SyntenyFinder: A Synteny Blocks Generation and Genome Comparison Tool

Intern: Ilya Minkin
Advisor: Son Pham

We present an algorithm for finding synteny blocks in genomes. The algorithm is based on de Bruijn graphs and graph simplifications. Our method is suitable for finding synteny blocks in genomes that contain regions of highly conserved DNA, i.e. genomes that are evolutionary close to each other. We evaluated our tool on several datasets that include different strains of *Mycobacterium tuberculosi* and *Pseudomonas aeruginosa.*We also show that with some modifications the algorithm can be applied in more complicated cases.

1. Introduction

Recent advances in high throughput sequencing and genome assembling technologies result in appearance of high number of completely sequenced genomes, ranging from bacteria to mammalian. It raises a lot of interesting biological question, and one of the most exciting one is the evolutionary relationship of all these species.

It is known that genome rearrangements play important role in the evolution [1, 2, 3, 4]. The rearrangements involve large portions of DNA and change their location and orientation. Particularly, rearrangements help bacteria in adaptation to various environments [5]. Another interesting evolutionary event is the whole genome duplication, which is usually followed by massive gene loss [6, 7].

Although there are algorithms that address problem of restoring evolutionary scenario for a given set of genomes [8], they require input genomes to be decomposed to a collection of synteny blocks – long regions of conserved DNA. Currently existing tools [10] are able to reconstruct synteny blocks from genomes represented as sequences of enumerated local alignments, or *anchors*. Usually, anchors denote homologous genes.

In this paper we propose *SyntenyFinder* – a tool for finding synteny blocks in genomes represented as nucleotide sequences. Our approach is based on de Bruijn graphs that are extensively used in bioinformatics for genome assembly [11, 12]. It requires input genomes to share exact substrings of fixed length ($>$ 100 bp). Therefore, our method can be applied directly to only closely related genomes, like different strains of bacteria. But with some modifications it can be extended to wider range of use.

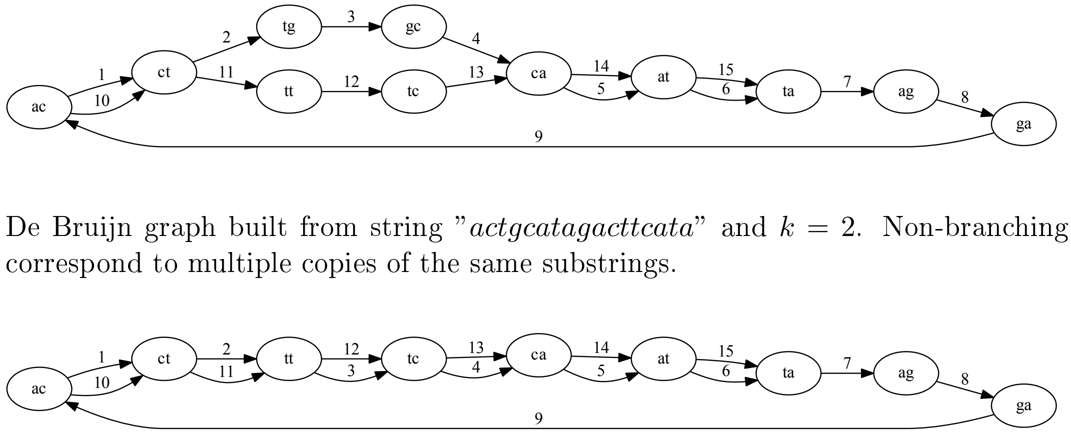
In *Overview* section we describe the problem definition and basic intuitions behind our method, section *Algorithm description* contains formal description of our method, *Results* describes results of evaluation of our program on several datasets.

2. Overview

Given a set $S = \{S_1, S_2, \ldots, S_n\}$ of chromosomes, where each chromosome is represented as a string over alphabet $\{A, C, G, T\}$. The task of finding synteny blocks is to find a set of so called conserved regions $C = \{C_1, C_2, \ldots, C_n\}$, where each conserved region C_i is a set of substrings of chromosomes from S. Such regions are supposed to cover most of the genome for closely related species. All substrings forming a conserved region C_i must be similar to each other according to some criterion of similarity. Note that problem of finding synteny blocks in a set of chromosomes is equivalent to a problem of finding synteny blocks in one superchromosome obtained from concatenating all chromosomes from the set – we can just separate chromosomes by special characters.

