10 Segment match refinement and applications

This exposition is based on:

1. Aaron Halpern, Daniel Huson, Knut Reinert: Segment Match Refinement and Applications, WABI 2002.

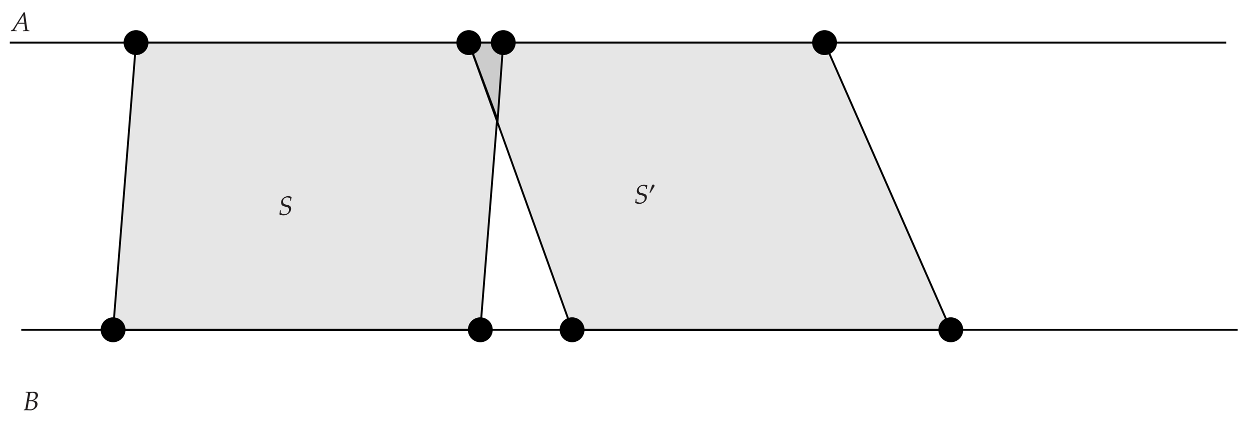
We will present the algorithms used for refining a set of segment matches in such a way that the resulting set of refined matches does not contain any two matches that overlap, but the union of all matches in the refined set is the same as in the original set.

10.1 Motivation

Comparison of large, unfinished genomic sequences requires fast methods that are robust to misordering, misorientation, and duplications. A number of fast methods exist that can compute local similarities between such sequences, from which an optimal one-to-one correspondence might be desired. However, existing methods for computing such a correspondence are either too costly to run or are inappropriate for unfinished sequence.

We discuss an efficient method for refining a set of segment matches such that the resulting segments are of maximal size without non-identity overlaps. This resolved set of segments can be used in various ways to compute a similarity measure between any two large sequences, and hence can be used in alignment, matching, or tree construction algorithms for two or more sequences.

Assume the following two local overlaps are given. It can be easily seen that a greedy algorithm which takes the largest match first and does not allow overlaps can be suboptimal by a factor of almost 3 in the sense that we could also choose a large subsegment of the second match.



We assume the existence of a pre-computed set of local *segment matches* between two sequences. Such sets can be obtained efficiently by various techniques, e.g. BLAST, using suffix arrays, or hashing based schemes. However, a set of local segment matches is not disjoint: more than one match may cover a given interval, whereas the two analyses described above require either a one-to-one matching or a many-to-one matching.

In the following, we introduce an efficient and provably optimal approach to the problem of constructing one-to-one or many-to-one match sets by minimally "refining" (subdividing) matches until all overlaps between the refined matches are "resolved": the projections of any two refined matches onto each sequence are either disjoint or are identical.

Given such a resolved set of matches, several analyses that are otherwise difficult at best become quite straightforward, including not only the one-to-one matching problem and the many-to-one matching problems introduced above but also sequence alignment based on conserved segments.

