Amplicon Analysis I

Eduardo Pareja-Tobes, Raquel Tobes

Oh no sequences! research group, Era7 Bioinformatics

Plaza de Campo Verde, 3, 18001 Granada, Spain

September 16, 2019

Abstract

Here we present

1. a model for amplicon sequencing
2. a definition of the best assignment of a read to a set of reference sequences
3. strategies and structures for indexing reference sequences and computing the best assignments of a set of reads efficiently, based on (ultra)metric spaces and their geometry

The models, techniques, and ideas are particularly relevant to scenarios akin to 16S taxonomic profiling, where both the number of reference sequences and the read diversity is considerable.

Keywords *bioinformatics, NGS, DNA sequencing, DNA sequence analysis, amplicons, ultrametric spaces, indexing.*

Corresponding author Eduardo Pareja-Tobes eparejatobes@ohnosequences.com

1 Reads, Amplicons, References, Assignments

In this section we will define what we understand as reads, amplicons, references, and the best assignment of a read with respect to a set of references.

In our model the reads are a product of the probability distribution for the 4 bases (A, C, G and T) in each position of the sequence.

Definition 1.0.1 *Read.* A read R of length n is a product $\prod_{i=0}^{n-1} R_i$ of n probability distributions on $\Sigma = \{A, T, C, G\}$. We will denote the length of a read as $|R|$.

The corresponding probability mass function is thus defined on the space Σ^n of sequences in Σ of length n, the *domain* of R. The probability that R assigns to a sequence $x = (x_i)_{i=0}^{n-1} : \Sigma^n$ is

$$R(x) = \prod_{i=0}^{n-1} R_i(x_i)$$

Remark 1.0.2. The concrete finite alphabet Σ is of course irrelevant in what follows; we will assume it being nucleotides as it is the one we find in DNA sequencing. The use of extended alphabets such as IUPAC codes covering combinations of bases is rendered superfluous by working with the read as a distribution, which can be evaluated in any $U \subseteq \Sigma^n$.

Defined as products of single-base distributions reas have positions both well-defined and independent. From the standard viewpoint of sequencing errors, read distribution positions are well defined if and only if indel errors are deemed impossible/highly unlikely; Their presence thus would force us to simply throw away all per-position quality information: we can no longer ascribe scores to positions which are simply not well-defined. Fortunately, indel errors in Illumina sequencers[1] have been considered as quite exceptional: in [11] it is reported that indel errors occur at rates of at most $5 * 10^{-6}$ per base.

All DNA sequencers output `fastq` : (sets of) sequences (x_i, q_i) where the quality score q_i encodes the probability of the base x_i being correct[2]. In the now standard Phred encoding a quality score $q : \mathbb{N}$ represents a probability of *error* $e = 10^{\frac{-q}{10}}$.

A generic read R is determined by $|R||\Sigma|$ values p_i^j in $[0, 1]$ subject to the n equations $\sum_j p_i^j = 1$. A Phred-encoded `fastq` read $(x_i, q_i)_{i=0}^{n-1}$ though orresponds to a read $\prod_{i=0}^{n-1} D_i$ where the distribution D_i assigns probability $1 - e_i$ to x_i and $\frac{e_i}{3}$ to any of the other 3 bases; $|R|$ values and a sequence determine

[1] the dominant platform by far, even more so in amplicon sequencing.

[2] Correct here means that it matches what was ultimately put into the sequencer. Modifications introduced during the library preparation process, contamination –whatever happens *before* entering the sequencer– cannot be accounted for by quality scores.

	A	C	A	T	G	T	T	A	C	G
p(A)	0.70	0.07	0.90	0.10	0.03	0.13	0.10	0.50	0.10	0.03
p(C)	0.10	0.80	0.03	0.10	0.03	0.13	0.10	0.17	0.70	0.03
p(G)	0.10	0.07	0.03	0.10	0.90	0.13	0.10	0.17	0.10	0.90
p(T)	0.10	0.07	0.03	0.70	0.03	0.60	0.70	0.17	0.10	0.03

Figure 1: **Read distribution derived from fastq**

fastq reads. In what follows by a `fastq` read we will refer to a read for which there is a unique most likely base at each position, and the probabilities of any erroneous base -those different from the most likely one are all equal (see figure 1).

Inside the sequencer all erroneous bases are not equally likely; `fastq` implies a considerable loss of information. But even putting that aside, it can be important in practice to allow for different error probabilities. For example, merging of the distributions corresponding to reads sequenced from the same molecule[3] becomes non-associative (and erroneous).

