Mapsembler, targeted assembly of larges genomes on a desktop computer

Pierre Peterlongo, Rayan Chikhi

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Mapsembler, targeted assembly of larges genomes on a desktop computer

Pierre Peterlongo — Rayan Chikhi

N° 7565

Mars 2011

_ Computational Biology and Bioinformatics _

ISSN 0249-6399 ISRN INRIA/RR--7565--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
RENNES - BRETAGNE ATLANTIQUE

Mapsembler, targeted assembly of larges genomes on a desktop computer

Pierre Peterlongo*, Rayan Chikhi

Theme : Computational Biology and Bioinformatics
Computational Sciences for Biology, Medicine and the Environment
Équipes-Projets Symbiose

Rapport de recherche n° 7565 — Mars 2011 — 14 pages

Abstract:

Background: The analysis of next-generation sequencing data from large genomes is a timely research topic. Sequencers are producing billions of short sequence fragments from newly sequenced organisms. Computational methods for reconstructing sequences (whole-genome assemblers) are typically employed to process such data. However, one of the main drawback of these methods is the high memory requirement.

Results: We present MAPSEMBLER, an iterative targeted assembler which processes large datasets of reads on commodity hardware. MAPSEMBLER checks for the presence of given regions of interest in the reads and reconstructs their neighborhood, either as a plain sequence (consensus) or as a graph (full sequence structure). We introduce new algorithms to retrieve homologues of a sequence from reads and construct an extension graph.

Conclusions: MAPSEMBLER is the first software that enables *de novo* discovery around a region of interest of gene homologues, SNPs, exon skipping as well as other structural events, directly from raw sequencing reads. Compared to traditional assembly software, memory requirement and execution time of MAPSEMBLER are considerably lower, as data indexing is localized. MAPSEMBLER can be used at http://mapsembler.genouest.org

Key-words: algorithms, genome, bioinformatics, NGS, targeted assembly, light memory usage

* Corresponding author

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

Mapsembler, assemblage ciblé de grands génomes sur un ordinateur de bureau

Résumé : L'analyse des données de séquençage nouvelle génération pour les grands génomes est un sujet de recherche en pleine expansion. Les séquenceurs produisent des milliards de fragments de séquences courtes à partir d'organismes nouvellement séquencés. Les techniques d'assemblage des génomes sont généralement utilisées pour traiter de telles données. Cependant, l'un des principaux inconvénients de ces techniques est leur importante demande en mémoire vive.

Nous présentons MAPSEMBLER, un assembleur itératif ciblé permettant de traiter de grands ensembles de données sur un ordinateur classique. MAPSEMBLER confirme la présence d'une région d'intérêt dans les données de séquençage et reconstruit son voisinage, soit sous la forme d'une séquence ordinaire (consensus) ou d'un graphe (structure complète). Nous présentons de nouveaux algorithmes pour retrouver les homologues d'une séquence et pour construire un graphe d'extensions. MAPSEMBLER est le premier logiciel permettant la découverte *de novo* du voisinage d'une région d'intérêt, par exemple des gènes homologues, SNPs, ou exon skipping, ou tout autre événement de nature structurale. Cette analyse s'effectue directement à partir des lectures de séquençage brutes. Par rapport aux logiciels d'assemblage traditionnels, les besoins en mémoire et temps d'exécution de MAPSEMBLER sont considérablement plus faibles, car l'indexation des données est localisée.

MAPSEMBLER peut être librement utilisé sur http://mapsembler.genouest.org.

