PATHRACER: racing profile HMM paths on assembly graph

Alexander Shlemov¹[0000–0002–2317–8587] and Anton Korobeynikov^{1,2}[0000–0002–2937–9259]

¹ Center for Algorithmic Biotechnology
Saint Petersburg State University
Saint Petersburg, 198504, Russia

² anton@korobeynikov.info

Abstract. Recently large databases containing profile Hidden Markov Models (pHMMs) emerged. These pHMMs may represent the sequences of antibiotic resistance genes, or allelic variations amongst highly conserved housekeeping genes used for strain typing, etc. The typical application of such a database includes the alignment of contigs to pHMM hoping that the sequence of gene of interest is located within the single contig. Such a condition is often violated for metagenomes preventing the effective use of such databases.

We present PATHRACER — a novel standalone tool that aligns profile HMM directly to the assembly graph (performing the codon translation on fly for amino acid pHMMs). The tool provides the set of most probable paths traversed by a HMM through the whole assembly graph, regardless whether the sequence of interested is encoded on the single contig or scattered across the set of edges, therefore significantly improving the recovery of sequences of interest even from fragmented metagenome assemblies.

Keywords: profile HMM · graph alignment · set of most probable paths
Availability: <http://cab.spbu.ru/software/pathracer/>

1 Introduction

The recent advances of metagenomics quickly revealed scalability issues arising from the amount of raw sequencing data necessary to describe complex microbial communities. Even more, due to many inherent problems and challenges the assembly of metagenomic data remains a non-trivial task, thus slowing down biological discoveries [13]. Still, in the majority of cases the interest of researchers lies around the recovery of certain important genes this way the use of additional information about these genes might be crucial for the full gene sequence recovery even from very fragmented assemblies.

The typical way to represent the sequence of the particular gene family is via so-called profile Hidden Markov Models [5]. Recently large databases such as Pfam [8] or NCBIfam-AMR [1] containing thousands of pHMMs emerged. These

pHMMs may represent the sequences of antibiotic resistance genes, or allelic variations amongst highly conserved housekeeping genes.

The recent releases of XANDER [19] and MEGAGTA [11] tools opened the possibility of the gene-targeted metagenomics assemblies, where the trained Hidden Markov Model is used to guide the traversal of de Bruijn graph. Such an approach gives obvious advantage over other assembly methods. Still, these tools might be somehow non-trivial to use, for example, XANDER forces a user to build both forward and reverse HMMs and MEGAGTA requires original sequences the pHMM was built from. All these requirements makes the usage of these tools non-straightforward as one cannot simply download the HMM from the database and use it straight away. Even more, both tools includes their own genome assembler engines essentially ignoring the recent progress made in the field of genome and metagenome assemblies.

PATHRACER is a novel standalone tool that aligns profile HMM directly to the assembly graph produced by modern assemblers in standard GFA format¹. This way PATHRACER could utilize all the features and improvements (e.g. hybrid or mate-pair assemblies) provided by state-of-the-art assemblers. Below we describe algorithmic approaches used in PATHRACER and showcase them over several datasets.

2 Methods

2.1 General Definitions

Let *Sequence graph* G be a directed graph where each vertex (called *position*) is labelled by a letter from the given alphabet Σ . Thus, each path in G corresponds to some string in Σ^* . Denote by $V(G)$ and $E(G)$ the sets of vertices and edges of G correspondingly.

pHMM graph is a profile HMM in HMMER 3 *Plan 7* format [6, 5] on Σ . Profile HMM can be viewed as weighted directed graph with vertices called *states* and edges called *transitions*. A particular state could be of one of the following types: I (*insertion*), M (*match or mismatch*), D (*deletion*); in addition to this there are also two special states denoted by START and END (see Figure 1). I and M states together are called *emissive states* or *emitters*. Each emissive state E contains its own emission probabilities $\mathcal{E} = (E, \cdot)$ that define a probability distributions on Σ . Edges weights denoted as $\mathcal{T}(\cdot, \cdot)$ are transition probabilities; in each vertex sum of its outgoing transitions equals to 1. Note that pHMM graph is *almost* acyclic: the only allowed cycles are simple loops from each I state to itself (*I-loops*). Therefore, pHMM states could be topologically sorted.

Background distribution $\mathcal{B}$ is a discrete probability distribution on Σ . This distribution is an *a priori* distribution of letters in the model. See [6] for more information about background distribution and its role.