1.1 Amplicons

Amplicon sequencing in the Illumina platform normally produces two reads[4], corresponding to subsequences of the target. Our model covers any type of amplicon, with overlapping or not overlapping reads, single or paired.

1.1.1 Extracting Amplicon-specific Fragments from References

An amplicon, loosely defined, corresponds to a set of primer pairs which would yield a pair of sequenced reads (R1 and R2) from each target in the sample. These reads, viewed as products of probability distributions on Σ, are isomorphic to a single read of length the sum of the amplicon reads lengths. If, as is normally the case, we have a collection of known complete target sequences[5] we simply need to produce the corresponding amplicon *in-silico*: extract the subsequences that would correspond to the sequenced fragments if the reference would be included in the sample. This will be further translated to the/an amplicon read along the permutation $((n_1), \cdots, (n_k)) \longrightarrow (\sum n_i)$.

[3] such as what can be assumed to happen when using UMIs or equivalent mechanisms.
[4] Illumina single read sequencing, while possible, is not the standard.
[5] the whole 16S gene in amplicons targetting hypervariable regions, for example.

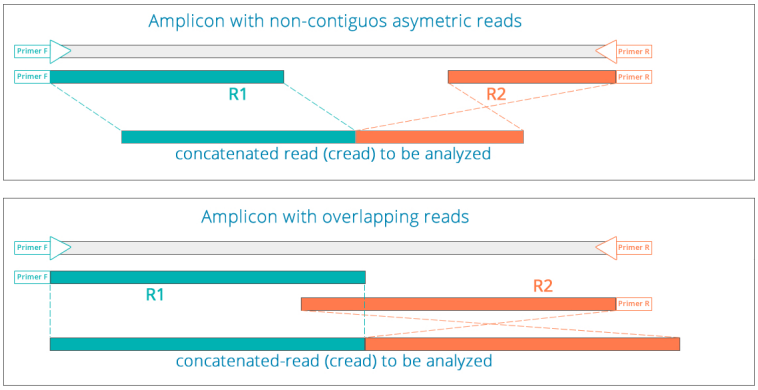

Figure 2: **Non-overlapping vs overlapping amplicons**

Paired-end amplicons whose reads are expected to overlap are analyzed by first merging the reads at those positions. We argue that the suposedly overlapping positions should be analyzed independently, as we do.

1. Amplicons normally have variable insert lengths, meaning that there is no *sequence-independent* way of determining which pairs of positions are supposed to come from the same nucleotide.
2. From the point of view of probability distributions this merging is commonly done wrong. Even in [8] the result is only correct with respect to the most likely base.

If there are positions which actually come from the same nucleotide, these are given more weigth than the rest in the most likely reference. If this is not what we want, we should simply be sequencing amplicons whose reads don't overlap[6]. In short, overlapping amplicon designs at the very best throw away sequence.

The reference sequences should be determined in terms of predicted primer binding sites, simulating sequencing *in silico*. Of course, as in any amplicon, the positions at which primers bind the target must be discarded; primers can and will bind even if a few bases differ, and what is sequenced is the primer sequence, not the target sequence.

If the amplicon sequenced fragments are non-contiguous we simply need to extract the sequences that would be sequenced starting after each primer binding site. All these bases form a sequence on which we can evaluate our read qua product of distributions; that they could be separated by some other sequence or overlap in the target is irrelevant at this stage.

A reference thus is just a sequence which we expect can be obtained by sequencing an amplicon.

[6] or single read ones, for that matter.

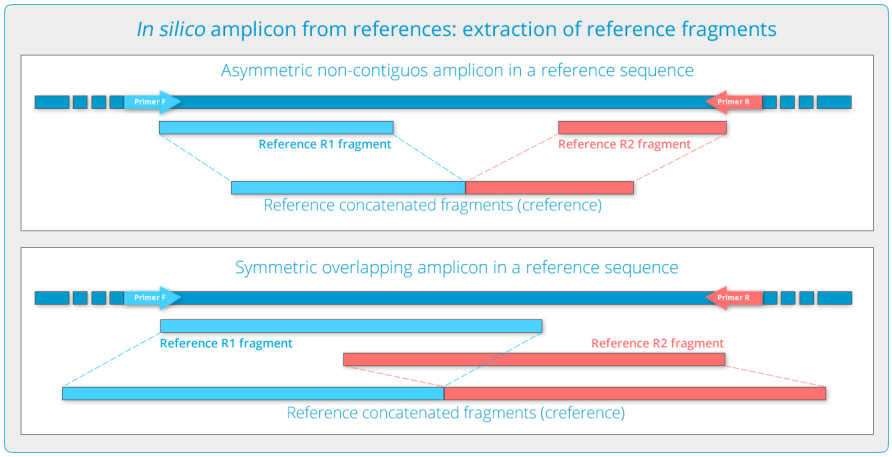

Figure 3: **Reference extraction from target sequences**

1.2 Alignment and Assignment

Definition 1.2.1 *Read Assignment.* Given a read R and a set $\mathcal{S}$ of sequences in the domain of R, the *assignment* of R in $\mathcal{S}$ is the subset $A(R, \mathcal{S}) \subseteq \mathcal{S}$ at which R attains maximum probability.

