

# NextGenMap-LR: Highly accurate read mapping of third generation sequencing reads for improved structural variation analysis

Philipp Rescheneder

Center for Integrative Bioinformatics Vienna

October 5, 2015

- Third generation sequencing
- Single Molecule, Real Time Sequencing (SMRT)
- Advantages:
  - Read length
  - No amplification step: uniform coverage
  - High consensus accuracy
- Disadvantages:
  - Sequencing error on average 10 - 15 %
  - Mostly insertions and deletions
  - Parts of reads can show a much higher error rate
- Applications
  - Assembly using error corrected reads
  - Sequencing of repetitive regions
  - Sequencing of regions showing high/low GC content, highly repetitive regions, palindromic regions or long homopolymers
  - Finding and characterising structural variations

- Third generation sequencing
- Single Molecule, Real Time Sequencing (SMRT)
- Advantages:
  - Read length
  - No amplification step: uniform coverage
  - High consensus accuracy
- Disadvantages:
  - Sequencing error on average 10 - 15 %
  - Mostly insertions and deletions
  - Parts of reads can show a much higher error rate
- Applications
  - Assembly using error corrected reads
  - Sequencing of repetitive regions
  - Sequencing of regions showing high/low GC content, highly repetitive regions, palindromic regions or long homopolymers
  - Finding and characterising structural variations

- Third generation sequencing
- Single Molecule, Real Time Sequencing (SMRT)
- Advantages:
  - Read length
  - No amplification step: uniform coverage
  - High consensus accuracy
- Disadvantages:
  - Sequencing error on average 10 - 15 %
  - Mostly insertions and deletions
  - Parts of reads can show a much higher error rate
- Applications
  - Assembly using error corrected reads
  - Sequencing of repetitive regions
  - Sequencing of regions showing high/low GC content, highly repetitive regions, palindromic regions or long homopolymers
  - Finding and characterising structural variations

- Third generation sequencing
- Single Molecule, Real Time Sequencing (SMRT)
- Advantages:
  - Read length
  - No amplification step: uniform coverage
  - High consensus accuracy
- Disadvantages:
  - Sequencing error on average 10 - 15 %
  - Mostly insertions and deletions
  - Parts of reads can show a much higher error rate
- Applications
  - Assembly using error corrected reads
  - Sequencing of repetitive regions
  - Sequencing of regions showing high/low GC content, highly repetitive regions, palindromic regions or long homopolymers
  - Finding and characterising structural variations

- Third generation sequencing
- Single Molecule, Real Time Sequencing (SMRT)
- Advantages:
  - Read length
  - No amplification step: uniform coverage
  - High consensus accuracy
- Disadvantages:
  - Sequencing error on average 10 - 15 %
  - Mostly insertions and deletions
  - Parts of reads can show a much higher error rate
- Applications
  - Assembly using error corrected reads
  - Sequencing of repetitive regions
  - Sequencing of regions showing high/low GC content, highly repetitive regions, palindromic regions or long homopolymers
  - Finding and characterising structural variations

- Third generation sequencing
- Single Molecule, Real Time Sequencing (SMRT)
- Advantages:
  - Read length
  - No amplification step: uniform coverage
  - High consensus accuracy
- Disadvantages:
  - Sequencing error on average 10 - 15 %
  - Mostly insertions and deletions
  - Parts of reads can show a much higher error rate
- Applications
  - Assembly using error corrected reads
  - Sequencing of repetitive regions
  - Sequencing of regions showing high/low GC content, highly repetitive regions, palindromic regions or long homopolymers
  - Finding and characterising structural variations

- Third generation sequencing
- Single Molecule, Real Time Sequencing (SMRT)
- Advantages:
  - Read length
  - No amplification step: uniform coverage
  - High consensus accuracy
- Disadvantages:
  - Sequencing error on average 10 - 15 %
  - Mostly insertions and deletions
  - Parts of reads can show a much higher error rate
- Applications
  - Assembly using error corrected reads
  - Sequencing of repetitive regions
  - Sequencing of regions showing high/low GC content, highly repetitive regions, palindromic regions or long homopolymers
  - Finding and characterising structural variations



