appears at most twice, and hence the total size of the data stored is no more than \( 2G \) bases. This requires a storage space of at most \( 4G \) bits. Furthermore, it follows from a similar argument that the total number of contigs stored in MergedConting is at most \( \frac{2G}{L} \). We implement MergedConting as a linked list, and the overhead caused by the pointers employed is no more than \( \frac{2G}{L} \ln \frac{G}{2} \). For each entry of MergedConting, we also employ pointers to keep track of its prefix and suffix that are stored in K-mers, and this consumes a space of at most \( \frac{2G}{L} \ln \frac{G}{2} \ln \frac{G}{2} \). From (4) we have \( L \geq \frac{4}{\ln 2} \ln G \) and hence the total space required to store these pointers is upper bounded by \( \frac{1}{2} \ln \frac{G}{2} \ln \frac{G}{2} \). Summing these items up, memory of size no larger than \( (4 + 3H_2(p))G \) bits is required for MergedConting.

Next we analyse the space requirements of the K-mers data-structure. The analysis encompasses all three tables using this data-structure, i.e., K-mers, pre-K-mers and suf-K-mers. Since K-mers is allowed to be deleted at the end of the online phase, the storage requirements for these tables is max(storage required by K-mers table, sum of storage required by pre-K-mer and suf-K-mer tables). Thus, each of the parameters (e.g., storage requirement, number of entries) that are analysed below actually correspond to the max(parameter under K-mers table, sum of parameter under pre-K-mer and suf-K-mer tables). The aggregate length of all the entries in the table is upper bounded by \( 4G \), and hence storing this data requires a space no more than \( 8G \) bits. We also need pointers referring the entries of K-mers back to their respective parents in the MergedConting table. The number of entries in this table is at most \( \frac{4G}{K} \), while the number of entries in MergedConting is at most \( \frac{2G}{K} = \frac{4G}{2K} \). These thus require a space of at most \( \frac{4G}{2K} \ln \frac{G}{2} \) bits. We implement the K-mers data-structure as a self-balanced binary tree, and since the K-mers contains at
most $\frac{4G}{K}$ entries, the pointers for this implementation require a space of at most $\frac{4G}{K \ln 2}$ bits. Summing up these quantities, and substituting the value of $K$ from (6), we see that a memory of size no more than $(8 + \frac{4H_2(p)}{\ln 2}) G$ bits is required for the K-mers data-structure.

Thus the total space requirements for the algorithm is upper bounded by $(11 + \frac{5.5H_2(p)}{\ln 2}) G$ bits.

D. Computational Complexity

We now analyze the computational complexity of the algorithm. We start with an analysis of the complexity of the online phase. The online phase requires processing of $N = \frac{G \ln G}{L}$ reads. For each read, we obtain $O(L)$ number of K-mers. For each of these K-mers, an exact match is searched for in the K-mers table. Since this table is implemented as a red-black-tree, each search takes a worst case of $O(\frac{G}{K} \ln G)$ instances of matching two length $K$ strings. Thus the search complexity is $O(K \ln \frac{G}{K})$ for each K-mer. Furthermore, one needs to update the MergedContig and the K-mer tables, which also takes at most $O(K\ln G)$ computations. Aggregating these quantities and substituting the value of $K$ from (6), we obtain that the offline phase requires an aggregate computation upper bounded by $O(G \ln^3 G)$.

Next we analyse the computational-complexity of the offline phase. To this end, note that the total number of entries in MergedContig can be at most $O(\frac{G}{K})$, and hence the same also applies to the pre-K-mer and the suf-K-mer tables. Finding the best match in suf-K-mer for an entry of pre-K-mer takes $O(K \ln \frac{G}{K})$. Since there are $O(\frac{G}{K})$ entries in MergedContig, the subroutine of finding the best match is run a total of $O(K \frac{G}{K})$ times. Aggregating these quantities and substituting the value of $K$ from (6), we obtain that the offline phase requires an aggregate computation upper bounded by $O(G \ln^3 G)$.