Mots-clés : algorithms, génome, bio-informatique, NGS, assemblage ciblé, utilisation mémoire légère

Background

Genomics witnessed an unprecedentedly deep change a few years ago with the arrival of the Next Generation Sequencers (NGS). These technologies enable sequencing of biological material (DNA and RNA) at much higher throughput and a cost that is now affordable to most academic labs. Consequently, these new technologies generate datasets of size ranging from some gigabytes to over a petabyte. Datasets size is one of the two main bottlenecks for NGS. The other bottleneck is the analysis of generated data. Current technologies cannot output the entire sequence of a DNA molecule, instead they return small sequence fragments (*reads*) of length around a few hundreds base pairs. Without a reference genome, reconstructing the entire sequence from these fragments (*assembly*) is challenging, especially in terms of computational ressources. For instance, assembly of sequencing data from a mammalian genome requires hundreds of gigabytes of memory and several CPU weeks of computation [2]. Despite the ressource bottleneck, it should also be noted that assembly algorithms perform heuristics that lead to suboptimal reconstruction of the original sequence.

Since the cost of sequencing is becoming affordable to all, sequencing efforts are no longer limited to the main species of interest (human and other primates, mouse, rat, E. coli, yeast, Drosophila, ...). Thus, biologists are increasingly working on data for which they do not have any close reference genome. In such situations, reads assembly is often carried as a preliminary step. However, assembly is not always feasible, either because sequencing data is not be adequate (insufficient coverage, genome too complex or many genomes present) or computational ressources are lacking. Fortunately, many biological questions can be answered by analyzing unassembled reads. In particular, the user may possess *a priori* information on the data he is looking for. For instance, one may be interested in finding the homologue of a known gene in a newly sequenced set of reads, or a known enzyme in a metagenomic read set. The presented tool, MAPSEMBLER, enables to answer such questions.

The MAPSEMBLER software checks if a known piece of information called a *starter* (a sequence fragment) is present in a set of reads. In case of success it reconstructs its neighborhood, either as a plain sequence, or as a graph showing divergences and convergences in the neighborhood structure. One key aspect of MAPSEMBLER is that its memory usage is independent from the size of the reads set. This enables MAPSEMBLER to analyse huge sets of reads on a simple desktop computer.

Applications of MAPSEMBLER cover a broad range of biological questions, including but not limited to:

- Does this gene have an homologue in this set of reads? Using the gene as starter, MAPSEMBLER detects its presence and its homologous copies, and also reconstructs the genomic context for each copies.
- Does this enzyme exist in this metagenomic set? The DNA or RNA enzyme sequence is used as a starter, and MAPSEMBLER provides the information of its presence in the metagenomic read set.

- What are the copies of this sequence fragment? In case of highly repeated genomes, e.g. plant genomes, one may be interested in finding repeats of a sequence fragment. Using the sequence fragment as starter, all its occurrences (authorizing a genomic distance) are found.
- Detect and remove all reads corresponding to a known contaminant organelles, or symbiont.
- Understand what happens at the extremities of *de novo* assembled contigs. After *de novo* assembly of a read set, contigs (series of contiguous and overlapping reads) are created. A contig stops to be extended either because no more reads permits an extension, or several divergent possibilities are found. Using the extremity of a contig as a starter, MAPSEMBLER enables to understand why a contig was not extended further.
- For a known biological event, e.g. a SNP, or a splicing event, checks its presence in a set of reads, and provide abundances in each case by using a fragment localizing the event as starter.

Results and Discussion

The main result presented in this paper is a new software called MAPSEMBLER. Following this spirit, in this section we expose the features of MAPSEMBLER before presenting some applications on synthesized and real biological data.

Mapsembler features

MAPSEMBLER is a new simple yet powerful tool for targeting a specific piece of information from a possibly huge set raw reads generated by sequencers. MAPSEMBLER inputs are a sequence fragment or a set of fragments called the *starter(s)*, and a set of reads. For each input starter, the algorithm is separated into two main phases:

(1) **Mapping**. MAPSEMBLER checks if the starter could be reconstructed from reads. In this case the starter is said *read coherent*. During the reconstruction, a bounded number of substitutions is authorized between the reads and the starter. This does not include sequencing errors that are corrected on the fly.

