CRISPR DETECTION FROM SHORT READS USING PARTIAL OVERLAP GRAPH

ILAN BEN-BASSAT AND BENNY CHOR

1. Supplementary Methods

S1. Complexity analysis

Notations: Let n be the number of ℓ-long reads in a dataset sampled from a bacterial genome G, with a coverage of c. Let K be the number of frequent k-mers in the data set, which are not part of a CRISPR repeat (also referred to as irrelevant frequent k-mers). For every frequent k-mer, u, in G, let F_u be the number of reads containing u. Let F be the total number of reads containing some frequent k-mer. We denote by m the maximum number of spacers in a CRISPR array of G, and by p the number of CRISPR arrays in G.

We also define several constants to be used in the analysis. Two of these express our current knowledge of the attributes of CRISPR loci: R is the maximum length of a CRISPR repeat, and S is the maximum length of a CRISPR spacer. In addition, we assume that k-mers from repeats are not frequent outside of CRISPR arrays. We also assume that sequences of at least σ bases (where σ is the minimum overlap length), that are a combination of a spacer subsequence and a repeat subsequence, are also not frequent in rest of the bacterial genome, G. We, therefore, denote by I the upper bound on the number of times that a k-mer, whose origin is either a repeat, or a spacer, or a consecutive combination of them, can appear in the genome outside of a CRISPR array. This bound also takes into account isolated copies of such a k-mer indside a CRISPR array (due to degenerated or truncated repeats).

For simplicity, we assume that different CRISPR arrays share no common repeats.

Lemma 1.1. *The time complexity of the algorithm is* $O(n\ell + Kc^2\ell + m^2c^2p\ell)$.

Proof. We analyze each of the stages of the algorithm in order to derive the overall time complexity.

Counting all k-mers can be done in $O(n\ell k)$ (on the average), using hash tables. Although not employed in our current implementation yet, we analyze the complexity with regard to rolling hash functions (such as Karp-Rabin fingerprint [2]). Such hash functions provide a better time complexity of $O(n\ell)$ for counting all k-mers (on the average).

Fetching the reads containing some frequent k-mer is done in $O(n\ell)$ (on the average). Identifying reads that contain the k-mer at the beginning can be done simultaneously.

The indexing step takes $O(F\ell)$ as we have F reads containing frequent k-mers. Each read is indexed in $O(\ell)$ time and so we get a total indexing time of $O(F\ell)$.

Bounding the time complexity of detecting frequent k-mers is more challenging. Since the time needed for analysing a single frequent k-mer, u, depends on whether u is part of a CRISPR repeat or not, we first analyse the time complexity of each of the two cases (given in claims 1.5 and 1.6), before proving that finding all frequent k-mers takes $O(Kc^2\ell + m^2c^2p\ell)$. We start by proving several simple claims regarding the coverage of the data set in certain regions.

Claim 1.2. *Each given copy of a k-mer, u, in the genome is contained in $O(c)$ reads (on the average).*

School of Computer Science, Tel-Aviv University. emails: ilanbb@gmail.com, benny@cs.tau.ac.il.

Proof. As proven in the paper, for every i , $1 \leq i \leq n$, the probability that the i-th read covers a certain copy of u is $\frac{c(\ell-k+1)}{n\ell} < \frac{c\ell}{n\ell} = \frac{c}{n}$. Therefore, the expected number of reads covering a given copy of u is $O(c)$. □

Claim 1.3. *Each given read, r, from a given location in the genome, is overlapped (to the right) by $O(c)$ reads from the same region of the genome (on the average).*

Proof. First, we note that the claim does not count overlaps with reads from other parts of the genome. The proof is similar to the one in claim 1.2. We only need to define the random variables χ_i as indicators for whether read i overlaps r (to the right), or not. The number of possible start positions for an overlapping read is $\ell - \sigma$, and the the proof continues as in claim 1.2. □

Using the former two claims, we can now address the upper bounds that are used for optimizing the process of constructing partial overlap graphs. The values of these bounds are clearly a matter of choice. However, given the assumption described in the beginning of this section, we can estimate the order of magnitude of these bounds, and set them accordingly in the algorithm.