- Third generation sequencing
- Single Molecule, Real Time Sequencing (SMRT)
- Advantages:
  - Read length
  - No amplification step: uniform coverage
  - High consensus accuracy
- Disadvantages:
  - Sequencing error on average 10 - 15 %
  - Mostly insertions and deletions
  - Parts of reads can show a much higher error rate
- Applications
  - Assembly using error corrected reads
  - Sequencing of repetitive regions
  - Sequencing of regions showing high/low GC content, highly repetitive regions, palindromic regions or long homopolymers
  - Finding and characterising structural variations

- Third generation sequencing
- Single Molecule, Real Time Sequencing (SMRT)
- Advantages:
  - Read length
  - No amplification step: uniform coverage
  - High consensus accuracy
- Disadvantages:
  - Sequencing error on average 10 - 15 %
  - Mostly insertions and deletions
  - Parts of reads can show a much higher error rate
- Applications
  - Assembly using error corrected reads
  - Sequencing of repetitive regions
  - Sequencing of regions showing high/low GC content, highly repetitive regions, palindromic regions or long homopolymers
  - Finding and characterising structural variations

- Third generation sequencing
- Single Molecule, Real Time Sequencing (SMRT)
- Advantages:
  - Read length
  - No amplification step: uniform coverage
  - High consensus accuracy
- Disadvantages:
  - Sequencing error on average 10 - 15 %
  - Mostly insertions and deletions
  - Parts of reads can show a much higher error rate
- Applications
  - Assembly using error corrected reads
  - Sequencing of repetitive regions
  - Sequencing of regions showing high/low GC content, highly repetitive regions, palindromic regions or long homopolymers
  - Finding and characterising structural variations

- Third generation sequencing
- Single Molecule, Real Time Sequencing (SMRT)
- Advantages:
  - Read length
  - No amplification step: uniform coverage
  - High consensus accuracy
- Disadvantages:
  - Sequencing error on average 10 - 15 %
  - Mostly insertions and deletions
  - Parts of reads can show a much higher error rate
- Applications
  - Assembly using error corrected reads
  - Sequencing of repetitive regions
  - Sequencing of regions showing high/low GC content, highly repetitive regions, palindromic regions or long homopolymers
  - Finding and characterising structural variations

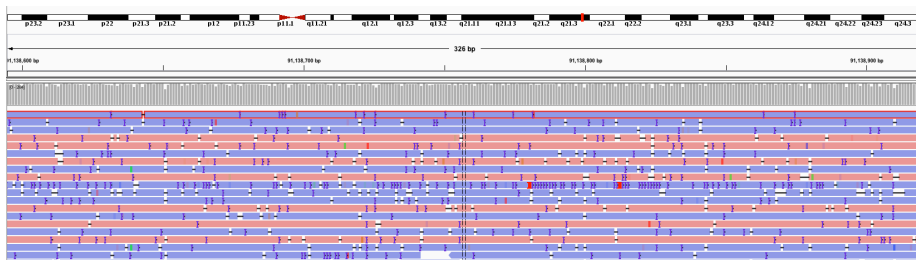
- Third generation sequencing
- Single Molecule, Real Time Sequencing (SMRT)
- Advantages:
  - Read length
  - No amplification step: uniform coverage
  - High consensus accuracy
- Disadvantages:
  - Sequencing error on average 10 - 15 %
  - Mostly insertions and deletions
  - Parts of reads can show a much higher error rate
- Applications
  - Assembly using error corrected reads
  - Sequencing of repetitive regions
  - Sequencing of regions showing high/low GC content, highly repetitive regions, palindromic regions or long homopolymers
  - Finding and characterising structural variations