(2) **De novo extensions**. Each read coherent starter is extended left and right. In accordance to the user choice:

   (a) either the extension is stopped as soon as several divergent extensions exist (apart from sequencing errors). In this case, the output is a simple fasta file containing the starter(s) and their reconstructed neighbors;

   (b) or an extension continues even in the case of several possible divergent extensions (apart from sequencing errors). In such a case the extensions could be represented in a tree. In practice, as two divergent extensions may lead to the same third extension, two nodes may

reach the same target third node, creating a directed graph instead of a tree.

Each node stores a sequence fragment and its average coverage. This graph, outputted in xgmml format, is useful for finding SNPs, or distinct splicing events for instance. Several tools, including Cytoscape [1], read xgmml data and can be used for results visualization.

MAPSEMBLER presents the advantage of not indexing reads but only starters (see Section *Methods* for algorithmic explanations). In practice, independently of the size of the reads file (even terabyte-sized), it is possible to run MAPSEMBLER on any desktop or laptop computer, not requiring large memory facilities.

The mapping phase (point (1)) performs several tasks. As mentioned, a limited number d of substitutions (Hamming distance) is authorized between a starter and coherent reads. As a consequence, for one starter s, several distinct subsets of reads may reconstruct s at distance at most d. See Fig. 1 for an example. Thus, with $d > 0$, several *sub-starters* may be generated and extensions (point (2)) are started from each sub-starter.

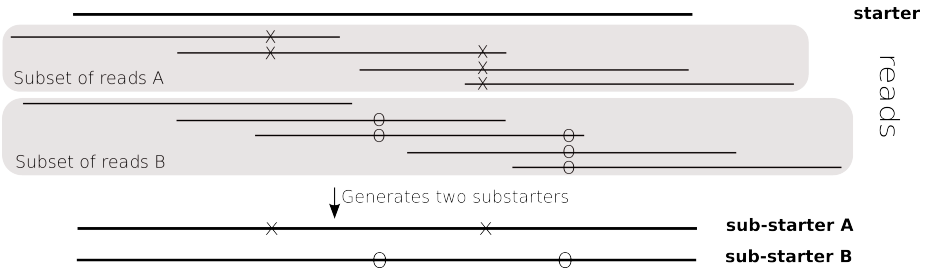

Figure 1: Representation of the sub-starter generation. a set of reads map a starter represented by the bold top line. An 'X' or a 'O' between a read and the starter represents a substitution, all other positions map perfectly. In this example two distinct subset of reads may confirm the starter with at most two substitutions. Two sub-starters are thus generated.

In practical terms, if a starter belongs for instance to a repeated region whose copies diverged, each distinguishable fragment of copy housing the starter at distance at most d will generate a sub-starter that will be extended further.

Mapsembler and the state of the art

To date, we are aware of only one software, TASR [4], that can be compared with MAPSEMBLER. TASR is based on the SSAKE assembler [5]. It maps a set of reads on targets using seeds of length 15: mapping between a read and a starter is tested if at least one word of length 15 exactly matches both. TASR outputs the result of this mapping, including extensions obtained from reads mapped to extremities of starters. Similarly to MAPSEMBLER, TASR indexes only starters and memory requirements does not depend on the size of reads file.

Note that MAPSEMBLER significantly differs from TASR as it offers the following novel features:

- sub-starters retrieval;
- multiple iterations to extend starters as far as possible. This is equivalent to re-running TASR multiple times, using its results as starters;
- graph output of the left and right neighborhood of starters.

We compared TASR and MAPSEMBLER performance, on a 2.66Ghz dual-core laptop with 3Mb cache, running with linux Fedora fc13. We used a set of 6.5 millions of short reads of length 36 (unpublished aphid RNA-seq). We ran MAPSEMBLER with an option avoiding iterative extensions and fixing seeds length to 15, simulating TASR behaviour.

Using a unique randomly picked read as starter, MAPSEMBLER was run in 40 seconds, using 2.15 Mb of memory, while TASR finished in 165 seconds using 4.21 Mb of memory.

