GapFiller: a Preprocessing Step for the De Novo Assembly Problem

NADALIN F (1, 2), POLICRITI A (1, 2), VEZZI F (1, 2)
(1) Dipartimento di Matematica e Informatica - Università di Udine
(2) Istituto di Genomica Applicata (IGA) - Udine

Motivation
One of the most studied problems of bioinformatics is the Assembly Problem (AP). Recently, as a consequence of the strong impact of Next Generation Sequencing (NGS), a flourish of new tools claiming to assemble the “short” reads produced by new instruments has appeared. These tools, in most cases, do not present algorithmically new ideas but they simply differ from one to another for the implemented collection of heuristics. All NGS assemblers claim the need for high coverages (higher than 30x). While this need is reasonable and easy to satisfy at relatively low cost, no clear connection between the coverage and the expected quality of the result is provided. Moreover, it is rather largely accepted that coverage alone cannot solve one of the most tricky problem with AP, namely the correct usage/placement of repeated reads belonging to the genome being assembled. The aim of our work is twofold: we propose a new tool (GapFiller) to be used at various stages of de-novo assembly, based on fixed-k-mer search and reads extension, together with a statistical analysis framework aiming at a better understanding of the limits and potential of our instrument-as well as, in general, of AP. GapFiller does not aim at directly assembling large and complex genomes. Its main purpose is, instead, to be an aid in the first steps of an assembly pipeline by filling the gap within the paired reads (and hence the name GapFiller). As shown by experimental results, our tool returns small-Sanger’s reads size-but highly accurate contigs. The statistical approach wants to support the results achieved by GapFiller and to provide a statistical background able to describe the limits of an assembly project.

Methods
We developed a software (GapFiller) that iteratively extends a sequence (a read or a contig) using reads that overlap for at least k characters. In order to efficiently compute all the overlaps of length k all the reads are indexed accordingly to their prefix in an hash table H. Given the hash function f: A, C, G, T^k \rightarrow 0..q (with q being a large Mersenne's prime), we store in H all the reads accordingly to the value of their first k characters, in other words H[i] contains a pointer to all the reads such that f(r[1.. k])= i. In this way, given a sequence S, we have to compute f(S[|S|−k+1..|S|]) and f(rc(S[|S|−k+1..|S|])) (with rc a function returning the reverse complement of a string) in order to obtain all the possible reads that overlap with S. In theory k needs not to have a predefined limit, and we decided to fix it to half the read's length in order to compute reliable overlaps.
extension halts for the following reasons: (i) no more reads are found, (ii) a read already used during the extension is found and (iii) a repetition is identified. Point (iii) is the most important and delicate to deal with. A repetition represents a branching node in the assembly graph that we can imagine to have at the background of our procedure. Therefore, when we try to extend a repetition we have an high probability to make a mistake. In order to identify such situation we check if the reads being used to perform the extension phase form more than one cluster. Once a repetition is identified, instead of trying to resolve it, we opt for the more conservative approach of halting the extension. In the second part of our work we present a statistical framework in order to estimate the average number of steps done by GapFiller within a single extension. For this purpose, we define the random variable $X$ counting the number of steps executed during an extension. As initial approach, we assume that $X$ follows a geometric distribution, i.e. the probability of doing $j$ steps is $P(X = j) = p^j(1-p)$, where $p$ is the probability of doing one step (using the approach followed by Lander and Waterman in the Sanger sequencing context). The expected number of steps is then given by $E(X)$. More refined estimates are possible.

Results

GapFiller has been implemented in C++, with particular care on memory performances/requirements. The first tests on a 30x coverage (14Gbp) of grapevine genome consisting of 100bp reads, showed that only 30GB of RAM memory are necessary to store all the reads and perform the extension phase at reasonable speed. Every processed reads is extended 24 times, on average, and the mean contig’s length produced is 1,4Kbp. In order to test the accuracy of our output, we aligned the contigs against the reference sequence, obtaining that more than 94% of the assembled contigs perfectly align against the reference genome. The results obtained are used to test the adherence of the statistical model proposed. The probability $p$ of doing one step is defined in a simple way, depending on the frequency of repeated k-mers, and provides a lower bound to the average number of steps. A refinement of the model that takes into account the coverage, the reads’ length, as well as the distribution of the repeats, is currently under study.

Contact email
francesca.nadalin@uniud.it