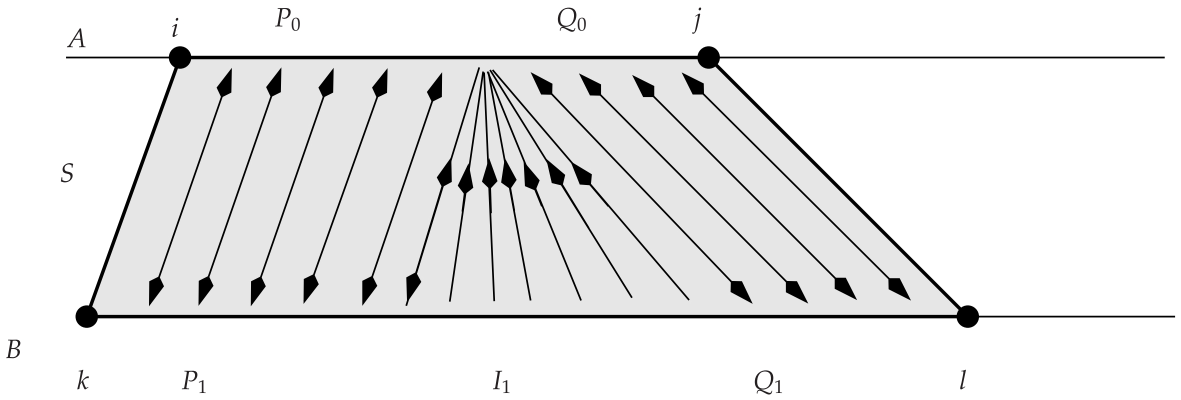
For instance the one-to-one correspondence on the refined set is reduced to the classical matching problem. In addition, it is straightforward to compute an alignment between the sequences if desired (How?).

10.2 Definitions

Let $A = a_1a_2\ldots a_p$ and $B = b_1b_2\ldots b_q$ be two sequences. We call $A_{ij} = a_{i+1}a_{i+2}\ldots a_j$ an *A-segment*, with $0 \le i < j \le p$. A *B-segment* is defined similarly. In the following, we will assume without specific mention that any definitions relative to A carry over to analogous definitions relative to B.

A *segment match* $S = (A_{ij}, B_{kl})$ consists of an A-segment A_{ij} and a B-segment B_{kl}. We call i, j, k and l the *left-A*, *right-A*, *left-B* and *right-B* positions of S, respectively. Given a set of segment matches $\mathcal{S}$, let $supp_A(\mathcal{S})$ denote the *A-support* of $\mathcal{S}$, i. e. the set of left-A and right-A positions of all matches $S \in \mathcal{S}$.

Assume that every segment match $S = (A_{ij}, B_{kl})$ comes with two *projection maps* $\alpha_S : [i, j] \to [k, l]$ and $\beta_S : [k, l] \to [i, j]$ that reflect e.g. the alignment associated with S, as depicted in the below Figure.



In the case of a direct (i. e. orientation preserving) match, we require that α_S and β_S are *non-crossing*, i. e. that the two maps are monotonically increasing and that we have $\alpha_S(h) \le h' \Leftrightarrow h \le \beta_S(h')$ for all $h \in [i, j]$ and $h' \in [k, l]$.

In the case of a reversal, we require that the projections are *(completely) crossing*, i. e. that they are monotonically decreasing and that we have $\alpha_S(h) \le h' \Leftrightarrow h \ge \beta(h')$ for all $h \in [i, j]$ and $h' \in [k, l]$.

A match $S' = (A_{i'j'}, B_{k'l'})$ is called a *submatch of* S, if $i \le i' \le j' \le j$, $k \le k' \le l' \le l$, and we have $\alpha_{S'}(i') = k'$ or $\beta_{S'}(k') = i'$, and $\alpha_{S'}(j') = l'$ or $\beta_{S'}(l') = k'$. Here, we assume that $\alpha'_{S'}$ and $\beta'_{S'}$ equals the restriction of α_S to $[i', j']$, and β_S to $[k', l']$, respectively.