On a bigger set of 500 starters randomly picked from reads MAPSEMBLER was run in 59 seconds, using 23.8 Mb of memory. TASR was stopped after 10 hours, while using 287 Mb of memory.

Note that in both cases, MAPSEMBLER produces strictly more results than TASR as it outputs sub-starters. Since both TASR and MAPSEMBLER algorithms extend starters in a similar fashion (see the Methods section), sequences returned by TASR are similar to MAPSEMBLER results, but not identical, due to different error-correcting heuristics and cutoff thresholds.

Application to simulated sequences

This section presents results obtained on simulated sequences. We performed mainly two kinds of experiments: one testing practical time and memory requirements on large datasets, and one testing the graph output, focusing on exons skipping. All random sequences are generated on a Bernouilli model, each nucleotide having probability 1/4 to occur at each position.

Results on large datasets on a desktop computer

We downloaded human chromosome 1, release hg19. Using Metasim [3], we simulated sets of 10k, 100k, 1M and 10M Illumina unpaired reads of length 75 nt. We picked 100 random reads as starters.

Results, summarized in Table 1 show that memory usage is very slightly impacted by the linearly increasing amount of reads and remains low: less than 10 Mb with 0.7 billion of read bases. While using the graph option, memory and time requirements grow when the graph becomes large as this is the case while using 10 million reads. Afters 126 iterations, the graph contains 17144 nodes. Each node stores several information, consuming memory. However, the maximal memory usage was 275.26 Mb for analysing 10 millions reads, making MAPSEMBLER usable on a desktop computer.

The maximal number of iteration is difficult to predict as it depends on several input parameters and on the structure of the biological data being sequenced. Thus, an option enables to limit the number of iterations. However,

Table 1: MAPSEMBLER time and memory requirements

Number of reads	Total read bases	Graph output	Number of iterations	Time	Memory (Mb)
10.000	750K	no	1	<1s	1.46
		yes	1	<1s	1.63
100.000	7.5M	no	1	1s	1.45
		yes	1	2s	1.63
1.000.000	75M	no	4	35s	1.71
		yes	22	3m23s	1.73
10.000.000	0.7G	no	10	15m58	2.09
		yes	126	3h33m02s	275.56

the iteration time is reasonable. For instance, even asking for a graph output and with data as big as 0.7Gb of reads for 100 starters, an average of a hundred of seconds is used per iteration.

One of the immediate purposes of graph representation is visual inspection by the user. In such use case, the number of starters is typically kept low, in order to target specific interesting areas. We illustrate this feature in the upcoming sections.

Retrieving novel exon skipping using graph visualisation

In this section, we present a typical application of MAPSEMBLER for visualizing events on a targeted area of a transcriptome. We simulated the presence of an exon skipping in a data set. We randomly generated two copies of a RNA sequence of length 10k. In the second copy, we added a synthetic "exon" of length 100 bases. We simulated the sequencing of the two sequences together using Metasim, yielding a single reads set. Using a fragment of the input sequence as starter, the exon was retrieved by MAPSEMBLER, as shown Fig. 2. On this figure, one may notice that the node containing the exon is twice less covered than other nodes as reads covering the exon are present in only one of the two initial merged read sets.

Application on biological sequences

Exon skipping in drosophila

On Drosophila Melanogaster, one Illumina HiSeq2000 RNA-Seq run of 22.5 million reads of length 70 nucleotides was performed (data not published). We chose a starter located close to a known exon skipping even (Chr4:488,592-488,619 BDGP R5/dm3). Using less than one megabyte of memory and 33 minutes, MAPSEMBLER found the result presented on Fig. 3, enabling to confirm the presence of the exon skipping.