¹ So far only only GFA from de Bruijn graph assemblers like SPADes and MEGAHIT is supported, but we will address this restriction in the next PATHRACER versions

Fig. 1. Plan 7 profile hidden Markov model (pHMM) scheme

For string $(L_1, \dots, L_n) \in \Sigma^*$ *HMM alignment* is a path in the pHMM graph that contains exactly n emitters. If the path begins from START and ends with END the alignment is called *global*, otherwise the alignment is called *local*. For the sake of simplicity we will consider only the problem of global alignment search. Note that the local alignments could be turned into the global ones by a simple transformation of pHMM graph [6]: insert additional transitions from START to all emitters and from all emitters to END.

Furthermore, the global alignment is fully defined by the series of its emitters $(E_1, \dots, E_n)$. Indeed, the “missed” intermediate D states in the alignment path could be unambiguously reconstructed due to a particular structure of pHMM.

For the given pHMM graph, background distribution $\mathcal{B}$, string $\mathcal{L} = (L_1, \dots, L_n)$, and its global alignment $\mathcal{A} = (E_1, \dots, E_n)$ alignment score $Score(\mathcal{L}, \mathcal{A})$ is defined as:

$$Score(\mathcal{L}, \mathcal{A}) = \log(\mathcal{T}(\text{START}, E_1)) + \log(\mathcal{E}(E_1, L_1) - \log(\mathcal{B}(L_1))) + \\ \log(\mathcal{T}(E_1, E_2)) + \dots + \log(\mathcal{E}(E_n, L_n) - \log(\mathcal{B}(L_n))) + \log(\mathcal{T}(E_n, \text{END})),$$

where $\mathcal{T}(E_i, E_{i+1})$ may include intermediate D states $D_i^{(1)}, D_i^{(2)}, \dots, D_i^{(T)}$:

$$\log(\mathcal{T}(E_i, E_{i+1})) = \log(\mathcal{T}(E_i, D_i^{(1)})) + \dots + \log(\mathcal{T}(D_i^{(T)}, E_{i+1})).$$

This score is essentially

$$\log \frac{Prob(\text{“string produced by this sequence of HMM states”})}{Prob(\text{“string is randomly obtained from the background distribution”})}$$

and is known as *log-odds score* [6].

For given HMM graph and background define the alignment score $Score(\mathcal{L})$ for the string $\mathcal{L} = (L_1, \dots, L_n)$ as the best score along all its possible global alignments $\mathcal{A} = (E_1, \dots, E_n)$.

We consider the following problem of graph alignment to profile HMM: for given sequence graph G , profile HMM $pHMM$, background distribution $\mathcal{B}$, and integer $k > 0$, among all the paths in G obtain k different paths with the highest possible scores. We will call this problem *Top(k) path problem* throughout the article.

It is well known that for the one string the best score alignment could be found by Viterbi dynamic programming algorithm [18]. Here we seek for the generic solution of the alignment problem: (1) among all paths in the sequence graph, and (2) obtaining k best paths instead of just one.

2.2 Event Graph

We present the solution of $Top(k)$ path problem via the special data structure called *event graph*.

First, define the expanded sequence graph $\bar{G}$ as graph G with added special EMPTY vertex with no letter on it. Connect EMPTY to all other vertices by bidirectional edges.

Each vertex (or *event*) of an event graph is a pair (E, P) of an emissive (I or M) HMM state E and a non-empty position P of $\bar{G}$ or one of two special vertices: SOURCE = (START, EMPTY) and SINK = (END, EMPTY).

A directed edge from event (E_1, P_1) to event (E_2, P_2) exists only if and only if: there is an edge from P_1 to P_2 in $\bar{G}$; emitters are consequent, i.e., there is a path in HMM from E_1 to E_2 , that does not contain other emitters. Due to the used pHMM structure there could be only one such path between the states.

The weight of vertex (E, P) equals to $\log(\mathcal{E}(E, Letter(P))) - \log(\mathcal{B}(Letter(P)))$. Edge weight is a sum of logarithms of the probabilities of transitions on the edge. Since these probabilities are always less or equal to 1, edge weights are always non-positive, while vertices could have positive weights due to the presence of background distribution.

One can easily see that each path from SOURCE to SINK corresponds to a global alignment of the pHMM against some path in G whereas path weight (sum of weights of edges and vertices along the path) is equal to the alignment score.