All the previous definitions just formalize trivialities. Now we define the key concept, a *resolved refinement*. Consider a match $S = (A_{ij}, B_{kl} \in \mathcal{S}$. A set of matches $\mathcal{S}'$ is called a *refinement of* S, if each $S' \in \mathcal{S}'$ is a submatch of S and $\mathcal{S}'$ *tiles* S, i. e.:

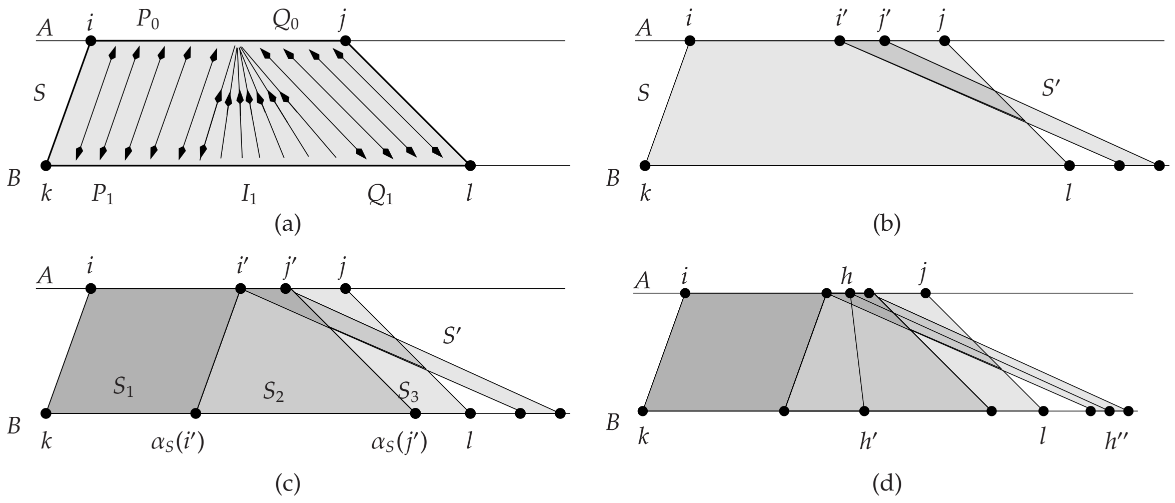
$$(i, j] = \dot{\bigcup_{(A_{i'j'}, B_{k'l'}) \in \mathcal{S}'}} (i', j'] \quad \text{and} \quad (k, l] = \dot{\bigcup_{(A_{i'j'}, B_{k'l'}) \in \mathcal{S}'}} (k', l'].$$

Definition 1. A set $\mathcal{S}'$ of matches is called a *refinement of* $\mathcal{S}$, if there exists a partitioning $\mathcal{S}' = \mathcal{S}'_1 \dot\cup \mathcal{S}'_2 \dot\cup \ldots \dot\cup \mathcal{S}_n$ such that $\mathcal{S}'_i$ is a refinement of S_i for every $S_i \in \mathcal{S}$.

We are particularly interested in refinements of $\mathcal{S}$ that have the property that the segments of any two matches are either disjoint or are identical and capture this property as follows:

Definition 2. A set of matches $\mathcal{S}$ is called *resolved*, if for any $S = (A_{ij}, B_{kl}) \in \mathcal{S}$ we have $[i, j] \cap supp_A(\mathcal{S}) = \{i, j\}$ and $[k, l] \cap supp_B(\mathcal{S}) = \{k, l\}$. For technical reasons, we also require $\{\alpha_S(i), \alpha_S(j)\} \subseteq supp_B(\mathcal{S})$ and $\{\beta_S(k), \beta_S(l)\} \subseteq supp_A(\mathcal{S})$.