Visualising SNPs in drosophila

On the same read data set, we used a fragment of sequence (chrX:17,783,737-17,783,812 BDGP R5/dm3) for which we knew the presence of genes around. We

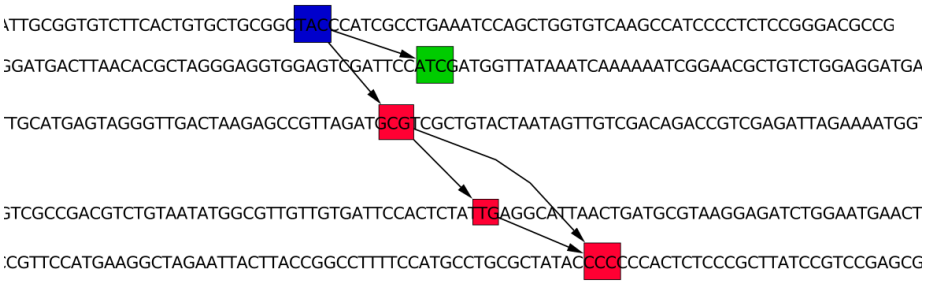

Figure 2: Example of the graph visualisation. Blue node is called the "root", as it contains only the starter. Green nodes are right extensions, while red ones are left extensions. Size of a node depends on the average coverage of the sequence it houses. In this example the left extensions is separated into two paths. One of the two path (lowest one) contains an exon artificially inserted. The two paths join back after the exon. Full sequences are truncated.

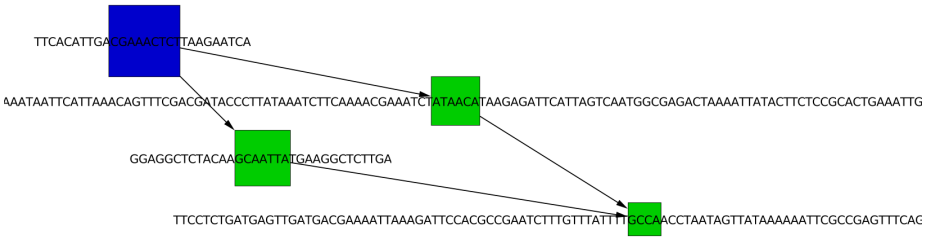

Figure 3: Visualisation of MAPSEMBLER results on real datasets, looking for known exon. The unique left node is not displayed. The exon can be seen as two distinct passes can be followed from starter. Full sequences are truncated.

applied MAPSEMBLER using this fragment as starter and obtained in less than 1 megabyte of memory and less than 110 minutes of execution (40 iterations). The results presented Fig. 4 enable to visualize the SNPs.

Methods

As presented in section "Mapsembler features", MAPSEMBLER is separated into two main phases: read mapping and sub-starters generation, and extension. We start by describing an overview of these phases. Then we present into details two specific points that deserve deeper explanations: the sub-starter generation and the graph management. Other algorithms, being classically used into mappers and assemblers, are already well detailed in the literature.

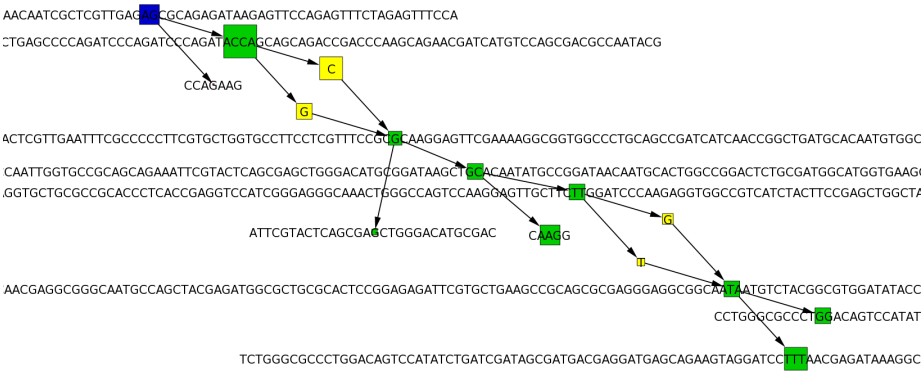

Figure 4: Visualisation of MAPSEMBLER results on real datasets, looking for known SNPs. On this graph, on can observe 2 SNPs (couples of yellow nodes) in the right extensions. Full sequences are truncated.

