src/abimerge

May 1, 2025

Contents

This module provides a command-line tool for merging two ABI trace files (forward and reverse) into a single sequence.

The abimerge tool uses Smith-Waterman local alignment to find the overlapping region between forward and reverse sequences, then merges them to create a consensus sequence with improved accuracy.

Command-line usage:

```
abimerge [options] <input_F.ab1> <input_R.ab1> [output.fastq]
```

Options:

`-h, --help`	Show help message
`-m, --min-overlap INT`	Minimum overlap length for merging (default: 20)
`-o, --output STRING`	Output file name (default: STDOUT)
`-j, --join INT`	Join with gap of INT Ns if no overlap detected
`--fasta`	Output in FASTA format instead of FASTQ
`--score-match INT`	Score for a match (default: 10)
`--score-mismatch INT`	Score for a mismatch (default: -8)
`--score-gap INT`	Score for a gap (default: -10)
`--min-score INT`	Minimum alignment score (default: 80)
`--pct-id FLOAT`	Minimum percentage identity (default: 85)

Examples:

```
# Merge two ABIF files with default settings
abimerge forward.ab1 reverse.ab1 merged.fastq

# Merge with custom alignment parameters
abimerge --min-overlap 30 --score-match 12 forward.ab1 reverse.ab1 merged.fastq

# Join sequences with N gap if no overlap
abimerge -j 10 forward.ab1 reverse.ab1 merged.fastq

# Output in FASTA format instead of FASTQ
abimerge --fasta forward.ab1 reverse.ab1 merged.fasta
```

1 Imports

abif

2 Types

```
Config = object
  inputFileF*: string
  inputFileR*: string
  outputFile*: string
  minOverlap*: int
  scoreMatch*: int
  scoreMismatch*: int
```

```
  scoreGap*: int
  minScore*: int
  pctId*: float
  joinGap*: int
  verbose*: bool
  windowSize*: int
  qualityThreshold*: int
  noTrim*: bool
  showVersion*: bool
  fasta*: bool
```

```
swAlignment = object
  top*: string                ## The first sequence in the alignment
  bottom*: string             ## The second sequence in the alignment
  middle*: string             ## The alignment representation (|, ., or space)
  score*: int                 ## The alignment score
  length*: int                ## The length of the alignment
  pctid*: float
  queryStart*, queryEnd*, targetStart*, targetEnd*: int
```

Represents a Smith-Waterman alignment between two sequences.

```
swWeights = object
  match*: int                 ## Score for matching bases
  mismatch*: int              ## Penalty for mismatched bases
  gap*: int                   ## Penalty for gap extension
  gapopening*: int            ## Penalty for opening a gap
  minscore*: int              ## Minimum score for accepting an alignment
```

Scoring parameters for Smith-Waterman alignment.

3 Lets

```
swDefaults = swWeights(match: 6, mismatch: -4, gap: -6, gapopening: -6,
                       minscore: 1)
```

4 Procs

```
proc makeMatrix[T](rows, cols: int; initValue: T): seq[seq[T]]
```

```
proc matchIUPAC(a, b: char): bool {.raises: [], tags: [], forbids: [].}
```

```
proc mergeSequences(forwardSeq: string; forwardQual: seq[int];
                    reverseSeq: string; reverseQual: seq[int]; config: Config):
                    ↪  tuple[
    seq: string, qual: seq[int]] {.raises: [], tags: [], forbids: [].}
```

```
proc revcompl(s: string): string {.raises: [], tags: [], forbids: [].}
```

```
proc reverseString(str: string): string {.raises: [], tags: [], forbids: [].}
```

```
proc simpleSmithWaterman(alpha, beta: string; weights: swWeights): swAlignment {.
    raises: [], tags: [], forbids: [].}
```

```
proc translateIUPAC(c: char): char {.raises: [], tags: [], forbids: [].}
```