2.3 Top(1) Path Problem via Event Graph

Event graph allows one to construct the solution of Top(1) problem straight from the definition.

Algorithm 1: Top(1) path finding

Input: sequence graph G , profile HMM $pHMM$, background distribution $Background$

- 1 Construct event graph from G , $pHMM$, and $Background$
 - 2 Find the best (highest scored) event path from SOURCE to SINK
 - 3 Extract sequence path (series of positions) and the correspondent string from the best event path
-

One could easily see that the problem of finding the best event path in general does not have a finite solution. Indeed, if the event graph is acyclic or at least has no cycles of positive weight, then the solution will certainly exist. In the general case event graph could have cycles due to the presence of I-loops in pHMM and depending to a particular background distribution some of these cycles may positive weight. If there is a path from START to END that contains one of such cycles, then one can roll on it endlessly increasing the score and therefore the proper solution would not exist.

Fortunately, this problem is more theoretical than practical. In our implementation we use a simple heuristic to deal with it. We find all I-loops of positive weight in pHMM and unroll them into C (default: 10) sequential I states. This procedure is equivalent to restricting the total number of emissions. Typically there is only small fraction of such I-loops (not exceeding 2.5% for all considered HMMs and therefore the size of pHMM graph increases negligibly.

2.4 Dynamic Programming Algorithm for Best Path Extraction

For each vertex and each edge of the event graph find the best (having largest score) path from SOURCE. Denote those best score values as $DIST(\text{vertex})$ and $DIST(\text{edge})$. Note that due to absence of positive-weighted cycles those $DIST$ s always exist and are finite.

The values of $DIST$ s could be calculated via the Bellman-Ford algorithm [4]. However, the computational complexity of this approach is $\mathcal{O}(|V(\text{EventGraph})| \times |E(\text{EventGraph})|)$, that is completely impractical. Still, due to special layered event graph structure, $DIST$ s could be computed significantly faster level-by-level. One possible approach to this is described in Algorithm 2.

The only non-straightforward step in Algorithm 2 is I-loop relaxation. Due to negativity of all I-loop edges we can sequentially update I-loop events $DIST$ using a priority queue as in classical Dijkstra algorithm [4]. Overall algorithm complexity is $\mathcal{O}(|pHMM| \times |E(G)| \log |V(G)|)$, where log comes from priority queue push/pop costs.

For sequence graphs $|E(G)| \sim |V(G)|$ since vertex degrees are bounded by above (e.g. by 4 for de Bruijn graphs), therefore, we will count them together denoting $|G| = |V(G)| + |E(G)|$. Thus, the overall complexity is $\mathcal{O}(|pHMM| \times |G| \log |G|)$

Having $DIST(\text{SINK})$ calculated we can easily reconstruct the best path by backtracking: start from SINK vertex and go backwards into each vertex taking incoming edge with the highest $DIST$ until we reach SOURCE. k best paths could be found iteratively using the observation that the i -th shortest path in the sequence must branch from one of the $i - 1$ shortest paths already identified (so-called Eppstein algorithm, [7]).

2.5 From Top(1) to Top(k)

Though there is a simple solution for the extraction of the best k paths from the event graph, the problem of finding top sequence graph paths is still non-trivial and could not be easily solved by extracting k best paths from SOURCE to SINK.

The first problem here is that top k alignments are not equal to top k sequences. Indeed, full event graph represents all possible alignments of the pHMM against all sequence graph paths. Each sequence path has a combinatorial number of alignments against the pHMM. And each alignment corresponds to its own path in the event graph. The highest scored path in the full event graph indeed corresponds to the best alignment of the best sequence path. However,

Algorithm 2: Level-by-level *DIST* computation (optimized Bellman-Ford algorithm)