Thus the overall computational complexity is $O(G \ln^3 G)$.

V. EXTENSIONS TO REALISTIC DATA

The algorithm presented here is based on the simplistic i.i.d. model, and some preliminary results on its extensions towards handling long repeats and errors are provided. Our immediate future goal is to build on this work and develop algorithms that can perform space, time and data efficient sequencing on real data. The two main departures of the i.i.d. model considered here, from the realistic data are that (a) the reads are assumed to be error free here, while in practice they may have indel or substitution errors or missing data, and (b) the underlying sequence may have much longer repeats that what is predicted by the i.i.d. model. The following two subsections describe preliminary thoughts on handling these issues. Also provided are bounds on the system parameters in the presence of substitution errors in the reads.

A. Handling Repeats

A significant difference between the i.i.d. model and real data is the presence of long repeats in real DNA sequences, which is a very low probability event under the i.i.d. model. In this section, we describe our preliminary work on extending the algorithmic framework described in Section III to handle the case of having long repeats. In a nutshell, the framework of Section III can detect repeats in the online phase. This enables one to avoid any confounding due to repeats in the online phase, and the greedy-yet-efficient algorithm of the offline phase can then handle the repeats. Details follow.

When the underlying sequence has multiple repeats of length $K$ or more, the algorithm of Section III is likely to get confounded, and produce an incorrect output. The following toy example illustrates the kinds of errors that will result in such a situation. Let us assume $G = 34$ and $L = 8$, which gives $K = 4$. Suppose the underlying sequence is $TTTTAAAAACCCCCAAAAAGGGGGAAAAATCGA$. Then, there is a chance that the algorithm will (incorrectly) output $TTTTAAAAAGGGGGAAAAACCCCCAAAAATCGA$. This sequence has three repeats $AAAA$ of length greater than $K = 4$, due to which the algorithm may interchange the data $GGGG$ and $CCCCC$ between two of the long repeats.

The framework developed in Section III inherently has the ability to detect long repeats. To see this, let us first consider an instance of running the algorithm on the toy sequence considered above. Let us assume that the first read is $TTTTAAAA$. The online algorithm will insert this read as the first entry in MergedContig table, and insert the extracted K-mers $AAAA$ and $TTTT$ in the K-mers table. Now suppose the next read is $AAAAATCGA$. Then, the K-mer $AAAA$ of this new read will match the $AAAA$ K-mer from the K-mer table. Since the online algorithm detects a length K repeat, it will merge these two reads, and replace the entry of the MergedContig table with $TTTTAAAAATCGA$ (and the K-mers table will now have $AAAA$, $TCGA$ and $TTTT$). This is what will subsequently lead to an erroneous output. Let us now see a modification of the algorithm to handle such cases. Suppose the algorithm subsequently observes the read $AAAAGGGG$. The K-mer $AAAA$ from the new read will match the K-mer $AAAA$ in the MergedContig. Thus, the algorithm would try to merge the two parent contigs $AAAAGGGG$ and $TTTTAAAAATCGA$ (treating $AAAA$ as a common sub-string), and would have failed in this process. However, observe that in this process, the algorithm does manage to detect the presence of a repeating $AAAA$, and we shall now exploit this ability of detecting long repeats.

Upon detection of any long repeat ($AAAA$ in this case), the algorithm now separates out the strings in the MergedContig table that contain this repeat as follows. The merged contig containing this repeat is split into two strings, one of which contains the prefix of the repeat followed by the repeat, and the other contains the repeat followed by its suffix. E.g., the string $TTTTAAAAATCGA$ in this case would be split into $TTTTAAAA$ and $AAAAATCGA$. The MergedContig and the K-mers tables are now updated with these two new strings, and the older string $TTTTAAAAATCGA$ is removed. Also, the new read $AAAAGGGG$ that triggered this is also not merged, and is stored as a separate string in MergedContig. At any subsequent time, a new read involving $AAAA$ has the K-
mer AAAA matching with multiple entries in MergedConting, and even in this case, a multiple repeat is safely detected. These contigs are subsequently merged in the offline phase in a greedy manner.