Methods, overview

In a few words, the first phase is very similar to a seed-based mapping algorithm (following points 1 and 2). Only the sub-starter generation presents an algorithmic novelty. The second phase (following points 4 to 7) is itself quite similar to the contig extension phase used in greedy de novo assembly algorithms [5]. The graph maintenance presents also a novelty presented in an upcoming section.

We now describe iteratively the MAPSEMBLER general approach:

First phase: mapping

1. An index of all words of length k (called k-mers) that appear in the starters is created. For each k-mer, a list of couples $(starter\_id, position\_on\_starter)$ is stored and can be accessed in constant time using a hash table. Additionally, all starters are stored in memory.

2. MAPSEMBLER sequentially processes input reads and their reverse complement, not storing any of them into memory. For each read r, its k-mers are extracted and used for anchoring r on starters. For a given starter s and anchored read r, a full mapping is done between s and r allowing a maximal amount of substitutions (user defined). If the mapping respects this constraints, r and its mapping position $p_{r,s}$ are stored in a hash table associated with s. Redundant computations are avoided by keeping track of previous successful or failed mappings.

3. Once all reads are read and possibly mapped, for each starter s:

   (a) Detection of its sub-starters $s_1, s_2, .., s_n$ (see Section "Sub-starter generation and read coherence step").

   (b) If at least one sub-starter was generated, s is said *read-coherent* and its sub-starters $s_1, s_2, .., s_n$ are conserved for next extensions. We call them *extensions* in the following.

Second Phase: assembly

4. Free the previous k-mer index. Index the *enriched extensions* as it is done in 1 for starters. By "enriched extension" we refer to the concatenation between the suffix of length $k-1$ of the source of the new extension and the new extension itself. Thus all possible k-mers including those overlapping two consecutive extensions are indexed.

   In the following we denote an enriched extension by "—∘extension".

5. Similarly to 2, reads are mapped to —∘extensions, not allowing any substitution. If a read r is mapped to an —∘extension e, the read and the anchoring position $p_{r,e}$ are stored in a hash table associated with e.

6. Once all reads are processed and possibly mapped to —∘extensions, for each —∘extension e: using positions information $p_{r,e}$, align and eventually correct all reads that mapped e and find the fragments consensus larger than —∘extension. Three possibilities arise:

   (a) There is no such fragment larger than e. The iterative extension of e is finished.

   (b) There is exactly one fragment consensus f larger than e. In this case the fragment f is stored and considered as a new possible extension for next iteration.

   (c) There are more than one fragment consensus $\{f_1, f_2, .., f_n\}$ larger than e. If a simple sequence output is required, the longest common prefix of all f_i is stored and the extension is stopped for e. If a graph output is required, n extending nodes are created, each storing a fragment f_i. All fragments in $\{f_1, f_2, .., f_n\}$ are stored and considered as extensions for next iteration.

7. If no new extension was discovered in 6b or 6c, stop the iterative process and output results. Else, go back to 4, using new extensions discovered in 6b and 6c.

Note that all extensions are always performed from left to right. For finding left extensions, the starters are simply reverse complemented once before the whole process and once after all to retrieve the original sequences.

Sub-starter generation and read coherence step

The sub-starter generation and read coherence step takes place at the end of the mapping phase. Given a fragment f and mapped reads R, this step generates a set of sub-starters $s_1, s_2..., s_n$ which:

- originate from the reads, e.g. each s_i is a consensus sequence of a subset of reads,
- are coherent with the fragment f, e.g. the Hamming distance between f and s_i is at most d.

We are interested in retrieving the largest set of sub-starters for each fragment f. If at least one sub-starter is found, the fragment f is said to be read coherent. Finding the largest set of sub-starters can be formulated as the following computational problem.