Claim 1.4. *The number of isolated nodes that are sampled for each k-mer is bounded by $O(c)$. The number of sampled edges per node is bounded by $O(c)$, as well.*

Proof. Given a k-mer, u, that is part of a CRISPR repeat, the number of isolated reads that contain u equals the number of reads covering all the isolated copies of u, and hence it is $Ic = O(c)$ (by claim 1.2 and the definition of I). As a result, the number of sampled isolated nodes for each k-mer is bounded by $O(c)$. The proof for the bound on the number of sampled edges per node is similar (using claim 1.3 instead of claim 1.2).

□

The following claim holds for a k-mer, v, that is neither a part of a CRISPR array nor a part of some other spatial repeat patterns (such as tandem repeats). The complexity analysis of other types of non CRISPR related k-mers is somehow similar to the analysis of k-mers that are part of CRISPR repeats, which comes next.

Claim 1.5. *The expected time complexity for analysing a frequent k-mer v which does not appear in any CRISPR repeat is $O(c^2\ell)$.*

Proof. If v is not part of any special spatial repeat pattern as well, then almost no spacer edges are found when analysing it. In other words, almost all reads containing v correspond to isolated nodes. Therefore, for such a k-mer we sample $O(c)$ nodes and for each node we sample $O(c)$ edges (using claim 1.4). When analyzing a node, we compute the hash values of all suffixes of the corresponding read (which takes $O(\ell)$ time). Given an overlap of two reads, we can verify in time $O(\ell)$ that the k-mer does not fully appear in the overlapped part. We do not construct the entire set of overlaps at once, as we can take one overlap at a time, until the analysis of v is complete. The overall time complexity is $O(c(\ell + c\ell)) = O(c^2\ell)$. □

The next claim establishes the time complexity of analysing k-mers that are part of CRISPR repeats:

Claim 1.6. *The expected time for analysing a k-mer, u, which belongs to a CRISPR repeat is $O(m^2c^2\ell + F_u\ell)$.*

Proof. The time spent on analysing isolated node that contain a copy of u is $O(c^2\ell)$ (same as in claim 1.5). As for non isolated nodes, in the worse case we sample $O((m + I)c) = O(mc)$ such reads (using claim 1.2 and the fact that reads whose origin is outside a CRISPR locus are not necessarily isolated nodes). $O(c)$ edges are sampled for each read (node in the graph), and the time needed for computing hash values for the suffixes of a read is $O(\ell)$. In case of overlaps with reads that also contain u, we check the location of the k-mer with regard to the overlap itself and generate the spacer

sequences (all in $O(\ell)$ time). The spacer is potentially aligned to all previously detected spacers, which takes $O(mS^2)$ operations. So far, we have $O(mc(\ell + c\ell mS^2)) = O(m^2c^2\ell)$ operations for non isolated nodes. In addition, we compute an index, mapping the beginnings and ends of all spacers to the reads they appear in, which take $O(F_u\ell)$ operations. When checking adjacency of spacers, we might try up to all possible combinations of spacers detected, yielding another $O(m^2)$ operations (for each pair we perform a constant number of operations, using the mapping described). So, the total number of operations for a k-mer, u, that is part of a CRISPR repeat is

$$O(c^2\ell + m^2c^2\ell + F_u\ell + m^2) = O(m^2c^2\ell + F_u\ell)$$

□

The parameter K denotes the number of k-mers that are not part of a CRISPR repeat. The number of k-mers that are part of repeats is bounded by claim 1.7:

Claim 1.7. *The number of k-mers that are part of a CRISPR repeat is $O(p)$.*

Proof. There are p CRISPR arrays in the genome. Each of them contains one repeat sequence. A repeat is of maximum length R, so it contains at most $R - k + 1 = O(1)$ k-mers. □

We can now sum up the time complexity of analysing all frequent k-mers:

Claim 1.8. *The expected time for analysing all frequent k-mers is $O(Kc^2\ell + m^2c^2p\ell)$.*

Proof. By claim 1.5 we get that the time complexity is $O(Kc^2\ell)$ for all k-mers, v, that are not part of a CRISPR repeat. By claims 1.7 and 1.6 we get that for all k-mers, u, that are part of a CRISPR repeat, the overall time complexity is $O(\sum_{u \in repeat} m^2c^2\ell + F_u\ell) = O(pm^2c^2\ell + pF_u\ell)$. Since each such k-mer, u, appears in no more than $O((m+I)c)$ reads (assuming repeats are not shared between CRISPR arrays), we get that this time complexity equals $O(pm^2c^2\ell + pmcl) = O(m^2c^2p\ell)$. The overall time complexity is, therefore, $O(Kc^2\ell + m^2c^2p\ell)$. □

After establishing the time complexity of the frequent k-mer analysis stage, we proceed to the next stages.