Input: Event graph (along with sequence graph G and $pHMM$ with unrolled positive I-loops)
Output: *DIST* values for graph vertices and edges

```

1   $DIST(\text{SOURCE}) := 0$ 
2  foreach  $edge \in \text{Outgoing}(\text{SOURCE})$  do
3     $DIST(edge) := \text{weight}(edge)$ 
4  end
5  foreach emissive state  $E \in pHMM$  sorted in a topological order do
6    // Initialization from preceding levels
7    foreach position  $P \in G$  do
8      Let vertex  $v = (E, P)$ 
9       $DIST(v) := \text{weight}(v) + \max_e DIST(e)$ , where  $e \in \text{Ingoing}(v)$  and not
      an I-loop edge
10   end
11   if emissive state  $E$  is I-loop (not unrolled) then
12     // I-loop relaxation
13      $Q :=$  empty priority queue
14     foreach position  $P \in G$  do
15       Push vertex  $v = (E, P)$  to  $Q$  with current  $DIST(v)$  as a priority
       value
16     end
17
18     while  $Q$  is not empty do
19       Extract vertex  $v$  with the highest priority from  $Q$ 
20       for  $edge \in \text{Outgoing I-loop edges}(v)$  do
21         Consider vertex  $v' = \text{End}(e)$ 
22         if  $DIST(v') < DIST(v) + \text{weight}(e) + \text{weight}(v')$  then
23            $DIST(v') := DIST(v) + \text{weight}(e) + \text{weight}(v')$ 
24           Increase vertex  $v'$  priority in  $Q$ 
25         end
26       end
27     end
28   end
29   // Outgoing edges setup
30   foreach position  $P \in G$  do
31     Let vertex  $v = (E, P)$ 
32     foreach  $edge \in \text{Outgoing}(v)$  do
33        $DIST(e) := DIST(v) + \text{weight}(e)$ 
34     end
35   end
36 end
37  $DIST(\text{SINK}) := \max_e DIST(e)$ , where edge  $e \in \text{Ingoing}(\text{SINK})$ 

```

the second best alignment usually corresponds to the second best alignment of the same sequence path and not of some other sequence path.

The second issue is driven by particular applications. Among top k paths we would like to see really different paths rather than paths that differ only by adding/trimming a small number of letters at the beginning/end. Typically extending/trimming the sequence in such way changes the final score only slightly and variations of the best path displace all other paths from the top list.

The third issue is that the number of different sequence paths with the proper score could also be combinatorial. For the string with d single-letter variations (corresponding to a path with d bubbles) the number of different paths would be 2^d . All the paths would be slightly different and would probably have the same or almost the same score. Without additional information we could not prefer one path to another and report an answer of a reasonable (non-combinatorial) size.

This way the proper $\text{Top}(k)$ problem statement might look like as follows: find k sufficiently (in the terms of the second issue) different paths in the initial sequence graph that maximize alignment score, but such problem seems to be very hard to solve. The only solution we know is increasing the number of extracted top alignments and further result filtering, but this approach is impractical for large k since the number of paths to extract is not known in advance. Also, the construction, storage and operations on full event graph (consisting of $|pHMM| \times |G|$ vertices) is time and memory consuming.

To deal with these issues we propose an heuristic solution that could significantly speed up $\text{Top}(k)$ paths extraction making it possible even for very large k (say, $k > 100000$) that could be useful for complicated metagenome graphs where the number of candidate paths could be really large. This approach introduces a new structure representing alignment of the whole sequence graph against the pHMM that could be also interesting and useful by itself. Particularly, it helps to overcome the third issue, allowing one to consider relatively compact object rather than an endless number of slightly different paths.

2.6 Collapsing Event Graph

Let us consider a subgraph of the full event graph called collapsed and trimmed event graph.

Collapsed event graph is a subgraph of an event graph where for each vertex (E, P) (including SOURCE and SINK) all its ingoing and outgoing neighbours (vertices linked by ingoing and outgoing edges respectfully) have different positions. This way different START-to-END paths in collapsed event graph correspond to different paths in the initial sequence graph.

Trimmed event graph is a subgraph where each vertex does not have ingoing edge from SOURCE and ingoing edge(s) from non-SOURCE vertex at the same time. The same property is ensured for outgoing edges and SINK.

All the different SOURCE-to-SINK paths in collapsed and trimmed event graph correspond to the paths in the initial sequence graph that are not substring or superstring of each other and do not have a perfect overlap, i.e. any prefix of one path cannot not be a suffix of another.

If we perform k best paths finding algorithm on collapsed and trimmed event graph, we extract k non-trivially different sequence paths. There are multiple ways how event graph could be trimmed and collapsed. Algorithms 3 and 4 make an arbitrary subgraph of an event graph collapsed and trimmed but still keeping the best scored paths intact.