Problem (Multiple consensuses from reads alignment)**.** *Given a fragment f, a parameter $d \geq 0$ and a set of error-free reads $R = \{(r_i, p_i)\}$ (each read r_i is aligned to f at a position p_i), find all maximal (with respect to the inclusion order) subsets S_i of R satisfying:*

1. *each subset S_i admits a perfect consensus s_i, i.e. each read r_i aligns to s_i at position p_i (relative to f) with no mismatch,*
2. *the substring of s_i which is aligned with f has at most d mismatches with f,*
3. *each position of f is covered by at least one read in S_i.*

A trivial (exponential) solution is (i) to generate the power set (all possible subsets) of R, (ii) remove sets which do no satisfy one of the conditions, and (iii) keep only maximal sets (ordered by inclusion). The exponential complexity of this solution clearly comes from step (i). We give a polynomial time procedure which subsumes (i), as it generates a solution which includes all the correct subsets.

Algorithm 1 Generating candidate subsets S_i for solving the multiple consensuses from reads alignment problem

Require: Set of error-free reads R sorted in increasing order of alignment positions
Ensure: A set of candidate subsets S_i.

```
S = ∅.
for each read (r, p) in R do
  for each subset S_i in S do
    if r overlaps perfectly with the farthest read of S_i then
      Add r to S_i.
    else
      if r overlaps perfectly with one of the reads of S_i then
        Let (r', p') be the farthest read of S_i overlapping with r.
        Create a new subset with all reads of S_i up to position p' and r.
        Insert it into S, provided it is not already present.
  if r was not appended to any group then
    Create a new subset with r and insert it into S.
return S.
```

The completeness proof that this algorithm finds all correct sub-starters is as follows. For the sake of obtaining a contradiction, let s be a correct sub-starter not found by the algorithm. Let $r_1, .., r_n$ be the maximal subset of reads which yields s, sorted by increasing mapping positions to f. We show by induction that the algorithm returns a subset which includes $r_1, ..r_k$, for $k \in [1..n]$. For $k = 1$, we remark that the algorithm assigns a subset to each read. Assuming $r_1, ..r_k$ is part of a returned subset S_0, we show that $r_1, ..r_{k+1}$ is also returned by the algorithm. Since r_{k+1} is part of a subset which yields s, it overlaps perfectly with r_k. However, r_{k+1} does not necessarily belong to S_0. Let r'_{k+1} be the read which follows r_k in S_0. In the ordering of the reads by increasing position, if r_{k+1} is seen before r'_{k+1}, then the algorithm selects $r'_{k+1} = r_{k+1}$. Else, as r_{k+1} perfectly overlaps with r_k, the algorithm creates a new subset from S_0 which contains exactly $r_1, ..r_{k+1}$. Eventually, from the induction, the algorithm finds

a subset which contains $r_1, .., r_n$. Since $r_1, .., r_n$ is itself maximal, the subset found by the algorithm is exactly $r_1, .., r_n$.

The actual sub-starter generation in MAPSEMBLER differs from the above formulation bacause reads are not error-free. This issue is fixed in practice by a read correcting step, performed before subsets construction.

Graph management

Graph creation

A sequence fragment (an initial sub-starter or any extension) could be extended by more than one fragment consensus. In such a case, if the "graph" option is required, a tree structure is initially created. The root of the tree is labelled by the sub-starter and any internal node stores an $\multimap$extension. Reading all paths from the root to leaves provides all possible assemblies starting from the sub-starter.

However, several common biological events like SNP or exon skipping lead to the creation of two distinct paths, that meet back after the event and then continue with the same sequence. Thus, in order to avoid to duplicate computations over all similar extensions, the tree is transformed into a directed graph $\mathcal{G}$. When a node n_1 from $\mathcal{G}$ is extended with consensus fragment f, the algorithm checks if f was not already present in $\mathcal{G}$ in a node n_2. For doing this, the last k-mer of the sequence of each node of $\mathcal{G}$ is indexed in a hash table $\mathcal{G}_{references}$. Checking if f was already present in the tree is done by mapping f on nodes anchoring the mapping thanks to the k-mers presents on f and those stored in $\mathcal{G}_{references}$.