We note that these are preliminary ideas, and have not yet been implemented or thoroughly analysed.

B. Handling Errors

As mentioned previously, in general, our framework operates as follows. Depending on the (stochastic) error model, define a similarity metric between any two reads. Also find a threshold such that under the model considered, there is a vanishing probability of two non-adjacent reads having a similarity greater than that threshold. Now in the online phase, merge two reads whenever their similarity crosses the threshold. (In the algorithm of Section III, the distance metric is the amount of contiguous overlap and the threshold is K.) In the offline phase, merge the reads in a greedy-yet-efficient manner, exploiting the knowledge that no two of the remaining contigs gave a similarity greater than the threshold. Finally, run a third phase that performs a consensus operation to obtain a final result from the scaffold construction (this third phase is not required in the absence of errors, or if missing data is the only form of errors). While we leave the general case for future work, below we present (fairly tight) bounds on the system parameters for exact reconstruction in the presence of substitution errors in the reads.

Let us consider the case when the reads might have random substitution errors. As in Section II-B, we assume that the underlying sequence $\bar{x}_G$ is generated with each base drawn randomly in an i.i.d. fashion. For simplicity, we assume for now that the distribution for each base is uniform over \{A, C, G, T\}. Each read is an $L$ length contiguous substring of $\bar{x}_G$, and is drawn uniformly at random from the entire sequence $\bar{x}_G$. We introduce errors into the model in the following manner. For each read, we assume that each base is randomly and independently substituted by a different base. In particular, we associate a new parameter $p$ to the model, and assume that each base flips to any one of the three other bases with a probability $\frac{p}{3}$ each.

Our goal is to find necessary and sufficient conditions on the number of reads $N$ required to allow for exact recovery of $\bar{x}_G$ with a high probability. More formally, if $\bar{x}_G$ is the actual underlying sequence, and $\bar{x}_G'$ is the sequence reconstructed, we wish to have

$$\lim_{G \to \infty} P(\bar{x}_G \neq \bar{x}_G') = 0.$$

We obtain a lower bound and an upper bound on minimum data requirement $N^*$ for exact recovery in an asymptotic setting as follows. Let

$$f(p) = \frac{2p(9/4 - p)}{(p - 3/4)^2},$$

and

$$g(p) = \frac{1}{1 - exp(-\frac{1}{f(p)})}.$$

Then,

$$\max \{1, f(p)\} \cdot \frac{G \log G}{L} < N^* < \max \{1, g(p)\} \cdot \frac{G \log G}{L} \quad (7)$$

The proof of this result, along with a more detailed discussion on it, is provided in the Appendix. A plot of the bounds is provided in Fig. 10.

Preliminary thoughts on incorporating substitution errors in the algorithmic framework described in Section III are as follows. We could choose a higher value of the parameter $K$, and in the online phase merge reads with an overlap of $K$ but allowing for a small fraction of differences between the reads. Likewise in the offline phase, the merge of contigs can be performed allowing for a small fraction of errors. The positions of misconsensus are marked, and a consensus is enforced at the end of the assembly algorithm.

VI. Conclusions and Future Work

In this report, we presented a new algorithm for de novo DNA sequencing that is efficient with respect to the storage space and computation requirements, and is optimal with respect to the number of reads required for reconstruction. The algorithm is based on an i.i.d. generative model for the underlying DNA sequence. Our assembly algorithm operates in two phases, an online and an offline phase. The online phase merges reads with high overlaps on the fly, while the subsequent offline phase merges the remaining reads greedily. The operations in both phases are designed to be efficient in terms of space and time. In particular, by combining reads at run-time, the online phase significantly reduces the storage space requirements, which can also be exploited to achieve a much faster assembly by storing a greater proportion of the data in the fast-but-expensive main memory. This also serves as an algorithmic framework for designing algorithms in the future addressing various other scenarios and models, and preliminary thoughts on the same are presented in the paper. Furthermore, this also provides evidence of the feasibility of using the information-theoretic perspective for constructing practical algorithms for DNA sequencing (and not merely for analysis and comparison of different algorithms).