At this moment there is no generally accepted formal criterion of similarity exist, so the problem of finding synteny blocks is ill-defined. In our work we introduce new criterion of similarity based on de Bruijn graphs and graph simplifications.

As previously mentioned, our method is based on de Bruijn graph. Given a fixed value k and a string S we can build de Bruijn graph G from the string as follows. Let's denote by k-prefix of a string S the first k characters of S, and by k-suffix the last k characters of S. For each unique substring of length k (called k-mer) found in S, we add a vertex to G and mark it with corresponding k-mer. For each $(k+1)$-mer w found in S we add edge that connects vertex corresponding to k prefix of w with vertex corresponding to k suffix of w and label the edge with position of first character of w (multiedges with different labels are allowed).



(a) De Bruijn graph built from string ”*actgcatagacttcata*” and $k = 2$. Non-branching paths correspond to multiple copies of the same substrings.

(b) Same de Bruijn graph after simplification. Replacing substring ”*ctGca*” by ”*ctTca*” we obtain long non-branching path that corresponds to original synteny block.

Figure 1: Illustration of de Bruijn graphs and graph simplification

In this graph we consider only paths that have consecutive (that differ by 1) labels on edges. It is easy to see that with such restriction every path in G corresponds to a substring in S. Example of such de Bruijn graph built from the string $S =$ ”*actgcatagacttcata*” and $k = 2$ is depicted on Figure 1a.

Note that two copies of substring ”*catt*” form a non-branching path consisting of edges with multiplicity 2 in this graph. Single mismatch in substrings ”*ctGca*” and ”*ctTca*” form so-called "bulge", unoriented cycle generated by two valid paths with coinciding ends. If we replace one branch of the bulge by another (replace”*ctGca*” by ”*ctTca*”), we obtain a long non-branching path (Figure 1b).

This heuristic forms basis of our method – conserved regions in different parts of the genome contain conserved basepairs, but such regions are disrupted by indels and mismatches. These differences form bulges in the graph that make it different to infer structure of the synteny blocks. We remove bulges having size less than some predefined constant and thus obtain non-branching paths corresponding to the conserved regions. The process of removing bulges from the graph is called *simplification*. We sustain one-to-one correspondence between the graph and the string – when we change something in the graph, we also change appropriate characters in the string.

Conserved regions can be located on opposite strands of DNA. To handle this, we add edges that correspond to reverse complementary of the input string. We distinguish between direct and reverse complementary edges by coloring them in different colors.

Given a string S, for each $(k+1)$-mer found in S we add corresponding edge to the graph and color it *blue*, for each $(k+1)$-mer found in reverse-complementary counterpart of S we add corresponding edge to the graph and color it *red*. In this graph, non-branching paths with different colors represent synteny blocks located on opposite strands of DNA.

Complete pipeline is following:

1) Concatenate input chromosomes into one superchromosome
2) Build de Bruijn graph from the superchromosome
3) Simplify the graph
4) Output synteny blocks as non-branching path in the graph

Our algorithm depends on two parameters: k and δ (minimum allowed size of a bulge). It is reasonable to use as high k as possible ($k > 100$) to keep graph structure simple and avoid connecting regions that are actually not homologous. So, our method requires that conserved regions in input genomes contain exact shared k-mers. This is not a problem in genomes that are very close to each other (like different strains of a bacteria), but it can create difficulties in genomes that had separated a long time ago.

This issue can be solved, for example, by finding a set of all local alignments in the genomes and substituting one subsequence in each found alignment by another. In results section we will demonstrate on a practical half-synthetic example that with such modifications our method is able to handle complicated cases. So at this point our method is directly applicable to only evolutionary close genomes and in near future we plan to extend it to wider range of use.