10.3 Example



The preceding example shows (a) A valid match $S = (A_{ij}, B_{kl})$, together with arrows depicting α_S (pointing down) and β_S (pointing up). (b) The set $\mathcal{S} = \{S, S'\}$ is unresolved. (c) The set $\{S_1, S_2, S_3, S'\}$ is a minimal resolved refinement of $\mathcal{S}$. (d) This set is a non-minimal resolved refinement of $\mathcal{S}$.

Any set of matches $\mathcal{S}$ possesses a (trivial) resolved refinement obtained by simply refining every $S \in \mathcal{S}$ into a set of single position matches. We are interested in resolved refinements of minimal cardinality and claim:

Lemma 3. *There exists a unique resolved refinement $\bar{\mathcal{S}}$ of $\mathcal{S}$ of minimal cardinality.*

Proof: (Exercise). Sketch: Consider two different resolved refinements $\mathcal{S}_1$ and $\mathcal{S}_2$ of $\mathcal{S}$,both of minimal cardinality. Divide proof into two cases. 1) $(supp_A(\mathcal{S}_1) \neq supp_A(\mathcal{S}_2)$ 2) $supp_A(\mathcal{S}_1) = supp_A(\mathcal{S}_2)$, $supp_B(\mathcal{S}_1) = supp_B(\mathcal{S}_2)$

We now show how to efficiently compute the minimal resolved refinement.

10.4 The algorithm

The input consists of set of segment matches $\mathcal{S}$ and projections α_S, β_S for all $S \in \mathcal{S}$. The output is a resolved refinement $\bar{\mathcal{S}}$ of $\mathcal{S}$ of minimal cardinality.

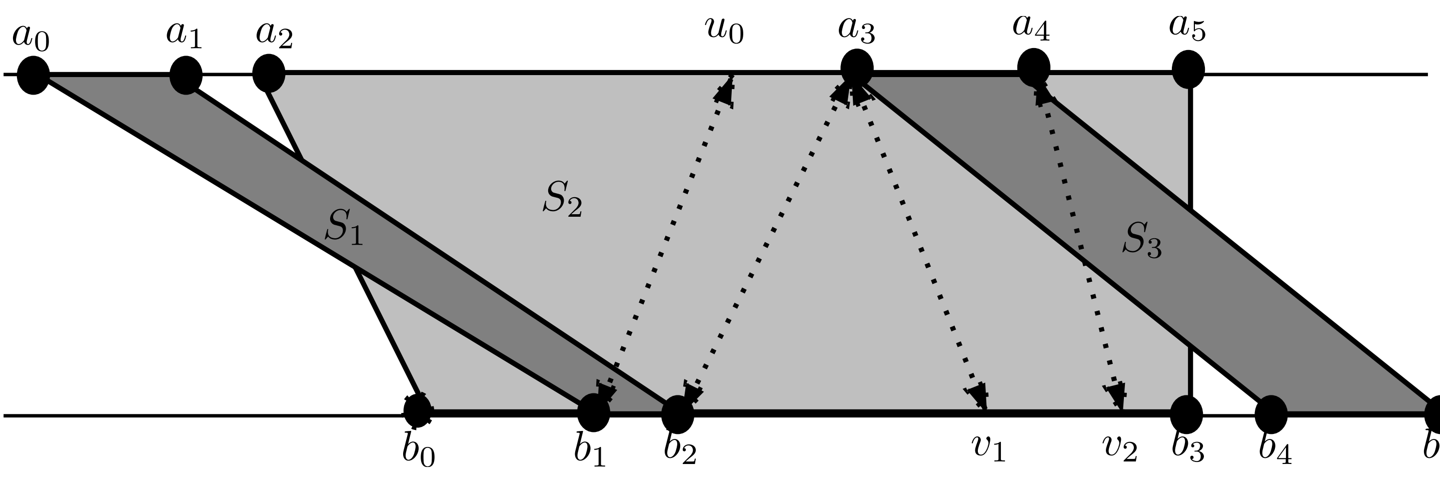
We maintain a bipartite graph $G = (V_A \dot{\cup} V_B, E \subseteq \mathcal{S} \times V_A \times V_B)$ with nodes sets $V_A \subseteq \{a_0, \ldots, a_p\}$ and $V_B \subseteq \{b_0, \ldots, b_q\}$ representing A- and B-positions, respectively, and a set E of (labeled) edges that keep track of pairs of positions projected onto each other by a given sequence match.

