Reweave function

May 2nd, 2024

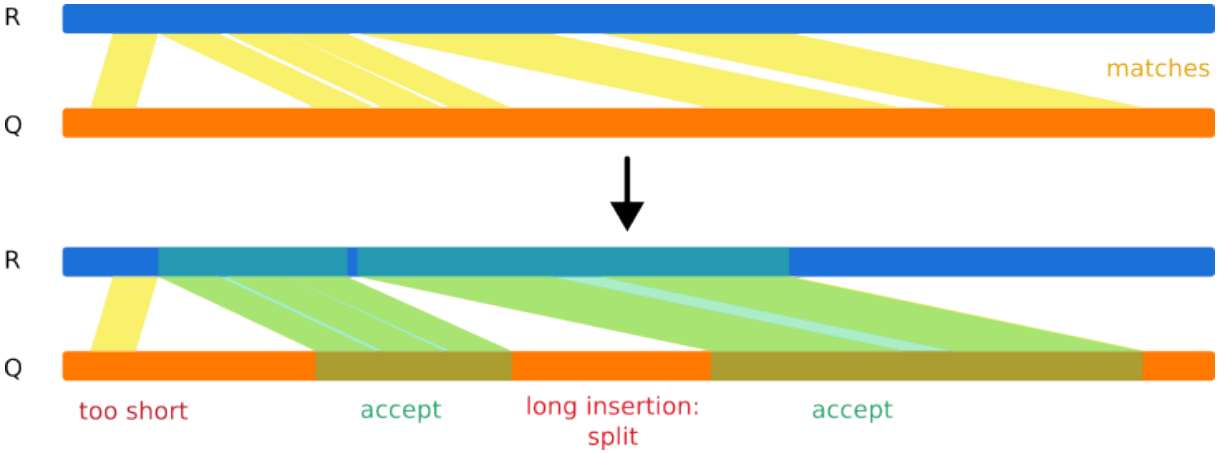
Marco Molari

variable	definition	default value
τ	block minimal length	100 bp
α	block cut cost	100 bp
β	alignment divergence cost	10 bp/SNP

1 Filter and Split matches

Alignment between block consensus sequences can return several matches. These need to be split and filtered to ensure that:

- in/dels longer than the threshold length τ will split a match into multiple *unit-matches*.
- unit-matches have **minimum number of mathches** $\geq \tau$, otherwise they are refused.
- unit-matches are **extended to the start/end of the block** if the overhang has length $\leq \tau$. This is done to avoid excessive fragmentation.



Unit-matches are then filtered based on their length, divergence, and overlap with other matches. The filtering is done by:

1. computing the energy $E = -L + \alpha C + \beta M$ of each unit-match. Here L is the **number of matches** in the alignment, C is the **number of cuts** introduced by the merging ($1 \leq C \leq 4$) and M is the number of mutations/mismatches.
2. unit-matches are sorted by increasing energy value. Matches with $E > 0$ are discarded.
3. unit-matches are then progressively accepted in this order. Every match is accepted if it does not overlap on the reference or query with previously-accepted matches.

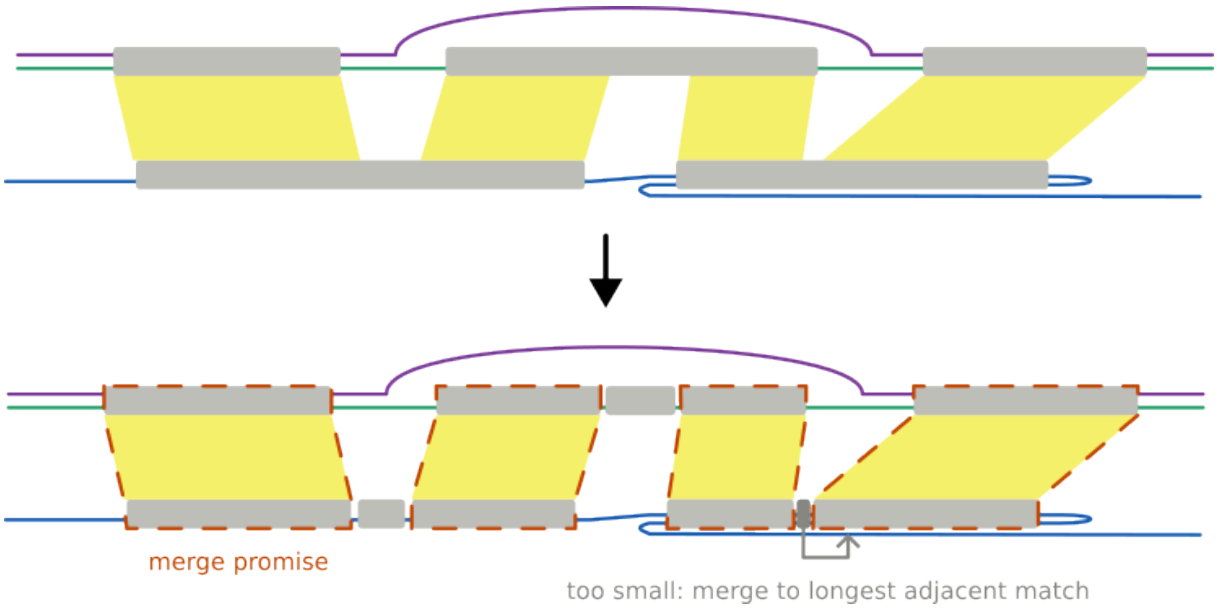
2 Graph Reweaving

This function is called after matches have been found and filtered. Takes as **input**:

- the **pangenome graph** pre-merging. At the first iteration this contains completely separated paths (e.g. π_1, π_2, π_3).
- the **list of filtered unit-matches**, found by the aligner and filtered to ensure that:
  - they are at least of minimal size τ.