Algorithm 3: Event graph collapsing

Input: Event graph or its arbitrary subgraph

- 1 For each vertex and each edge calculate the $DIST(\text{vertex})$ and $DIST(\text{edge})$ values using the same DP procedure as in best path finding algorithm (Section 2.4)
 - 2 For each vertex, group all incoming edges by position of an incident vertex. In each group find the edge with the highest score and remove other edges from the graph
 - 3 Do steps 1 and 2 on the graph backwards: with all the edges reversed and going from SINK to SOURCE
-

Algorithm 4: Event graph trimming

Input: Event graph or its arbitrary subgraph

- 1 Compute $DIST(\text{edge})$ for each edge as in the previous algorithm
 - 2 For each vertex incident to SOURCE: remove all incoming edges worse than the incoming edge from SOURCE. If there are still other incoming edges, remove the edge that goes from the SOURCE vertex
 - 3 Do steps 1 and 2 on the graph with all the edges reversed considering edges going to SINK
-

After collapsing and trimming we perform graph cleanup by removing all vertices unreachable from SOURCE or SINK.

Algorithm Discussion and Analysis Both algorithms preserve the best path intact since for each vertex that belongs to the best path incoming and outgoing edges with maximum $DIST$ value belong to the best path as well.

Collapsing procedure preserves existing sequence paths, but may reduce their scores. For each SOURCE-to-SINK path that got broken during the collapsing step there still exists another event path with the same corresponding sequence path. Note that collapsing removes edges, but it leaves one edge going to each position, in other words: for each sequence path $P_1, \dots, P_n$ collapsing procedure will preserve at least one corresponding event path.

Collapsing procedure can reduce score of a path therefore yielding suboptimal alignment. If the sequence path shares prefix with another path with better score,

the prefix alignment will be inherited from the other path. The same goes for the suffix. Our experiments show that this phenomenon is usually not a big problem for the real graphs. Even more, in order to address this problem one could re-align the extracted sequence paths against pHMM.

Trimming procedure could break existing sequence path but if it does so, then there does exist a path slightly different from this one with better score.

Full event graph represents all possible alignments of all graph paths against the pHMM. Collapsed and trimmed event graph represents an alignment (probably suboptimal, but still reasonable) of the whole graph against the pHMM. It could be viewed as an analogue of partial order graph alignment [10] but for graph paths rather than separate homologous sequences.

2.7 Alignment Algorithm: Collapsed Event Graph Construction

Collapsed trimmed event graph takes sufficiently less space and could be constructed on-fly, therefore we do not need to construct full graph and then collapse it. Algorithm 5 represents such an approach.

Algorithm 5: pHMM alignment against sequence graph

Input: Sequence graph G , profile HMM $pHMM$ with unrolled positive I-loops, k — the number of best paths to be extracted

- 1 Start with vertex $SOURCE = (START, EMPTY)$
- 2 Iterate over pHMM emissive states in a topological order
- 3 **foreach** state E **do**
- 4 Construct new event vertices (E, P) along with ingoing edges for all positions $P \in G$
- 5 For all new edges and vertices compute $DIST(edge)$ and $DIST(vertex)$ and perform a step of collapsing and trimming as in Algorithms 3 and 4
- 6 Remove all vertices that definitely could not be reached from $SINK$
- 7 **end**
- 8 Add final vertex $SINK = (END, EMPTY)$
- 9 Perform backward collapsing and backward trimming (forward collapsing and trimming are already performed on-line)
- 10 Extract best k paths going from $SOURCE$ using score values computed during backward collapsing using Eppstein algorithm [7]
- 11 Report best paths along with the collapsed trimming event graph

Two parts contribute to the algorithm complexity: collapsed event graph construction and extraction of best paths. The first part has the computational complexity of $\mathcal{O}(|pHMM| \times |G| \log |G|)$, the same as general Top(1) algorithm (see Section 2.4). The second part (Eppstein algorithm [7]) has complexity $\mathcal{O}(kN \log(kN))$, where N is average path length and $\log$ is from priority queue overhead. Note that in collapsed event graph incoming vertex degree is bounded like in sequence graph (usually by 4 for nucleotide graphs), therefore there is no additional term in the complexity equation.

2.8 Implementation and heuristics

We implemented the described approximate $\text{Top}(k)$ algorithm solution in the PATHRACER tool. PATHRACER takes as input de Bruijn graph in GFA format (only SPADES or SPADES-compatible graphs are currently supported) and nucleotide pHMM. For amino acids HMMs we perform on-fly translation that allows us to consider the initial graph as a sequence graph over extended (20 amino acids and stop codon) amino acid alphabet.