$\mathcal{S}$ above is normally a proper subset of $\Sigma^{|R|}$ the domain of R; if not, computing the corresponding assignment trivializes: it simply corresponds to all possible combinations of maximums at each position (recall that R is assumed to be a product of independent distributions).

An amplicon assignment algorithm is thus a process which has as input a set of reads $\mathcal{R}$, and a set $\mathcal{S}$ of reference sequences, where reads and sequences are assumed to have the same length n.

2 Assignment Strategies

For ease of exposition we will assume throughout that R has *one* most likely sequence m_R; the modifications are minimal when that is not the case. Also, as is normally done, we will work with logarithms of probabilities. These we will call *scores*, so that our goal will be determining the best scoring references in $\mathcal{S}$.

If $m_R \in \mathcal{S}$ then trivially $A(R, \mathcal{S}) = \{m_R\}$. But, even if it's not, m_R could be useful as a probe: intuitively $A(R, \mathcal{S})$ should be "close" to m_R, in some sense. We can make this idea precise through the introduction of a distance $(\Sigma^{|R|}, d)$ for which the differences between scores $R(x), R(y)$ can be bounded in terms of $d(x, y)$ and R. For this distance to be useful it should be faster to compute when compared with computing scores under R, of course. It turns out that $d(x, y)$ the number of mismatches between two sequences $x, y : \Sigma^{|R|}$ satisfies both needs.

2.1 Metric Bounds

Let's see how the score of a reference x under R can be bounded both above and below in terms of $d(m_R, x)$.

Definition 2.1.1 *Worst, Best, Second Best Scores.* For R a read let

$$B_R, BE_R, WE_R : [0, |R|[\subset \mathbb{N} \longrightarrow \mathbb{R}$$

so that

- $B_R(i)$ is the best score of R_i
- $BE_R(i)$ is the second best score of R_i (the best error)
- $WE_R(i)$ is the worst score of R_i (the worst error)

Example 2.1.2.

These functions just defined can be used to determine bounds for R on balls $B(m_R, k)$ and co-balls $F(m_R, k)$ with center m_R. If $d(m_R, x) \leq k$ at worst x has k mismatches with m_R. The worst score we could possibly obtain with k mismatches is the sum of the $n - k$ worst correct scores and the worst k errors. Dually, when $d(m_R, x) \geq k$ at best x has again k mismatches with m_R, and there could be no better way of picking them than (if possible) the best $n - k$ correct scores and the k second best scores. We have just proved

Proposition 2.1.3. *For R a read of length n*

$$\begin{aligned} R|_{B(m_R,k)} &\leq U(k) \\ R|_{F(m_R,k)} &\geq L(k) \end{aligned} \tag{1}$$

where

$$\begin{aligned} U(k) &= \sum \max_{n-k} B_R + \sum \max_k WE_R \\ L(k) &= \sum \min_{n-k} B_R + \sum \min_k BE_R \end{aligned} \tag{2}$$

Remark 2.1.4. Note that for a generic R these bounds (taking as domain of R all possible reference sequences) are not tight; the worst correct score and the worst error (for example) could correspond to the same R_i, thus being impossible to attain. They are tight though when R is of `fastq` type, as is easily seen by induction on the length of the sequence.

Definition 2.1.5. Let $\lambda_R(k) \equiv \lambda(k)$ be the minimum $k : \mathbb{N}$ at which $L(\lambda(k)) > U(k)$.