A additional (slightly theoretical) goal is to develop models that are closer to empirically observed data. We would then
like to compute the fundamental bounds on various parameters under these models, and also use them to analytically compare the performance of various algorithms. Extending the algorithmic framework towards such models will certainly be a high priority task.

REFERENCES


APPENDIX

BOUNDS ON THE DATA REQUIREMENTS IN PRESENCE OF SUBSTITUTION ERRORS

A. Previous work

The authors in [5] find the conditions for having the correct alignment on reads. We extend their results by considering a more stringent metric, exact recovery. We also provide an explicit coefficient factor on the coverage requirement for this more stringent metric.

In [5], authors consider a greedy algorithm. We reproduce this algorithm in Algorithm 4, along with an additional consensus requirement since our aim is to achieve exact recovery of the underlying sequence.

Let $\alpha^*$ be the value of $\alpha$ that satisfies the following equation

$$D(\alpha, 1 - \alpha)||\frac{3}{4}, \frac{1}{4}) = 2 \cdot D(\alpha, 1 - \alpha)||\frac{2}{p - 4 \cdot p^2, 1 - 2 \cdot p - 4 \cdot 3}{3}$$

In order to correctly align the reads, one can use the following overlap score metric along with the greedy algorithm (Algorithm 4) to correctly align the reads. Let function $d(\cdot, \cdot)$ denote the Hamming distance between its two arguments. We define the score metric between two reads $(\vec{r}_i^t, \vec{r}_j)$ to be the maximum overlap segment length that disagrees by at most a factor $\alpha$ of the overlapping segment. More formally, denoting the score metric by $s(\cdot, \cdot)$:

$$s(\vec{r}_i^t, \vec{r}_j) = \max \{ t \mid d(\vec{r}_i^t[L - t : L], \vec{r}_j[1 : t]) \leq \alpha t \}$$

where we use the notation $\vec{r}_i^t[a : b]$ to denote a segment of vector $\vec{r}$ spanning position $a$ to position $b$ of $\vec{r}$ (including the end points). The following lemma presents the precise asymptotic performance guarantees of Algorithm 4.

**Algorithm 4 Greedy algorithm with consensus**

**procedure** (GREEDYALGORITHM(reads))

- Compute overlap score between all pairs of noisy reads by $s(\vec{r}_i^t, \vec{r}_j)$
- Initialize contig list as reads
- for $w = L$ down to 0 do
  - for all $(\vec{r}_i^t, \vec{r}_j)$ do
    - if $(\vec{r}_i^t, \vec{r}_j)$ do not belong to the same contig and $s(\vec{r}_i^t, \vec{r}_j) = w$ then
      - Declare $(\vec{r}_i^t, \vec{r}_j)$ as consecutive read and declare them to belong to the same contig
    - end if
  - end for
- end for

**end procedure**

Align consecutive reads together to form a long string $\vec{x}_G$ which is the recovered sequence

Do a majority vote from the reads at each location in the $\vec{x}_G$ to form correctedGenome return correctedGenome

**end procedure**

**Lemma 1 ([5]):** When $L > 2/D([\alpha^*, 1 - \alpha^*])||[\frac{3}{4}, \frac{1}{4})] \cdot \ln G$ and $N > \frac{\ln G}{\ln L}$, the greedy algorithm (Algorithm 4) returns the correct alignment with high probability as $G \to \infty$.

B. Main theorem

**Proposition 2:** If $p < \frac{3}{4}$ and $L > 2/D([\alpha^*, 1 - \alpha^*])||[\frac{3}{4}, \frac{1}{4})] \cdot \ln G$, we obtain a lower bound and an upper bound on the data requirement for exact recovery as follows. Let $f(p) = \frac{2p(9/8 - p)}{(p - 3/4)^2}$.