The algorithm 5 has overall complexity $\mathcal{O}(|pHMM| \times |G| \log |G| + kN \log(kN))$, however, the first component in the sum might be intractable for large graphs. In order to reduce the computation complexity we implemented several heuristics:

1. We run HMMALIGN algorithm from HMMER 3 [6] on the edges of the input sequence graph and consider only the neighborhood of suitable size of matched edges: we walk forward and backward from each matched edge not far than the length of the corresponding pHMM overhang and then join all visited positions and extract sequence subgraph. HMMER options (e.g. domain E-value thresholds) are available to control the size of seed set.
2. During the event graph construction after each step new vertices are filtered out:
 - if the continuation of alignment through this vertex will always reach a dead-end before reaching SINK vertex;
 - if the continuation of alignment through this vertex will produce stop codon before reaching SINK vertex (for amino acid HMM);
 - all vertices with the score lower than a threshold parameter (default: -250);
 - all vertices except T top score vertices. The number T is reducing while moving along the pHMM.

This heuristic improves both the running time and memory consumption drastically (see Supplementary Table 1) with no significant changes in the results.

3 Results

In order to demonstrate the viability of our solution we conducted four experiments using the variety of datasets and pHMMs.

3.1 23S Search in *E. coli str. K12* Assembly

E. coli genome contains seven ribosomal RNA (rRNA) operons. Each operon contains a 16S rRNA gene, a 23S rRNA gene, and a 5S rRNA gene (except for one operon, which contains two 5S rRNA genes) interspersed with various tRNA genes.

We considered the *E. coli str. K12* dataset from [3]. The Illumina reads were of length 100 bp with mean insert length 270 bp. SPADES 3.12 [13] assembly was performed in the normal multi-cell isolate mode with default settings and

the resulting assembly graph in GFA format was obtained. Quick check of the results revealed that SPAdes assembled the 16S gene on the single contig, but was unable to derive the whole sequence of 23S gene. Probably the reason is that among seven copies of 23S gene, six were inexact, still 3 variants of 23S gene were scattered over 3, 3 and 2 edges correspondingly. Other variants were absent in the assembly graph and probably were collapsed during the graph simplification procedures.

We aligned final assembly graph to prokaryotic rRNA pHMM models from [16] and all 3 variants of 23S gene were the top paths extracted by PATHRACER (see Figure 2). No other (false) paths were produced.

Fig. 2. *E. coli str. K12*: extracted 23S paths (yellow-violet-red, green-violet-red and blue-red) and their neighborhood

3.2 16S Components of SYNTH Mock Metagenome Dataset

To showcase the ability of PATHRACER to deal with complex metagenomic data we considered 16S genes contained in SYNTH mock metagenome dataset from [17].

Synthetic community data set (SYNTH) is a set of reads from the genomic DNA mixture of 64 diverse bacterial and archaeal species (SRA acc. no. SRX200676) that was used for benchmarking the OMEGA assembler [9]. The dataset contains 109 million Illumina HiSeq 100-bp paired-end reads with mean insert size of 206 bp. The reference genomes for all 64 species forming the SYNTH dataset are known.

The assembly was performed by METASPADES 3.12 [13] with default parameters. To increase the sensitivity we considered *strain* (rather than the final *consensus*) assembly graph. We aligned the graph against 16S profile HMM from [16] and analyzed paths reported. PATHRACER reported 1088577 paths. It is a reasonable number since the corresponding graph component is very complicated (see Figure 3) and therefore many chimeric alignments are expected.

From SILVA [15] we obtained 179 16S sequences (most of the species have several variants of this gene), constructed BLAST database from all reported paths and performed a search in this database of all 16S gene sequences from SYNTH. For 55 out of 64 species all the 16S sequences were found with >95% BLAST identity. For 22 species at least one of 16S sequences was found with 100% BLAST identity (some variations were missing).

Fig. 3. SYNTH: 16S matches of known sequences and their neighborhood. Different species are colored differently

3.3 Urban Wastewater Metagenome

In [12] comparative metagenomics was used to investigate the occurrence of antibiotic resistant genes in wastewater and urban surface water environments in Singapore. For this experiment we took H1 (clinical isolation ward, SRA acc. no SRR5997548) dataset and performed assembly by METASPADES 3.12 with default settings. As with SYNTH dataset we used *strain* assembly graph to increase the sensitivity of the search.

The assembly graph was aligned by PATHRACER against selected beta-lactamase gene pHMMs obtained from NCBIfam-AMR database [1]: bla_{CTX-M} , bla_{IMP} and bla_{TEM} among the others. The resulting paths consisting of several graph edges could be seen on Figures 4,5,6.