- Third generation sequencing
- Single Molecule, Real Time Sequencing (SMRT)
- Advantages:
  - Read length
  - No amplification step: uniform coverage
  - High consensus accuracy
- Disadvantages:
  - Sequencing error on average 10 - 15 %
  - Mostly insertions and deletions
  - Parts of reads can show a much higher error rate
- Applications
  - Assembly using error corrected reads
  - Sequencing of repetitive regions
  - Sequencing of regions showing high/low GC content, highly repetitive regions, palindromic regions or long homopolymers
  - Finding and characterising structural variations

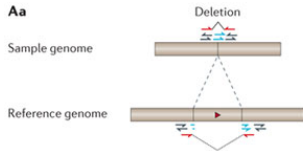
- Third generation sequencing
- Single Molecule, Real Time Sequencing (SMRT)
- Advantages:
  - Read length
  - No amplification step: uniform coverage
  - High consensus accuracy
- Disadvantages:
  - Sequencing error on average 10 - 15 %
  - Mostly insertions and deletions
  - Parts of reads can show a much higher error rate
- Applications
  - Assembly using error corrected reads
  - Sequencing of repetitive regions
  - Sequencing of regions showing high/low GC content, highly repetitive regions, palindromic regions or long homopolymers
  - Finding and characterising structural variations

- Third generation sequencing
- Single Molecule, Real Time Sequencing (SMRT)
- Advantages:
  - Read length
  - No amplification step: uniform coverage
  - High consensus accuracy
- Disadvantages:
  - Sequencing error on average 10 - 15 %
  - Mostly insertions and deletions
  - Parts of reads can show a much higher error rate
- Applications
  - Assembly using error corrected reads
  - Sequencing of repetitive regions
  - Sequencing of regions showing high/low GC content, highly repetitive regions, palindromic regions or long homopolymers
  - Finding and characterising structural variations

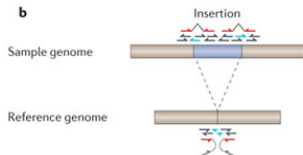




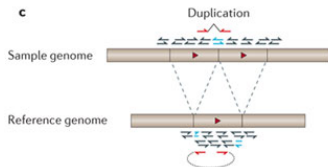
**Aa**



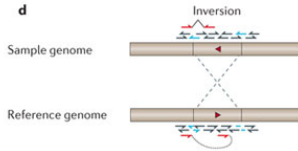
**b**



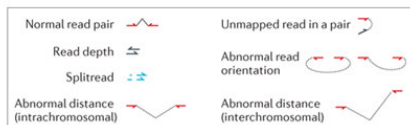
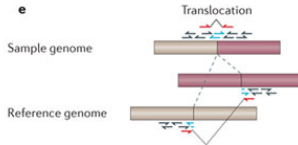
**c**



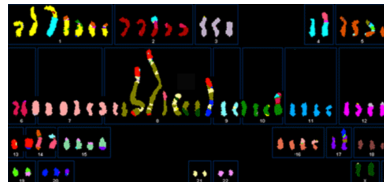
**d**



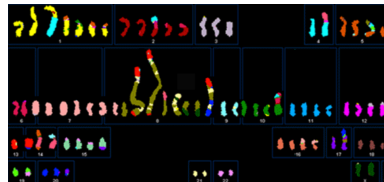
**e**



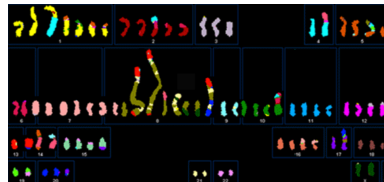
- Cell line: SKBR3
- Her2 region strongly amplified
- 20 % of breast cancers, 2-3x recurrence risk, 5x metastasis risk
- Goal of project: find structural variations, gene fusions, etc.