```
Refine($\mathcal{S}, \{\alpha_S, \beta_S\}$);
$V_A = V_B = E = \emptyset$;
$W_A = supp_A(\mathcal{S})$;  $W_B = supp_B(\mathcal{S})$;
while ($W_A \neq \emptyset \wedge W_B \neq \emptyset$) do
        $A\_refine(W_A)$;
        $B\_refine(W_B)$;
od
 lexicographically order edges in $E$ using
$(S, i, k)$ or $(S, i, -k)$ depending on whether $\alpha_S$ and $\beta_S$
are non-crossing or crossing;
$\bar{\mathcal{S}} := \{(A_{ij}, B_{kl}) \mid (S, i, j)$ and $(S', k, l)$
 are consecutive and $S = S'\}$;
```

```
A_Refine($W_A$);
if this is not the first execution of A_refine;
  then $W_B = \emptyset$;
fi
for each $h \in W_A \setminus V_A$ do
    for each match $S = (A_{ij}, B_{kl}) \in \mathcal{S}$ with $i \leq h \leq j$ do
        $h' = \alpha_S(h)$;
        if $\nexists$ edge $(S, h, h')$
          then if $\nexists h' \in W_B$
                  then create node $h'$;
               fi
               $G.new\_edge(S, h, h')$; insert $h'$ into $W_B$;
        fi
    od
od
$V_A = V_A \cup W_A$;
```

```
B_Refine($W_B$);
$W_A = \emptyset$;
for each $h \in W_B \setminus V_B$ do
    for each match $S = (A_{ij}, B_{kl}) \in \mathcal{S}$ with $i \leq h \leq j$ do
        $h' = \beta_S(h)$;
        if $\nexists$ edge $(S, h, h')$
          then if $\nexists h' \in W_A$
                  then create node $h'$;
               fi
               $G.new\_edge(S, h, h')$; insert $h'$ into $W_A$;
        fi
    od
od
$V_B = V_B \cup W_B$;
```