- If $N < \min \{ 1, f(p) \} \cdot \frac{\ln G}{L}$, then $\exists \alpha > 0$ such that $\mathcal{P}(\vec{x}_G \neq \vec{x}_G^*) \geq \alpha > 0$
- If $N > \max \{ 1, \frac{1}{1 - \exp(-\frac{\ln G}{\ln L})} \} \cdot \frac{\ln G}{L}$, then Algorithm 4 achieves $\mathcal{P}(\vec{x}_G \neq \vec{x}_G^*) = 0$

That is the minimum data requirement on $N^*$ lies in the range of $\min \{ 1, f(p) \} \cdot \frac{\ln G}{L} < N^* < \max \{ 1, \frac{1}{1 - \exp(-\frac{\ln G}{\ln L})} \} \cdot \frac{\ln G}{L}$.

This expression is consistent with our intuition on coupon collector problem that the number of reads required should be multiple of $\frac{\ln G}{L}$. To have more intuition of the coefficient of growth of $\frac{\ln G}{L}$, we have plotted the lower and upper bounds in Fig. 10. Indeed, when $p = 0$, it reduces to the coverage constraint without noise, which is consistent of the Lander-Waterman coverage condition [11]. Thus this condition can be viewed as a generalization of Lander-Waterman coverage condition in the presence of noise.

C. Proof of Proposition 2

a) Precondition: Let $(n_1, ..., n_G)$ be the number of reads covering location $(1, ..., G)$ of the underlying sequence $\vec{x}_G$
respectively. Note that $\sum_{i=1}^{G} n_i = N \cdot L$. We first show a result Prop(3) that allows us to work on the asymptotic regime. Prop (3) means that the number of reads covering any base of the underlying sequence $x_G$ grows with $G$. This is a precondition for us to later use the normal approximation. The precise statement of this precondition is in Prop (3).

**Proposition 3:** We have, $\forall \epsilon > 0$, if $N > \frac{G \ln G}{L} (1 + \epsilon')$, 
\[
\lim_{G \to \infty} P(\exists i \text{ such that } , n_i < \ln \ln G) = 0
\]

**Proof:**
\[
P(\exists i \text{ such that } , n_i < \ln \ln G) 
\leq G \cdot P(n_1 < \ln \ln G) 
\leq G \cdot \sum_{i=0}^{\ln \ln G} \left( \frac{N}{i} \right) \left(1 - \frac{L}{G}\right)^{N-i} 
\leq G \cdot \sum_{i=0}^{\ln \ln G} \frac{1}{i!} N(1 - i/(1 - \frac{L}{G}))^{N-i} 
\leq G \cdot \sum_{i=0}^{\ln \ln G} \frac{1}{i!} \left( e^{-\frac{L}{G}} \right)^i (1 - \frac{L}{G})^{N-i} 
\leq G \cdot \left( 1 - \frac{L}{G} \right)^{-\ln \ln G} \cdot \sum_{i=0}^{\ln \ln G} \frac{1}{i!} \left( e^{-\frac{L}{G}} \right)^i (1 - \frac{L}{G})^{N-i}
\]

If $N > \frac{G \ln G}{L}$ is satisfied but $N < f(p) \cdot \frac{G \ln G}{L}$, we want to show that $\exists \alpha > 0$ such that $P_2 \geq \alpha > 0$ as $G \to \infty$. This, in turn, suggests $P(\text{error}) \geq \alpha > 0$. (Remark: the precondition ensures that $P_3 \to 1$).

Now, let us proceed to establish the lower bound on $P_2$ by considering an upper bound on $1 - P_2$. Here we condition on the event that $\{ \forall i, n_i > \ln \ln G \}$. For any vector $\vec{v}$, we shall denote its $i^{th}$ element as $v(i)$.

\[
1 - P_2
\leq \mathbf{E}_{n^G_{1}}[\mathbf{P}(\mathbf{v}_G(i) = x_G(i) \text{ } \forall i) \mid n^G_1] 
\leq \mathbf{E}_{n^G_{1}}[\mathbf{P}(\mathbf{x}_G^{MAP}(i) = x_G(i)) \mid n^G_1] 
\leq \mathbf{E}_{n^G_{1}}[\mathbf{P}(\text{At location } i, \text{ reads having } \text{A} \text{ reads having } C \right) = A) \mid n^G_1]
\]