The clustering stage starts with $O(p)$ k-mers to cluster (assuming no false positive repeats are reported), each having a list of $O(mc)$ reads containing it. At any round of the clustering process, we merge pairs of clusters whose intersection is big enough. A final cluster is obtained after a constant number of merges, since the number of k-mers in a repeat is considered constant. Since in each round we perform at least one merge for every final cluster, the number of rounds is constant as well. A round contains $O(p^2)$ operations of comparing clusters, each takes $O(mc)$ operations (assuming all clusters were first sorted in time $O(pmc\log(mc))$. Finally, we get $O(mcp^2 + pmc\log(mc))$. Knowing the range of possible values for each of the parameters p, m, c and ℓ, we can safely discard this expression from the total running time.

The consensus derivation step is not so time consuming and involves $O(mc)$ reads for each one of the p repeats. Setting the orientation of the k-mers and the reads in a cluster takes $O(mc)$ for a cluster, as there is a constant number of k-mers in a repeat. The efficient alignment process takes $O(mc\ell)$ operations per cluster, and therefore we get $O(pmc\ell)$. Note that since all reads contain the CRISPR repeat, the length of the multiple sequence alignment of all reads can not be too long.

Finally, the total time complexity is $O(n\ell + Kc^2\ell + m^2c^2p\ell)$. In most cases, the first expression is the most dominant one, so the time complexity is linear in the size of the data set.

□

Lemma 1.9. *The space complexity of the algorithm is $O\left(\frac{n\ell}{c} + \sum_u F_u + F\ell\right)$, for all frequent k-mers, u.*

Proof. Counting the number of k-mers requires storing all possible k-mers in the genome, whose number can be as large as the length of genome G. This is estimated by $n\ell/c$. When extracting all

frequent k-mers, we also maintain for each frequent k-mer, u, a list of indices of reads which contain u. This occupies $O(\sum_u F_u)$ space. In addition, we store an index of hash values of all long-enough prefixes of reads containing frequent k-mers, which takes $O(F\ell)$ space. □

For most genomes and data sets, the space complexity can be given by $O(\frac{n\ell}{c})$.

References

[1] Kececioglu J. D., Myers E. W. (1995). Combinatiorial algorithms for DNA sequence assembly. Algorithmica, 13:751
[2] Karp R.M., Rabin M.O. (1987) Efficient randomized pattern-matching algorithms. IBM Journal of Research and Development, 31(2): 249–260.

2. Supplementary Figures

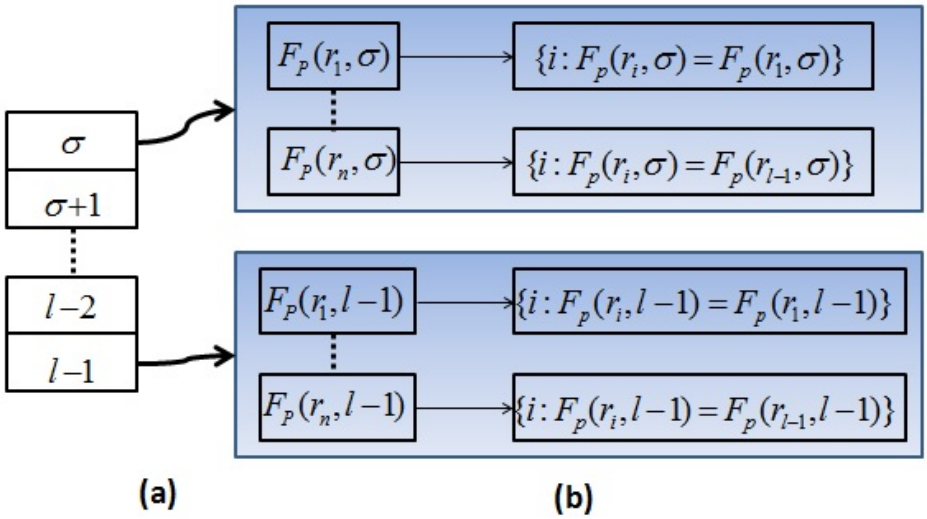

Figure S1. The index of hash values: (a) A mapping between a proper prefix length to data on prefixes of the same length. (b) A secondary mapping between a hash value and a list of read indices which have a prefix of a certain length with such a hash value.