Trivially $k < \lambda(k)$, and once we have found $x : \mathcal{S}$ with $d(m_R, x) = k$ our bounds imply that it is strictly better than any $x' : \mathcal{S}$ for which $d(m_R, x') \geq \lambda(k)$:

Proposition 2.1.6. *Let x be a reference, $k = d(m_R, x)$. Then the assignment is contained in $B(m_R, \lambda(k))$.*

Proof. As by hypothesis the open ball is nonempty, it is enough to prove that $R(x) < R_{|F(m_R,\lambda(k))}$. But from the definition of $\lambda(k)$ and 2.1.3

$$R(x) \leq U(k) < L(\lambda(k)) \leq R_{|F(m_R,\lambda(k))}$$

□

Computationally this result is as advantageous as calculating $d(m_R, x)$ is with respect to evaluating $R(x)$. But we are in a metric space, and the triangular inequality gives us (once we compute a distance $d(m_R, x)$) a ball containing the assignment defined *only* in terms of the reference set and the read:

Corollary 2.1.7. *Under the hypotheses of 2.1.6 the assignment is contained in the ball $B(x, k + \lambda(k))$.*

Proof. Obvious (triangular inequality). □

These results restrict the domain of our search once we compute a distance $d(m_R, x)$. But the U, L bounds can also be used to discard whole balls once we have a x' far enough from m_X relative to a previously inspected x.

Proposition 2.1.8. *Let $x : \mathcal{S}, k = d(m, x)$. Then for any $x' : \mathcal{S}$ such that $k' = d(m, x') > \lambda(k)$*

$$R_{|B(x',k'-\lambda(k))} > R(x)$$

Proof. The ball $B(x', k' - \lambda(k))$ is contained in $F(m, \lambda(k))$ and thus 2.1.6 applies. □

All this already points to a possible index structure: the balls $B(x, k + \lambda(k)), B(x', k' - \lambda(k))$ are all intrinsic to the reference set, and $\lambda(k)$ is easy to compute[7]. The complexity of the resulting structure looks daunting though. Checking containment for two balls $B(x, k), B(x', k')$ potentially requires computing all the relevant distances, the intersection of two balls is not a ball that would require *something* more than mere balls.

What we do instead is freely build a new metric space out of $\mathcal{S} \subset (\Sigma^n, d)$ on which the balls are as easy to manage as one could possibly hope.

[7] For `fastq` reads we normally have $\lambda(k) = k + 1$.

2.2 Free Ultrametrics as Indexes

Definition 2.2.1 *Ultrametric Space.* An ultrametric space is a metric space (X, d) for which

$$d(x, z) \leq \max\{d(x, y), d(y, z)\}$$

See [15, 12, 6, 14, 9, 7] for references on ultrametric spaces. What interests us about them is the particularly simple structure of the poset of balls:

Proposition 2.2.2. *In an ultrametric space*

- $x' \in B(x, \lambda) \implies B(x, \lambda) = B(x', \lambda)$
- $B(x, \lambda) \cap B(x', \lambda')$ *is either empty or the one with minimum radius*

In particular, two balls are either disjoint or one is contained in the other. It follows that for a fixed λ the balls $B(x, \lambda)$ are equivalence classes for the relation $x \sim y$ whenever $d(x, y) \leq \lambda$: transitivity follows from 1. above. The poset of balls is thus a tree where leaves are the points.

From now on we will refer to balls in an ultrametric space as *clusters*, and denote them as $C(x, \lambda)$.

Definition 2.2.3 *Lower/Upper Radius, Point Radius.* The lower radius of a cluster $C(x, \lambda)$ is the minimum λ' for which $C(x, \lambda') = C(x, \lambda')$; the upper radius is the maximum onesuch. The point radius of $x : (X, d)$ in an ultrametric space is $p(x) = \sup\{\lambda : C(x, \lambda) = \{x\}\}$.

Definition 2.2.4 *Cluster-compatible Order.* A cluster-compatible order is any *linear* order $\leq$ on the points of (X, d) for which every cluster is an interval:

$$C(x, \lambda) = [s(x, \lambda), e(x, \lambda)[$$

and for which cluster intersection corresponds to interval intersection

$$C(x, \lambda) \cap C(x', \lambda') = [\max(s(x, \lambda), s(x', \lambda')), \min(e(x, \lambda), e(x, \lambda'))[$$

Proposition 2.2.5. *Any ultrametric space admits a cluster-compatible order.*

Proof. There's a maximum radius λ_0 at which $X \neq C(x, \lambda_0)$. Pick a linear order on the clusters $\{C(x, \lambda_0) : x : X\}$, and proceed recursively on each. Points correspond to the clusters $C(x, p(\lambda))$, which get ordered linearly; it's easy to see that each cluster corresponds to an interval in the resulting order. □

Note that in the proof above we can pick *any* linear order on clusters of the same radius.

We can then store *all* balls in an ultrametric space as intervals, and thus store points in any linear data structure (arrays) so that points get closer as we reduce intervals centered at any point. How many of such intervals, though? It turns out that an ultrametric space X, d has at most $2|X|$ different clusters.