3. Algorithm description

Given two numbers k and δ and a set $S = \{S_1, S_2, \ldots, S_n\}$ of chromosomes, where each chromosome $S_i = (s_{i,1}, s_{i,2}, \ldots, s_{i,m_i})$ is a string over alphabet $\Sigma = \{A, C, G, T\}$. Let's denote by $X_1 \uplus X_2$ concatenation of strings X_1 and X_2. We denote by $X[i,j]$ a substring of a string X that starts at i and ends at j, $X[i,j] = (x_i, x_{i+1}, \ldots, x_j)$. $Rev(X)$ means reverse-complementary counterpart of a string X.

First step of the algorithm is to obtain superchromosome $\hat{S} = S_1 \uplus S_2 \uplus \ldots \uplus S_n$. Concatenated strings are interleaved by special characters that indicate ends of the chromosomes, we omit these technical details. We build a linked list of pairs $L = ((\hat{s}_1, 1), (\hat{s}_2, 2), \ldots, (\hat{s}_m, m))$ and l_i is a pointer to i-th item in the list, function $Next(l_i) = l_{i+1}$ returns element after l_i in the list. First element of a pair represents character of the sequence, and second element represents it's original position in string $\hat{S}$. We preserve original indices of the characters to be able to compute coordinates of the synteny blocks after graph simplification. $Ptr(L) =$ is a set of all pointers of a list L. $\mathcal{P}\left(Ptr(L)\right)$ is a set of all subsets of $Ptr(L)$.

Colored de Bruijn graph is graph $G = (V, E)$ where $V = \Sigma^k$. Set of outgoing edges from vertex v is denoted by $Out(v)$. We define four functions:
1) $Pos : E \to Ptr(L)$
2) $Color : E \to \{Blue, Red\}$
3) $Spell : E \to \Sigma^{k+1}$
4) $Cover : E \to \mathcal{P}\left(Ptr(L)\right)$

For each $i \in \{1, 2, \dots, n - k\}$ we add two oriented edges to the graph:
1) $e^+ = (\hat{S}[i, i + k - 1], \hat{S}[i + 1, i + k])$, where:
$Pos(e^+) = l_i$
$Color(e^+) = Blue$
$Spell(e^+) = \hat{S}[i, i + k]$
$Cover(e^+) = \{l_i, l_{i+1}, \dots, l_{i+k}\}$
2) $e^- = (Rev(\hat{S}[i, i + k - 1]), Rev(\hat{S}[i + 1, i + k]))$, where:
$Pos(e^-) = l_{i+k}$
$Color(e^-) = Red$
$Spell(e^-) = Rev(\hat{S}[i, i + k])$
$Cover(e^-) = \{l_i, l_{i+1}, \dots, l_{i+k}\}$

A *valid* path in G is a sequence of edges $P = (e_1, e_2, \dots, e_n)$ iff $Pos(e_{i+1}) = Next(Pos(e_i))$ and $Color(e_{i+1}) = Color(e_i)$. Let's denote by $Start(P)$ the first vertex of the path P and by $End(P)$ the last vertex of P.

A pair of valid paths $B = \{b_1, b_2\}$ is called a *bulge*, iff following holds:
1) $Start(b_1) = Start(b_2) \wedge End(b_1) = End(b_2)$
2) There are no edges $e_1 \in b_1, e_2 \in b_2$ such that $Spell(e_1) = Spell(e_2)$
3) There are no edges $e_1 \in b_1, e_2 \in b_2$ such that $Cover(e_1) \cap Cover(e_2) \neq \emptyset$

A bulge $B = \{b_1, b_2\}$ is called *bad* iff $|b_1| < \delta \wedge |b_2| < \delta$ where δ is a parameter. A vertex v is called *bifurcation* iff there are at least two outgoing (ingoing) edges e_1, e_2 incident v such that $Spell(e_1) \neq Spell(e_2)$. Function $MaxBifDegree(p)$ returns highest degree of a bifurcation that lies on the path p.

A set of paths $P_{nb} = \{P_1, P_2, \dots, P_n\}$ is said to form a *non-branching path* iff $|P_1| = |P_2| = \dots |P_n|$ and $Spell(e_{i,k}) = Spell(e_{j,k})$, where $e_{i,j}$ denote j-th edge in the i-th path, i.e. all paths spell the same substring.