- they have energy $E < 0$.
- they do not have flanking regions smaller than the minimal size τ.

It will return as output:
- the **new pangenome graph**, with re-woven paths and split blocks.
- a list of **merge promises**, to be later executed in parallel.



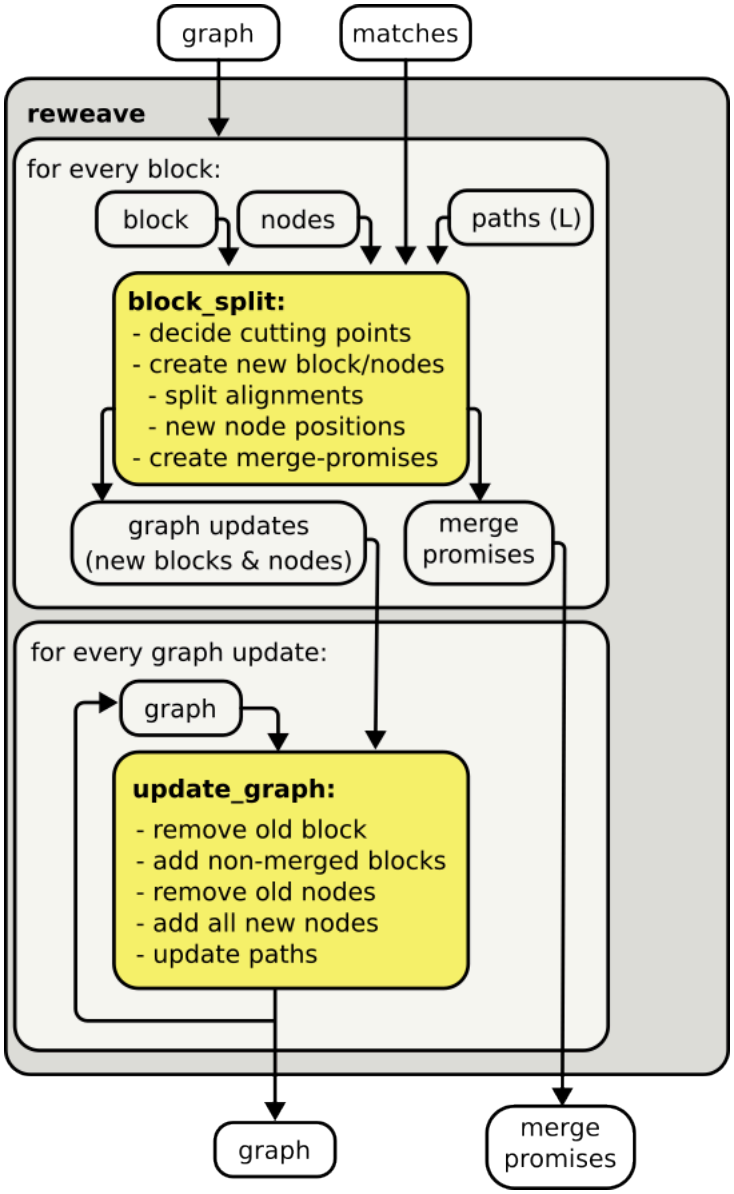
The function works as follows:
1. create a dictionary of merge promises.
2. for every block:
   1. collect all unit-matches whose ref or qry sequence is on the block.
   2. decide split pattern for the block:
      - split along matches
      - any remaining fragments shorter than τ are assigned to the flanking longest fragment.
   3. split and create new blocks / nodes:
      - every fragment is assigned to a new block, with new node ids.
      - if a fragment belongs to a *merge promise*, then create a new entry in the merge-promises dictionary and append it as qry or ref. If the entry already exists then just add it as qry or ref.
   4. update paths:
      - paths are updated with the new block ids and nodes. For merge promises the id of the future block is used.
   5. return the set of merge promises and the new updated graph.

[?] Parallelizing block splitting? But can we create a deterministic and parallelizable id assignment? E.g. hash function of input block/node ids?

[?] ids: numbers or strings? Block ids: from fasta file id or new sequential ids (avoid problems with duplicated record ids?)

[?] Merge promises are composed by two blocks. How do we identify them? Give IDs to alignments?

[?] Provide path lengths for node position decomposition? Or should we just derive from the positions of the full nodes?



[!] later, the merging will **just update blocks, and not update nodes or paths**.