Proposition 2.2.6. *For (X, d) an ultrametric space the image of d has cardinality at most $|X| - 1$.*

Proof. Induction on $|X|$. □

Proposition 2.2.7. *[13] There are at most $2|X|$ balls in an ultrametric space.*

Proof. Again induction on $|X|$. □

2.2.1 Free Ultrametric Spaces

We have just seen that the structure of balls in an ultrametric space is particularly simple:

1. There are at most $2|X|$ different clusters
2. The inclusion relation is a tree order
3. This order admits linear order extensions under which clusters are intervals in an intersection-preserving way

Being ultrametric though can be seen as a *property* that a metric space might satisfy –or not, as is the case of (Σ^n, d_H). What we can try to construct is the best possible generic approximation of a metric space by an ultrametric one: a *left* adjoint $C : \mathbf{Met} \longrightarrow \mathbf{UMet}$ to the inclusion $\mathbf{UMet} \longrightarrow \mathbf{Met}$. The unit $\eta : Id \longrightarrow CU(X)$ yields at each metric space (X, d) a morphism in $\mathbf{Met}$ $\eta_X : X \longrightarrow C(X)$.

Proposition 2.2.8. *The free ultrametric space exists and is given by the same set of points and distance*

$$d_{CX}(x, y) = \inf \left\{ \max \{ d_X(x_i, x_{i+1}) : (x = x_0, \dots, x_n = y) \} \right\}$$

Proof. C as defined is clearly a functor, the identity on points the components of a natural transformation $Id \longrightarrow UC$, and the unit-counit equations are trivial to verify. There is also a totally straightforward proof establishing the standard bijection between morphisms $CX \longrightarrow L$ and $X \longrightarrow L$. □

As the right adjoint is fully faithful $C(X) \cong X$ for any ultrametric space X. By construction $\eta_X : X \longrightarrow CX$ is essentially surjective on objects; it is further guaranteed to be injective on objects in our main case of interest:

Proposition 2.2.9. *If (X, d) is a finite metric space then the unit at X: $\eta_X : X \longrightarrow CX$ is the identity on objects.*

Proof. Straightforward (every point is isolated in a finite metric space). □

Remark 2.2.10. Having a morphism $\eta \equiv \eta_X : X \longrightarrow CX$ implies that $d(x, y) \geq d_{CX}(\eta(x), \eta(y))$. If η is bijective on objects we can think of d_{CX} as a distance on the same set points which is allowed to get points closer, but not set them farther than they were. As a consequence

Lemma 2.2.11. $B(x, \lambda) \subseteq C(x, \lambda)$.

2.2.2 Clusters and Assignments

We wanted to use clusters in $C(\mathcal{S})$ as a sort of substitute for balls in $\mathcal{S}$; let's see how what we know about bounding and discarding references for assignment relates with these clusters.

Just for future reference let's record the obvious

Proposition 2.2.12. *Let $x : \mathcal{S}, k = d(m, x)$.*

Then the assignment is in any cluster $C(x, k') \subseteq B(x, k + \lambda(k))$.

Proposition 2.2.13. *Let $x, x' : \mathcal{S}$ such that $k' = d(m, x') > \lambda(k)$.*

Then we can discard any cluster $C(x', k'') \subseteq B(x, k' - \lambda(k))$.

With respect to bounding 2.2.11 and 2.1.7 yield

Corollary 2.2.14. *Let x be a reference, $k = d(m, x)$. Then the assignment is contained in $C(x, k + \lambda(k))$.*

For discarding though we don't have any generic way of determining when $C(x', \lambda') \subseteq B(x, \lambda)$.

2.2.3 Radii

We will stop now and look at different radii on subsets of metric spaces; these and derived notions will be used to bound/discard clusters later on.

Definition 2.2.15 *inner/outer radii.* Let $x : (X, d)$. The *inner, outer radii at x* are the maps $i_x, o_x : P(X) \longrightarrow R$ defined by

$$\begin{aligned} i_x(U) &= \sup\{\, \lambda : B(x, \lambda) \subseteq U \,\} \\ o_x(U) &= \inf\{\, \lambda : U \subseteq B(x, \lambda) \,\} \end{aligned} \tag{3}$$

When allowing any basepoint we have the inner, outer radii $i, r : P(X) \longrightarrow R$

$$\begin{aligned} i(U) &= \sup\{\, i_x(U) : x : U \,\} \\ o(U) &= \inf\{\, o_x(U) : x : U \,\} \end{aligned} \tag{4}$$

An (inner/outer) center is a point $x : U$ at which $i(U) = i_x(U)$ respectively $o(U) = o_x(U)$.