Let's illustrate above definitions on a simple example. Colored de Bruijn graph built

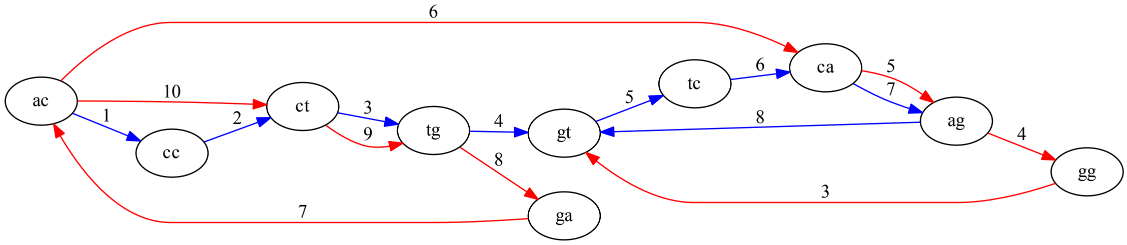


Figure 2: Colored de Bruijn graph for string $S = "acctgtcagt"$

from string $\hat{S} = "acctgtcagt"$ is depicted on figure 2. Here $Rev(\hat{S}) = "actgacaggt"$. Edges' labels denote indices of elements of list L. Vertices $"ac", "ct", "tg"$ are bifurcations, while $"cc", "tc", "ga"$ are not. Two paths $("ac", "ct")$ and $(("ac", "cc"), ("cc", "ct"))$ form a bulge. Two multiedges $("ct", "tg")$ form a non-branching path.

Pseudocode of our algorithm for synteny blocks finding is presented below. Procedure of building a de Bruijn graph is described at the start of the section. Function $ProbeBadBulge(e_1, e_2, \delta)$ trivially extends paths starting from edges e_1 and e_2 until they form a bad bulge or the branch length threshold gets exceeded. It returns set of two paths if bad bulge was found and an empty set otherwise. Procedure $CollapseBadBulge(b_1, b_2, L, G)$ modifies the list L so that in the graph G path b_2 gets replaced by path b_1. Note that at any moment there is one-to-one correspondence between the list and the graph – any changes in L are immediately reflected in G. Procedure $OutputNonBranchingPaths(G)$ finds and outputs all maximal unique non-branching paths in G and calculates starting and ending indices of the substring of the string $\hat{S}$ that these paths correspond.

```
$SyntenyFinder(S = \{S_1, S_2, \ldots, S_n\}, k, \delta)$
$\hat{S} = S_1 \uplus S_2 \uplus \ldots \uplus S_n$
$L = ((\hat{s}_1, 1), (\hat{s}_2, 2), \ldots, (\hat{s}_m, m))$
$G = BuildDeBruijnGraph(L, k)$
$run = True$
while $run == True$ do
    $run = False$
    for $v \in V(G)$ do
        for $e_1, e_2 \in Out(v)$ do
            $\{b_1, b_2\} = ProbeBadBulge(e_1, e_2, \delta)$
            if $b_1 \neq \emptyset \, and \, b_2 \neq \emptyset$ then
                $run = True$
                if $MaxBifDegree(b_1) > MaxBifDegree(b_2)$ then
                    $CollapseBadBulge(b_1, b_2, L, G)$
                else
                    $CollapseBadBulge(b_2, b_1, L, G)$
$OutputNonBranchingPaths(G)$
```

4. Experimental results

We have implemented our algorithm in C++. To evaluate performance of our program, we have performed several tests.

First dataset include three strains of *Mycobacterium tuberculosis* H37Rv: laboratory reference strain H37Rv, CCDC5180, CCDC5079. We used $K = 1000$ and $\delta = 5000$ as parameters for this running. Our tool found 19 synteny blocks shared between all three genomes. These blocks cover 96% of the genome.