Fig. 4. bla_{CTX-M} paths and their neighborhood. Green path corresponds to CTX-M-15 family; blue and red corresponds to CTX-M-9 and CTX-M-14 respectively

Different non-overlapping paths on the Figure 4 represents several different families of bla_{CTX-M} gene: green corresponds to CTX-M-15 family, while blue and red corresponds to CTX-M-9 and CTX-M-14 respectively (differs from each other only by 2 SNPs). Note that Table S5 in [12] mentions the presence of CTX-M-15 and CTX-M-18 beta-lactamase families in this dataset, however we found that the extracted graph path that corresponds to CTX-M-9 aligned to the

Fig. 5. *bla*_{IMP} paths and their neighborhood. Red path corresponds to IMP-1 family

Fig. 6. *bla*_{TEM} paths and their neighborhood. Red path corresponds to TEM-1 family

reference with 100% identity. Since the amino acid sequence of the CTX-M-18 beta-lactamase differs from that of the CTX-M-9 beta-lactamase by an Ala-to-Val change at position 231 [14], the additional checking of the results of [12] might be necessary (in [12] different gene sequences were searched in reads rather than assemblies and therefore might be prone to sequencing errors).

3.4 PATHRACER and MEGAGTA Running Time and Memory Consumption

We compared PATHRACER performance with MEGAGTA using 16 threads on urban dataset on all 159 beta-lactamase pHMMs obtained from NCBIfam-AMR database [1]. The memory consumption and running times are presented in Table 1.

We need to note that along with the input profile HMM files MEGAGTA also requires original gene sequences and a set of gene sequences for FRAMEBOT tool [20] that is used inside XANDER/MEGAGTA pipeline. Contrary to this, PATHRACER needs input graph and pHMM file only. We have not included XANDER into the benchmarking since MEGAGTA is its improved version and it outperforms XANDER as it shown in [11].

Table 1. PATHRACER and MEGAGTA benchmark: running time and peak memory consumption. *urban* dataset, 159 beta-lactamase HMMs, 16 threads. PATHRACER performance with disabled event graph online filtering is also shown

Tool	Graph assembly	pHMM alignment
MEGAGTA	30m / 30GiB	11h / 30GiB
METASPADES + PATHRACER	1h50m / 20GiB	50m / 7GiB
METASPADES + PATHRACER (no filtering)	1h50m / 20GiB	2h20m / 11GiB

4 Conclusion

PATHRACER utilizes both the assembly graph topology and the information about the known genes encoded in profile HMM during its operation. This way

Table 2. PATHRACER and MEGAGTA results for *urban* data. blaCTX-M, blaIMP, blaTEM beta-lactamase gene families. Annotation performed by online NCBI BLAST [2], only results with 100% coverage and identity included

Gene family	As shown in [12]	MEGAGTA	PATHRACER
<i>bla</i> _{CTX-M}	blaCTX-M-15; blaCTX-M-18	OXY-1-1/2/3 (177aa); CTX-M-15 (291aa); CTX-M-3 (291aa); CTX-M-9 (291aa); SFO-1 (295aa); CTX-M-14 (291aa); + 4 chimeric sequences (247aa, 203aa, 295aa, 242aa)	CTX-M-15 (291aa); CTX-M-14 (291aa); CTX-M-9 (291aa); SFO-1 (295aa)
<i>bla</i> _{IMP}	blaIMP-1	IMP-1 precursor + 8 chimeric sequences (245aa × 8)	IMP-1 precursor
<i>bla</i> _{TEM}	blaTEM; blaTEM-157; blaTEM-163; blaTEM-211	TEM-1 (286aa); + 43 chimeric and frag- mented sequences	TEM-1 (286aa)
<i>bla</i> _{KPC}	blaKPC2	KPC-2 (293aa); + 113 chimeric and frag- mented sequences	KPC-2 (293aa)

the putative gene sequences could be reconstructed even from the fragmented metagenomic assemblies; it does not matter whether the gene sequence is located within the single contig or scattered across several edges of an assembly graph.

Currently PATHRACER could be viewed as a standalone tool that might be integrated into analysis pipeline. Though we anticipate that deeper integration into assembler pipeline might be possible, e.g. one could utilize the information from paired-end reads to filter out more paths in the even graph, or, use pHMM alignment to supplement repeat resolution and scaffolding.