3. Supplementary Tables

	Counting	Refinement	Indexing	Repeats	Non repeats	Others	**Total**
Bacteroides fragilis YCH46	86	50	11	1	18	3	**169**
Ammonifex degensii KC4	33	20	29	18	35	11	**146**
Clostridium difficile R20291	73	63	19	5	55	5	**221**
Acidilobus saccharovorans	25	22	2	1	1	3	**54**
Ferroglobus placidus	36	33	4	1	11	2	**87**
Thermus thermophilus HB8	4	4	2	9	2	1	**22**
Acinetobacter sp. ADP1	62	56	8	6	30	5	**167**
Helicobacter pylori B8	27	25	1	0	1	2	**56**
Magnetospirillum magneticum	86	75	15	0	33	8	**217**
Natrialba magadii	64	57	3	1	9	3	**137**
Oligotropha carboxidovorans OM5	64	59	4	0	1	1	**129**
Sulfolobus solfataricus P2	48	43	63	9	117	16	**296**
Tsukamurella paurometabola	73	66	9	0	34	6	**188**
Francisella philomiragia	33	30	3	1	9	1	**77**
Pelobacter propionicus	3	3	0	1	0	0	**7**
Porphyromonas gingivalis W83	40	34	24	3	56	6	**163**
Streptococcus agalactiae A909	36	32	9	1	27	4	**109**

Table S1. Time performance of running the algorithm on data sets of simulated reads. Time is measured in seconds for the algorithm itself, as well as for every major step of it: counting k-mers (Counting), retrieving frequent k-mers and corresponding reads (Refinement), computing hash values of all relevant prefixes (Indexing), analysing frequent k-mers, both those which are part of a CRISPR repeat (Repeats) and those which are not (Non repeats), and other stages, such as the clustering and orientation of k-mers and reads (Others).

	Genome Size (Mbp)	No. of Reads (M)	**Memory peak (GB)**
Bacteroides fragilis YCH46	5.28	1.38	**1.9**
Ammonifex degensii KC4	1.80	0.55	**1.3**
Clostridium difficile R20291	4.19	1.10	**1.6**
Acidilobus saccharovorans	1.50	0.39	**0.6**
Ferroglobus placidus	2.20	0.58	**0.9**
Thermus thermophilus HB8	0.26	0.07	**0.1**
Acinetobacter sp. ADP1	3.60	0.94	**1.4**
Helicobacter pylori B8	1.67	0.44	**0.7**
Magnetospirillum magneticum	4.97	1.30	**1.8**
Natrialba magadii	3.76	0.99	**1.5**
Oligotropha carboxidovorans OM5	3.76	0.98	**1.5**
Sulfolobus solfataricus P2	2.99	0.77	**2.7**
Tsukamurella paurometabola	4.38	1.15	**1.6**
Francisella philomiragia	2.05	0.53	**0.8**
Pelobacter propionicus	0.20	0.05	**0.1**
Porphyromonas gingivalis W83	2.34	0.61	**1.1**
Streptococcus agalactiae A909	2.13	0.56	**0.8**

Table S2. Memory peaks in GB when running the algorithm on data sets of simulated reads. The size of the genomes and number of reads in each data set are listed as well.

4. Supplementary Code

Algorithm S1 *FindOverlaps* - find all overlaps of a read r, where r is extended to the right

Input: r - a read, *prefixes* - index of prefixes data, τ - minimum overlap
Output: *overlaps* - all overlaps where r is extended to the right

```
1: suffixes ← compute hash values of all relevant suffixes of r
2: ℓ ← |r|
3: for all len in range τ .. ℓ do
4:    suffixValue ← the value computed for the suffix of length len of r
5:    if suffixValue in prefixes[len] then
6:       for all otherRead in prefixes[len][suffixValue] do
7:          overlaps ← overlaps ∪ {otherRead}
8:       end for
9:    end if
10: end for
11: return overlaps
```

Algorithm S2 Partial construction of the induced subgraph G_u. For the sake of clarity, we do not address reverse complement issues in the pseudo-code. It assumes the existence of the following methods: (1) *FindOverlaps* as described in algorithm S1. (2) *ExtractEdgeSequence* which extracts the edge sequence of a spacer edge. (3) *ComputeOffsets* computes the offsets of the spacers from the two sides of the edge sequences. (4) *IndexReadsByOffsets* which indexes all reads using the computed offsets. (5) *IsAdjacent* checks whether two given spacer-edge sequences are consecutive or not.