[I'll add evaluation on *Pseudomonas aeruginosa* a bit later]

We also performed evaluation on well known yeast dataset including *K.waltii* and *S. cerevisiae*. Paper [7] describes 253 so called regions of *double conserved synteny*: DNA that are conserved between two regions in *S. cerevisiae* and one region in *K.waltii*. Our method cannot be applied directly for this dataset since homologous genes in these genomes lack exact shared k-mers even for $k < 20$. We used matching of genes between these two genomes to enrich number of shared k-mers – for each pair of homologous genes we substituted sequence of one gene by another in the genome. We ran our program on the resulting enriched genome with $k = 1000$ and $\delta = 20\,000$ and selected blocks shared across all three genomes having size at least 1000 base pairs. We have found 270 such blocks and 185 of them match blocks described in [7]. Total length of the blocks (we count only *S. cerevisiae* regions) described in [7] is 8 390 452 bp, total length of our blocks is 7 773 519 bp, and 6 641 468 bp from our blocks overlap with blocks from [7]. Our blocks cover $\frac{7\,773\,519}{12\,495\,682} \times 100\% = 62\%$ of *S. cerevisiae* genome. Most blocks from [7] that are unmatched by our blocks are short (less than 10 000 bp), many of them look inconsistent. This half-synthetic example shows that our method is indeed correct and after incorporating a local alignment tool our algorithm can be used to find synteny blocks even in complicated cases.

5. Conclusion

In this work we present an algorithm, that is able to find synteny blocks from unannotated sequences. It can be applied to genomes of species that are evolutionary close to each other. With some modifications our method can be extended for finding synteny blocks in more distant genomes.

References

[1] Pavel Pevzner, Glenn Tesler. Genome rearrangements in mammalian evolution: lessons from human and mouse genomes. Genome Res. 2003 January 1; 13(1): 37–45.

[2] Pavel Pevzner, Glenn Tesler. Human and mouse genomic sequences reveal extensive breakpoint reuse in mammalian evolution PNAS June 24, 2003 vol. 100 no. 13 7672-7677

[3] Guillaume Bourque, Pavel A. Pevzner, and Glenn Tesler. Reconstructing the Genomic Architecture of Ancestral Mammals: Lessons From Human, Mouse, and Rat Genomes. Genome Res. 2004. 14: 507-516. Genome Research. 2003 Jan;13(1):37-45.

[4] David Sankoff, Joseph H. Nadeau. Chromosome rearrangements in evolution: From gene order to genome sequence and back. PNAS September 30, 2003 vol. 100 no. 20 11188-11189

[5] Fabien Aujoulat, Frederic Roger, Alice Bourdier, Anne Lotthe Brigitte Lamy, Helene Marchandin, Estelle Jumas-Bilak. From environment to man: genome evolu-

tion and adaptation of human opportunistic bacterial pathogens. Genes 2012, 3(2), 191-232.

[6] The Arabidopsis Genome Initiative. Analysis of the genome sequence of the flowering plant Arabidopsis thaliana. Nature 408, 796-815 (14 December 2000)

[7] Manolis Kellis, Bruce W. Birren, Eric S. Lander. Proof and evolutionary analysis of ancient genome duplication in the yeast Saccharomyces cerevisiae. Nature 2004 Apr 8;428 (6983): 617-24.

[8] Max A. Alekseyev, Pavel A. Pevzner. Breakpoint graphs and ancestral genome reconstructions. Genome Res. 2009 May; 19(5): 943–957.

[9] Genome 10K Community of scientists. Genome 10K: A proposal to obtain whole-genome sequence for 10000 vertebrate species. Journal of Heredity, 100(6): 659-674.

[10] Son K. Pham, Pavel A. Pevzner. DRIMM-Synteny: decomposing genomes into evolutionary conserved segments. Bioinformatics (2010) 26 (20): 2509-2516.

[11] Pavel A. Pevzner, Haixu Tang, Michael S. Waterman. An Eulerian path approach to DNA fragment assembly. Proc. Natl. Acad. Sci. USA. 2001 Aug 14; 98(17): 9748-53.

[12] Zamin Iqbal, Mario Caccamo, Isaac Turner, Paul Flicek, McVean. De novo assembly and genotyping of variants using colored de Bruijn graphs. Nat Genet. 2012 Jan 8;44(2):226-32