Acknowledgments

This work was supported by the Saint Petersburg State University (grant number 15.61.951.2015). The authors would like to extend a special thanks to Sergey Nurk and Tatiana Dvorkina for all the fruitful discussions that were of great help in improving the algorithms.

References

1. NCBIfam-AMR. <https://ftp.ncbi.nlm.nih.gov/hmm/NCBIfam-AMR/latest/>
2. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. *Journal of Molecular Biology* **215**(3), 403–10 (1990)

3. Chitsaz, H., Yee-Greenbaum, J.L., Tesler, G., et al: Efficient *de novo* assembly of single-cell bacterial genomes from short-read data sets. *Nat Biotechnol* **29**(10), 915–921 (Sep 2011)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: *Introduction to Algorithms* 2nd edition. MIT Press (2001)
5. Eddy, S.R.: Profile hidden Markov models. *Bioinformatics* **14**(9), 755–63 (1998)
6. Eddy, S.R.: Accelerated Profile HMM searches. *PLOS Computational Biology* **7**(10), 1–16 (10 2011)
7. Eppstein, D.: Finding the k shortest paths. *SIAM J. Comput.* **28**(2), 652–73 (Feb 1999)
8. Finn, R.D., Coghill, P., Eberhardt, R.Y., Eddy, S.R., et al: The Pfam protein families database: towards a more sustainable future. *Nucleic Acids Research* **44**(D1), D279–5 (2016)
9. Haider, B., Ahn, T.H., Bushnell, B., Chai, J., Copeland, A., Pan, C.: Omega: an overlap-graph *de novo* assembler for metagenomics. *Bioinformatics* **30**(19), 2717–22 (2014)
10. Lee, C., Grasso, C., Sharlow, M.F.: Multiple sequence alignment using partial order graphs. *Bioinformatics* **18**(3), 452–64 (2002)
11. Li, D., Huang, Y., Leung, C.M., Luo, R., Ting, H.F., Lam, T.W.: MegaGTA: a sensitive and accurate metagenomic gene-targeted assembler using iterative de Bruijn graphs. *BMC Bioinformatics* **18**(Suppl 12), 408 (Oct 2017)
12. Ng, C., Tay, M., Tan, B., Le, T.H., Haller, L., Chen, H., Koh, T.H., Barkham, T.M.S., Gin, K.Y.H.: Characterization of metagenomes in urban aquatic compartments reveals high prevalence of clinically relevant antibiotic resistance genes in wastewaters. *Front Microbiol* **8**, 2200 (Nov 2017)
13. Nurk, S., Meleshko, D., Korobeynikov, A., Pevzner, P.A.: metaSPAdes: a new versatile metagenomic assembler. *Genome Research* **27**(5), 824–34 (2017)
14. Poirel, L., Naas, T., Le Thomas, I., Karim, A., Bingen, E., Nordmann, P.: CTX-M-type extended-spectrum β -lactamase that hydrolyzes ceftazidime through a single amino acid substitution in the omega loop. *Antimicrobial Agents and Chemotherapy* **45**(12), 3355–3361 (2001)
15. Quast, C., Pruesse, E., Yilmaz, P., Gerken, J.e.a.: The SILVA ribosomal RNA gene database project: improved data processing and web-based tools. *Nucleic Acids Research* **41**(D1), D590–6 (2013)
16. Seemann, T.: Prokka: rapid prokaryotic genome annotation. *Bioinformatics* **30**(14), 2068–9 (2014)
17. Shakya, M., Quince, C., Campbell, J.H., Yang, Z.K., Schadt, C.W., Podar, M.: Comparative metagenomic and rRNA microbial diversity characterization using archaeal and bacterial synthetic communities. *Environmental Microbiology* **15**(6), 1882–99 (2013)
18. Viterbi, A.: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory* **13**(2), 260–9 (April 1967)
19. Wang, Q., Fish, J.A., Gilman, M., Sun, Y., Brown, C.T., Tiedje, J.M., Cole, J.R.: Xander: employing a novel method for efficient gene-targeted metagenomic assembly. *Microbiome* **3**, 32 (Aug 2015)
20. Wang, Q., Quensen, J.F., Fish, J.A., Kwon Lee, T., Sun, Y.e.a.: Ecological patterns of nifh genes in four terrestrial climatic zones explored with targeted metagenomics using framebot, a new informatics tool. *mBio* **4**(5) (2013)