Proposition 2.2.16. *Let $U : P(X)$ a subset of the metric space X, $x : U$. Then*

$$B(x, \lambda) \subseteq U \iff \lambda \leq i_x U \tag{5}$$
$$U \subseteq B(x, \lambda) \iff o_x U \leq \lambda \tag{6}$$

Proof. Trivial. □

Definition 2.2.17 *Entometer, Diameter.* The entometer, diameter $\operatorname{entom}(U), \operatorname{diam}(U)$ of a subset $U \subseteq (X, d)$ are

$$\begin{aligned} \operatorname{entom}(U) &= \inf\{\, i_x(U) : x : U \,\} \\ \operatorname{diam}(U) &= \sup\{\, o_x(U) : x : U \,\} \end{aligned} \tag{7}$$

Remark 2.2.18. $\operatorname{entom}(U)$ is the minimum distance between U and its complementary U^*, which implies that $\operatorname{entom}(U) = \operatorname{entom}(U^*)$.

Proposition 2.2.19. *Let $U : P(X)$ be a subset of the metric space X. Then*

$$\forall x : U \quad B(x, \lambda) \subseteq U \iff \lambda \leq \operatorname{entom} A \tag{8}$$
$$\forall x : U \quad U \subseteq B(x, \lambda) \iff \operatorname{diam} A \leq \lambda \tag{9}$$

Proof. Trivial. □

For reference

Lemma 2.2.20. *Let $U : P(X), x : U$. Then*

$$\operatorname{entom} U \leq i_x U \leq iU \tag{10}$$
$$oU \leq o_x U \leq \operatorname{diam} U \tag{11}$$

Proof. Trivial. □

Remark 2.2.21. Somewhat contrary to intuition, there is no generic relationship between inner and outer radii, or entometers and diameters. It is easy to find examples where $i_x U \leq o_x U$ and $o_{x'} U \leq i_{x'} U$. We do have though

Lemma 2.2.22. $\frac{1}{2} \operatorname{diam} U \leq o(U) \leq \operatorname{diam} U$

2.2.4 Cluster Radii, Bounding, Discarding

Let's see what we can say about the radii of clusters when seen as subsets of the original space (X, d).

Lemma 2.2.23. *Let λ be the upper radius of a cluster $C = C(x, \lambda)$. Then as a subset of the original metric space* entom $C = \lambda$.

Proof. Quasi-obvious. □

In particular for a cluster $\lambda \leq \operatorname{entom} C(x, \lambda)$. What about the inner radius? For any point $x' : C = C(x, \lambda)$ we have that $\lambda \leq i_{x'} C$ so we can only say that $\lambda \leq iC$.

In the context of our problem the inner radius at a point can be used to determine clusters which are known to contain the assignment:

Proposition 2.2.24. *Let x be a reference, $k = d(m, x)$. Then the assignment is contained in any cluster $C(x, k')$ for which $k + \lambda(k) \leq i_x(C(x, k'))$.*

Obviously, once we have computed $d(m, x)$, bounding through the inner radius at[8] x will consist in determining the *smallest* cluster C containing x for which $k + \lambda(k) \leq i_x C$.

The entometer plays the same role, with the difference that there is no dependence on the particular reference, only the clusters containing it:

Proposition 2.2.25. *Let x be a reference, $k = d(m, x)$. Then the assignment is contained in any cluster C containing x for which $k + \lambda(k) \leq$ entom C.*

As remarked before, we will always refer to the *smallest* such cluster when bounding by the entometer.

The situation when discarding clusters is of course dual. In the standard context for discarding a reference we have

Proposition 2.2.26. *Under the hypotheses of 2.2.13 we can discard any cluster C containing x' for which $o_{x'} C \leq k' - \lambda(k)$.*

And here when discarding by the outer radius at x' we will always find the *biggest* cluster satisfying the above.

The version without reference on the cluster side involves of course the diameter:

Proposition 2.2.27. *Under the hypotheses of 2.2.13 we can discard any cluster C containing x' for which* diam $C \leq k' - \lambda(k)$.

[8] Global inner radii and the corresponding centers are not immediately useful here.