10.5 Example



We go through the algorithm using the above example.

- Initially $W_A = \{a_0, \ldots, a_5\}$, $W_B = \{b_0, \ldots, b_5\}$ and $V_A = \emptyset$, $V_B = \emptyset$. We first *A_Refine*. a_3, a_4 lie in the match S_2. Hence we create nodes v_1, v_2 and edges (S_2, a_3, v_1) and (S_2, a_4, v_2). $W_B = \{b_0, \ldots, b_5, v_1, v_2\}$. $V_A = W_A$, since V_A was empty.

10.6 Example

- Now we call *B_Refine* for the first time. b_1, b_2, v_1, v_2 lie in the match S_2. Hence we create a node u_0 (a_3 exists already) and edges (S_2, b_1, u_1) and (S_2, b_2, a_3). $W_A = \{u_0\}$. $V_B = W_B$ since V_B was empty.
- Now we *A_Refine*. u_0, a_3, a_4 lie in the match S_2. But only u_0 is new. No new edge needs to be created since we assume the projections in the example to point to each other. Hence $W_B = \emptyset$ and we can stop.

10.7 Running time

Lemma 4. *Given a set of n segment matches $\mathcal{S}$ between two sequences $A = a_1a_2 \ldots a_p$ and $B = b_1b_2 \ldots b_q$, with $p \geq q$. The graph G in algorithm Refine has at most $O(kp)$ edges and can be computed in at most $O(p \log^2 n + kp \log p)$ steps, where k is the maximal number of segment matches containing any given position.*

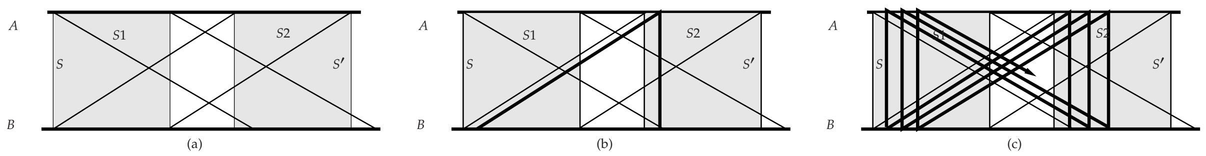
Proof: The maximum number of different A-positions is p. To insert an A-position h into V_A, we first determine which segment matches contain it in $O(\log^2 n + k)$ steps, using a range tree.

For each of the at most k matches $S \in \mathcal{S}$ that contain h, determine whether $\alpha_S(h)$ is contained in V_B in $O(\log p)$ steps. Putting this together, we obtain a bound of $O(p(\log^2 n + k + k \log p)) = O(p \log^2 n + kp \log p)$. Note that each of $O(p)$ positions gives rise to at most k edges in G.

Lemma 5. *Given a set of n segment matches $\mathcal{S}$ between two sequences $A = a_1a_2 \ldots a_p$ and $B = b_1b_2 \ldots b_q$, with $p \geq q$. The minimal resolved refinement $\bar{\mathcal{S}}$ of $\mathcal{S}$ can be computed in at most $O(p \log^2 n + kp \log kp)$ steps, where k is the maximal number of segment matches containing any given position.*

Proof: The preceeding result implies that the ordering step in algorithm Refine will take at most $O(kp \log kp)$ computations.

Note that the size of a minimal resolved refinement depends on the length of the input sequences. For example, a set of only four segment matches between segments of length d, say, can give resize to $4d$ matches, as demonstrated in the below Figure.



10.8 Experiments

The method was tested with two (partial) versions of the drosophila genome. The first was obtained from the BDGP (http://www.fruitfly.org/sequence/dlMfasta.shtml), corresponding to 50,107,720 bp of two finished chromosome arms (2L and 3R). The second was a unpublished assembly of whole genome shotgun sequencing data conducted at Celera in late 2001.

A suffix-tree based method (Mummer) yielded 292,769 alignments seeded with a perfect match of at least 50 bp. These matches covered 49,953,627bp of the finished sequence (i.e., they covered all but 154,093 bp).

43,740,070bp were covered by a single match, but the remainder was covered two or more times; since the total length of all matches was 125,863,932, a relatively small fraction of the sequence is clearly involved in a large number of matches.

The refinement method described in this paper transformed the match set into 17,167,891 distinct matches. This took a total of 880 seconds on a Compaq Unix workstation.

The initial match set, with mean length 429 bp, was cut up into much smaller pieces (mean length 7bp) by the refinement. However, since most of the original sequence was covered by a single match, there should still be matches of considerable size; indeed, the maximum length refined match was 466,877bp, and of the finished sequence covered by matches, half is covered by a refined match of length at least 97,008 bp.

An optimal matching was determined on the refined match set, as described above. The resulting set of matches involved 262,885 refined matches and covered 49,875,599bp, leaving 232,121bp of the finished sequence uncovered.

The greedy method selected 1042 matches totaling 49,754,205 bp, leaving 353,515 bp of the finished sequence uncovered. The mean length is 47,749bp and the N50 is 136,332.

Hence it is questionable whether the refinment is suitable for this analysis. It has probably more merits when there are more overlapping matches and the refined matches are used to compute an alignment.

10.9 Summary

- In many applications it is important to have a 1-to-1 correspondence of subsequences, given a set of sequences.
- Normally, one has a given set of overlapping local alignments.
- One can transform this given set in a set of minimal cardinality such that no two segments intersect properly (they are disjoiunt or identical).