Input: u - A k-mer to analyse, R_u - Set of reads containing u, *Index* - Index of hash values of prefixes of reads in $\mathcal{R}^*$.
Output: Whether u is part of a CRISPR repeat or not

```
1: starters ← all reads in R_u in which u appears at the beginning
2: sequences ← ∅
3: matches ← 0
4: readCache ← ∅
5: isolated ← 0
6: MS ← minimum number of spacers in a CRISPR array
7: for all starter in starters do
8:    overlappers ← FindOverlaps(starter, Index, τ)
9:    for all overlapper in a subset sampled from overlappers do
10:      if u not in the overlap of starter and overlapper then
11:         edgeSeq ← ExtractEdgeSequence(starter, overlapper, u)
12:         if edgeSeq not similar to any element in sequences then
13:            sequences ← sequences ∪ {edgeSeq}
14:            if |sequences| == MS then
15:               offsets ← ComputeOffsets(sequences)
16:               readCache ← IndexReadsByOffsets (R*, u, offsets)
17:               for all pair (first, second) in sequences do
18:                  matches ← matches + IsAdjacent(first, second, readCache, offsets)
19:               end for
20:            else if |sequences| > MS then
21:               for all seq in sequences do
22:                  matches ← matches + IsAdjacent(edgeSeq, seq, readCache, offsets)
23:               end for
24:            end if
25:            if matches > minimum number of spacers in a CRISPR array then
26:               return True
27:            end if
28:         end if
29:      end if
30:   end for
31:   if overlapper is an isolated node then
32:      isolated ← isolated + 1;
33:   end if
34:   if too many isolated nodes discovered then
35:      return False
36:   end if
37: end for
```

Algorithm S3 Determine the orientation of every k-mer in K and every read that contains a k-mer in K. The two different strands are marked by PRIMARY and REVCOMP.

Input: K - set of k-mers to orient, M - mapping between k-mers and reads that contain them.
Output: *kmersOrientations* - mapping of every k-mer in K to a strand.
readsOrientations - mapping of every read containing a k-mer in K to a strand.

kmersOrientations ← ∅
readsOrientations ← ∅
futureOrientations ← ∅
while some k-mers were oriented in the previous round (or if first round) **do**
  for all *kmer* in K **do**
    rckmer ← reverse complement of *kmer*
    if *rckmer* and *rckmer* are the first to be oriented **then**
      kmersOrientations[kmer] ← PRIMARY
      kmersOrientations[rckmer] ← REVCOMP
    else if *kmer* and *rckmer* in *futureOrientations* and not already oriented **then**
      kmersOrientations[kmer] ← *futureOrientations*[*kmer*]
      kmersOrientations[rckmer] ← *futureOrientations*[*rckmer*]
      Verify no contradiction in the orientation of *kmer* and *rckmer*
    end if
    if *rckmer* and *rckmer* were oriented in this round **then**
      kmerReads ← M[*kmer*]
      rckmerReads ← M[*rckmer*]
      Verify no read contains both *kmer* and *rckmer*
      Verify no contradiction in the orientation of reads in *kmerReads* and *rckmerReads*
      for all *read* in *kmerReads* **do**
        readsOrientations[*read*] ← *kmersOrientations[kmer]*
        for all kmer k in *read* **do**
          futureOrientations[k]← *kmersOrientations[kmer]*
        end for
      end for
      for all *rcread* in *rckmerReads* **do**
        readsOrientations[*rcread*] ← *kmersOrientations[rckmer]*
        for all kmer k in *rcread* **do**
          futureOrientations[k]← *kmersOrientations[rckmer]*
        end for
      end for
    end if
  end for
end while
return (*kmersOrientations*, *readsOrientations*)