2.3 Probability Thresholds

In most cases we would like to establish a lower? bound s_0 for assignment probabilities, treating as unassigned the resulting reads. For this we can first determine a minimum distance from m_R threshold K for which $d(m_R, x) > K \implies R(x) > s_0$. As in 2.2.26 we can discard whole clusters in terms of their outer radius

Proposition 2.3.1. *If $d(m_R, x) = k > M$ where M is some minimum distance threshold, we can discard any cluster $C(x, k)$ for which*

$$o_x(C(x, k')) \leq k - M$$

3 Implementation Strategies

3.1 Indexing

Given (X, d) a metric space, the set of all clusters (balls in the free ultrametric space), possibly aumented with further data at each cluster/point, will be called from now on an *ultrametric index* on X. The sort of extra data we will consider consists in inner/outer radii and the corresponding centers, and entometers/diameters. As with any index, there's a speed/space tradeoff; hardware limitations and any generic information about the sort of input this index will be queried with should guide the decision on which extra data the index should contain.

From what we saw in the previous sections, inner radii and entometers can be used to *bound* the search to a cluster, while outer radii and diameters to *discard* clusters. If we expect our queries to be diverse (each one close to far away reference points) we will spend much more time discarding than bounding our search; we could then add outer centers and their radii to all clusters, or simply entometers (just one number per cluster).

As for the points themselves, different storage layouts are available, each one more suitable to a certain sort of queries. For example, using any cluster-compatible order, the points themselves can be stored in a flat array, while clusters are sets of nested intervals over this array. These intervals can be stored using variants of segment trees, or using an Elias-Fano encoding of the start and end points. Another possibility lies in simply keeping the list of nested clusters at each point, and then compress those lists using incremental compression. These sort of structures will perform really good if the whole index can be fit in memory.

A different layout consists in storing the tree of clusters (and any extra data) directly, using a level-wise data structure. The whole space cluster is stored at the root, and then at the next level we have the clusters strictly behind it, and so on. When storing the outer centers and their radii, this layout admits a good algorithm for optimistically discarding queries.

In any case, when using clusters as an index there are two extremes

- Store the inner/outer radii for *each* point of every cluster
- Compute just the entom and diam of each cluster

Storing *all* i_x, o_x radii for each cluster requires $O(2n \log(n))$ space, while the cost of diam and entom is negligible. In terms of computation, both diam and entom can be computed without much extra cost while we build the clusters; computing all inner/outer radii is bound to be considerably expensive.

3.1.1 Index Computation

Computing the cluster of radius λ containing a given point reduces to computing a transitive closure; not much can be said without assuming something about the metric at hand. A straightforward algorithm consists in starting with $U = B(x, \lambda)$, then for each $x' \in U$ compute the corresponding ball $B(x', \lambda)$ and add those not in U; now do the same with the newly added points until we reach a fixed point.

In cases where the distance has a discrete range (like in our main case of interest, Hamming distance between strings of the same length) we can compute in parallel all clusters for each possible distance value; the inclusion order can be trivially computed afterwards. We can also follow either a bottom-up or top-down approach, where in the first case clusters get joined to form clusters of bigger radii, while in the second one we can restrict the ambient space to a cluster of a bigger radius.

3.2 Standard Amplicon Situation

We will now outline how a full sequence assignment process for an amplicon where

1. The reference set will be reused across several experiments, and thus it makes sense to index it
2. The number of *different* sequence assignments is small when compared with the number of reads
3. Most reads will be close to some point in the reference

These requirements are satisfied by all the standard amplicons used for taxonomic profiling: 16S for prokaryotes, 18S, mithochondrial 16S, or ITS for eukaryotes, etc.

This sequence assignment process computes the best assignment for each read, or returns it as unassigned if the probability of the best assignment is lower than a user provided threshold.

3.2.1 Find Exact Assignments

First, we compute those reads which have *exact* assignments: a reference equal to the read most likely sequence. A simple hash set will suffice, given that reference databases have on the order of millions of sequences. As output we get a set of references `seen`, sorted from more to less reads assigned to it.

3.2.2 Assign Inexactly or Discard

The rest of the reads will then be either unassigned or inexactly assigned, meaning that their most likely sequence is not part of the reference. We can obtain a metric bound `M` from the user-provided probability threshold for unassignment using the probability bounds 2.1.3 for each read and 2.3.1. The ultrametric index we will use has extra data for clusters their outer centers, outer radii, and inner radii of outer centers.

Two distinct processes are followed for discarding and assigning a read.