Now let us define random variables
\[
Y_j^{(i)} = \begin{cases} 
1 & \text{with probability } 1-p \vspace{1mm} \\
0 & \text{with probability } \frac{2p}{3} \vspace{1mm} \\
-1 & \text{with probability } \frac{p}{3}
\end{cases}
\]

We have $\mathbf{E}[Y_j^{(i)}] = 1 - \frac{4p}{3}$, $\text{Var}[Y_j^{(i)}] = 2p \cdot (1 - \frac{8p}{9})$. We proceed the bounding of (13), with notation $\mathbf{P}(x)$ being the cumulative distribution of standard normal distribution. $\forall \epsilon' > 0$, we have,

\[
1 - P_2 
\leq \mathbf{E}_{n^G_{1}}[\sum_{j=1}^{n} Y_j^{(i)} > 0) \mid n^G_1]
\]

Letting $G \to \infty$ in (12), we get the desired result. 

**b) Necessary condition:** We now establish the necessary condition on data requirement to have exact recovery of the target genome. That is, if $N < \max(1, f(p))$, then $P(\text{error}) > 0$ as $G \to \infty$.

Let us start by defining some notation. Let
\[
P_1 = P(\exists i \text{ such that } n_i = 0) 
\]
\[
P_2 = P(\exists i \text{ such that } x_G(i) \neq x_G(i) \mid \forall i, n_i > \ln \ln G) 
\]
\[
P_3 = P(\forall i, n_i > \ln \ln G)
\]

In terms of these quantities, the probability of error $P(\text{error}) \geq P_1 + P_2 + P_3$ 

If $N < \frac{G \ln G}{L}$, $P_1 \to 1$, thus $P(\text{error}) \to 1$, which is the Lander-Waterman condition on coverage.
c) Sufficient condition: For Algorithm 4, we have,
\[ P(\text{error}) \leq P(\text{misaligned}) + P(\text{mis-consensus | correctly aligned}) \]

From the results of [5], we know that, \( P(\text{misaligned}) \to 0 \) with the parameters given above. Thus it remains to bound the second term. We obtain the following bound, \( \forall \epsilon > 0 \), when \( G \) is sufficiently large, with \( \mathcal{E} \) as the precondition
\[ P(\text{misconsensus | correctly aligned}) \]
\[ \leq \mathbb{E}\left[ \sum_{i=1}^{G} P(\text{misconsensus at location } i | \mathcal{E}) \right] + \epsilon \]
\[ \leq 3\mathbb{E}\left[ \sum_{i=1}^{G} P(\vec{x}'_G(i) = C | \vec{x}_G(i) = A) | \mathcal{E} \right] + \epsilon \]
\[ \leq 3\mathbb{E}\left[ \sum_{i=1}^{G} \Phi\left( -\frac{n_i(1 - \frac{4}{3}p)}{\sqrt{2pm_i(1 - \frac{5}{6}p)}} \right) | \mathcal{E} \right] + \epsilon \]  
(16)
\[ \leq 3G \cdot \mathbb{E}_{n_i}\left\{ \frac{1}{2} \exp\left[ -n_i \cdot \frac{1}{f(p)} \right] \right\} + \epsilon \]  
(17)
\[ = \frac{3}{2}G \cdot \left[ 1 - \frac{L}{G} (1 - e^{-\frac{1}{\Pi(p)}}) \right] + \epsilon \]
\[ \leq \frac{3}{2} \exp\left[ \ln G - \frac{LN}{G} (1 - e^{-\frac{1}{\Pi(p)}}) \right] + \epsilon \]

Here, (16) follows from a normal approximation, (17) results from bounding the \( \Phi \) function. Thus, if the conditions are satisfied, \( P(\text{error}) \to 0 \) as \( G \to \infty \).