If mapping of f on the sequence of a node n_2 is correct, then n_1 is linked to n_2, keeping the graph coherency: managing prefixes and suffixes that could be longer in n_1 or n_2.

Graph simplification

After all extensions, each graph $\mathcal{G}$ is simplified as follows:

- $\multimap$extensions are transformed into extensions, by removing the first $k-1$ characters of each internal node except the root.
- Two nodes n_1 and n_2 are merged concatenating their sequences if and only if n_1 has only n_2 as child while n_2 has only n_1 as father.
- Common prefix of all children of a node n is pruned from these nodes and factorized as suffix of the sequence stored in n. Similarly common suffix of all fathers of a node n is pruned from fathers to be factorized as a prefix of n.

The first transformation removes redundant information in nodes. The second transformation is a classical compression of simple paths. The third transformation relocates branching in the graph, to the exact position where sequences diverge.

Conclusions

MAPSEMBLER is a simple yet powerful, non-specific tool for extracting targeted pieces of information from newly sequenced, non-assembled genomes or transcriptomes. Technically, MAPSEMBLER retrieves all the homologues of a region of interest and performs targeted assembly through repeated iterations of reads mapping. It also provides a visualization of the genomic context of assembled sequences as a graph. MAPSEMBLER can be executed on a classical desktop computer; without cleaning the data and without the need of a reference genome. Usage possibilities are numerous and fit the actual trend of sequencing, as more and more species, including meta-genomes, are sequenced without reference genomes

In this paper we presented the main MAPSEMBLER features and algorithmic ingredients. We have shown a selected overview of MAPSEMBLER applications. Benchmarks were ran on very large amount of data, as well as read sets containing biological elements of interest. We compared the simple targeted assembly operating mode of MAPSEMBLER to TASR, a recently published targeted assembler. MAPSEMBLER runs consistently faster and consumes less memory than TASR, as well as speed improvement over several orders of magnitude when multiple fragments are extended. More importantly, MAPSEMBLER has several novel features, such as iterative extensions and graph visualization.

Acknowledgements

This work was supported by the INRIA "action de recherche collaborative" ARC Alcovna. Authors warmly thank Vincent Lacroix, Claire Lemaitre, Pavlos Antoniou and Fabrice Legeai for their participation to discussions and their support on experimental results.

References

[1] M. S. Cline, M. Smoot, E. Cerami, A. Kuchinsky, N. Landys, C. Workman, R. Christmas, I. Avila-Campilo, M. Creech, B. Gross, K. Hanspers, R. Isserlin, R. Kelley, S. Killcoyne, S. Lotia, S. Maere, J. Morris, K. Ono, V. Pavlovic, A. R. Pico, A. Vailaya, P. Wang, A. Adler, B. R. Conklin, L. Hood, M. Kuiper, C. Sander, I. Schmulevich, B. Schwikowski, G. J. Warner, T. Ideker, and G. D. Bader. Integration of biological networks and gene expression data using cytoscape. *Nat. Protocols*, 2(10):2366–2382, Oct. 2007.

[2] R. Li, H. Zhu, J. Ruan, W. Qian, X. Fang, Z. Shi, Y. Li, S. Li, G. Shan, K. Kristiansen, et al. De novo assembly of human genomes with massively parallel short read sequencing. *Genome research*, 20(2):265, 2010.

[3] D. C. Richter, F. Ott, A. F. Auch, R. Schmid, and D. H. Huson. Metasim-a sequencing simulator for genomics and metagenomics. *PLoS ONE*, 3(10):e3373, 10 2008.

[4] R. Warren, R. Warren, and R. Holt. Targeted assembly of short sequence reads. *Nature Precedings*, 2011.

[5] R. L. Warren, G. G. Sutton, S. J. M. Jones, and R. A. Holt. Assembling millions of short DNA sequences using SSAKE. *Bioinformatics*, 23(4):500–501, Feb. 2007.

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399