For discarding, we traverse an ultrametric index which stores for each cluster its outer center and corresponding radius. All the clusters in a level of this tree constitute a partition of the whole space; thus if the read can be discarded based on the metric bound `M`, there will be a level at which all clusters can be discarded metrically using the distance of the read with the corresponding outer center[9]. Computing the distances with all outer centers per level, we either

- can discard all clusters and thus the read
- the read is assignable after comparing with an outer center

Note that whenever we cannot metrically assign/discard with respect to a center (the distance lies in between the corresponding bounds) we evaluate the corresponding probability directly. In this and in all other probability computations, we actually compute the mismatches count and a bit vector with their positions (between the most likely sequence and the reference), and the maximum probability of the read[10]. From this data we can thus compute those indexes at which they differ[11] and substract (in log space) the corresponding probabilities.

For assigning we simply compute the distances of the most likely sequence with the outer centers, bounding and discarding at each level. As when discarding, whenever metric bounds are uninformative we compute probabilities explicitly.

[9] at worst, at the point where all clusters are singletons
[10] This is trivial for fastq input.
[11] this is fast in any recent intel architecture thanks to specialized instructions.

3.2.3 One Pass Alternative

While the implementation does two passes over the input, it can be run online with minimal modifications. The two-pass implementation is chosen as default based on the assumption that most reads without exact assignments are similar to one of those which have them; this will be so if some of those differences are caused by sequencing errors with respect to those with exact ones.

4 Related Work

Finding the best assignment as we have defined bears a strong resemblance with the classical nearest neighbour problem [4, 3], for which there exist several relevant indexing strategies [1, 2, 10]. It would be interesting to compare the use of the free ultrametric space here with those index structures; in this area ultrametrics appear in [5]. Tropashko [16] is also relevant with respect to tree indexes.

References

[1] **A compact space decomposition for effective metric indexing**
Edgar Chávez and Gonzalo Navarro
Pattern Recognition Letters **26** (9): (2005), 1363–1376.

[2] **An effective clustering algorithm to index high dimensional metric spaces**
Edgar Chávez and Gonzalo Navarro
Proceedings Seventh International Symposium on String Processing and Information Retrieval. SPIRE 2000
IEEE 2000, 75–86.

[3] **Searching in metric spaces**
Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín
ACM computing surveys (CSUR) **33** (3): (2001), 273–321.

[4] **Nearest neighbor queries in metric spaces**
Kenneth L Clarkson
Discrete & Computational Geometry **22** (1): (1999), 63–93.

[5] **Fast, Linear Time Hierarchical Clustering using the Baire Metric**
Pedro Contreras and Fionn Murtagh
Journal of Classification **29** (2 2012), 118–143.
DOI: 10.1007/s00357-012-9106-3.

[6] **Finite Ultrametric Balls**
O. Dovgoshey (October 2018) eprint: 1810.03128v1
URL: http://arxiv.org/abs/1810.03128v1.

[7] **Extremal properties and morphisms of finite ultrametric spaces and their representing trees**
O. Dovgoshey, E. Petrov, and H. -M. Teichert (October 2016) eprint: 1610.08282v2
URL: http://arxiv.org/abs/1610.08282v2.

[8] **Error filtering, pair assembly and error correction for next-generation sequencing reads**
Robert C. Edgar and Henrik Flyvbjerg
Bioinformatics **31** (21 2015), 3476–3482.
DOI: 10.1093/bioinformatics/btv401.

[9] **Spectral decomposition of ultrametric spaces and topos theory**
Alex J Lemin
Topology Proc **26** (2): 2002, 721–739.

[10] **Recursive lists of clusters: A dynamic data structure for range queries in metric spaces**
Margarida Mamede
International Symposium on Computer and Information Sciences Springer 2005, 843–853.

[11] **Illumina error profiles: resolving fine-scale variation in metagenomic sequencing data**
Melanie Schirmer, Rosalinda D'Amore, Umer Z. Ijaz, Neil Hall, and Christopher Quince
BMC Bioinformatics **17** (1 2016)
DOI: 10.1186/s12859-016-0976-y.

[12] **Dissimilarities, Metrics, and Ultrametrics**
Dan A. Simovici (2019).

[13] **Multivalued and Binary Ultrametrics, and Clusterings**
Dan A. Simovici misc Iasi, Romania, 2014
URL: https://www.cs.umb.edu/~sim/sIASI.pdf.

[14] **Several remarks on dissimilarities and ultrametrics**
Dan A Simovici
Scientific Annals of Computer Science **25** (1): (2015), 155.

[15] **Mathematical tools for data mining**
Dan A Simovici and Chabane Djeraba
SpringerVerlag, London (2008).

[16] **Nested intervals tree encoding in SQL**
Vadim Tropashko
ACM SIGMOD Record **34** (2 2005)
DOI: 10.1145/1083784.1083793.