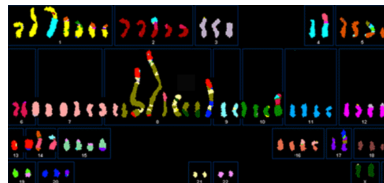
- Cell line: SKBR3
- Her2 region strongly amplified
- 20 % of breast cancers, 2-3x recurrence risk, 5x metastasis risk
- Goal of project: find structural variations, gene fusions, etc.



- Cell line: SKBR3
- Her2 region strongly amplified
- 20 % of breast cancers, 2-3x recurrence risk, 5x metastasis risk
- Goal of project: find structural variations, gene fusions, etc.



- Cell line: SKBR3
- Her2 region strongly amplified
- 20 % of breast cancers, 2-3x recurrence risk, 5x metastasis risk
- Goal of project: find structural variations, gene fusions, etc.



- BLASR: official tool from PacBio
- BWA mem: very fast
- LAST: good results, but slow for human data

- BLASR: official tool from PacBio
- BWA mem: very fast
- LAST: good results, but slow for human data



- BLASR: official tool from PacBio
- BWA mem: very fast
- LAST: good results, but slow for human data

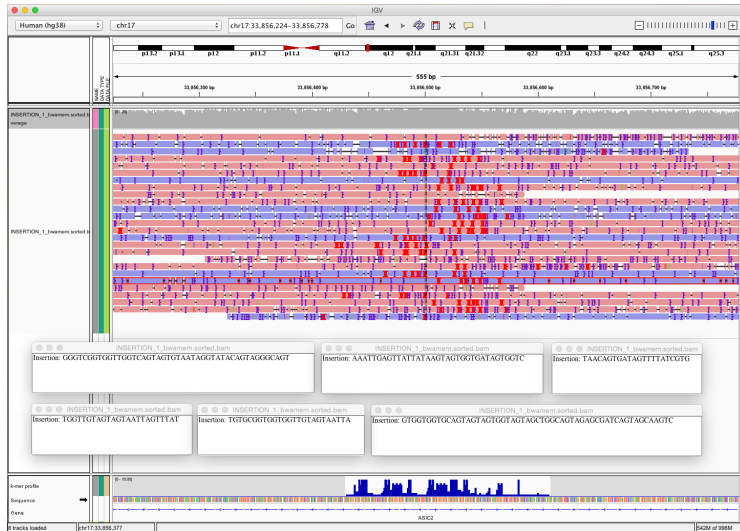
# Problems: inversion



# Problems: deletion



# Problems: insertion



AAAGAATTCA                      vs.                      AAAGAATTCA  
A-A-A-T-CA                      AAA----TCA

- Linear: gap cost always the same, only one matrix required ( $n^2$ )
- Affine: separate penalty for opening and for extending a gap
- Implementation: 3 matrices instead of one (still  $n^2$ )
- Using affine gap costs is considered state of the art
- Problem with PacBio: two different gap models required
  - Sequencing error: high number of 1 bp indels
  - Real indels: extending a gap more likely than opening a new one
- Sequencing error + repeats cause affine gap costs to fail even for real indels

AAAGAATTCA                      vs.                      AAAGAATTCA  
A-A-A-T-CA                      AAA----TCA

- Linear: gap cost always the same, only one matrix required ( $n^2$ )
- Affine: separate penalty for opening and for extending a gap
- Implementation: 3 matrices instead of one (still  $n^2$ )
- Using affine gap costs is considered state of the art
- Problem with PacBio: two different gap models required
  - Sequencing error: high number of 1 bp indels
  - Real indels: extending a gap more likely than opening a new one
- Sequencing error + repeats cause affine gap costs to fail even for real indels

AAAGAATTCA                      vs.                      AAAGAATTCA  
A-A-A-T-CA                      AAA----TCA

- Linear: gap cost always the same, only one matrix required ( $n^2$ )
- Affine: separate penalty for opening and for extending a gap
- Implementation: 3 matrices instead of one (still  $n^2$ )
- Using affine gap costs is considered state of the art
- Problem with PacBio: two different gap models required
  - Sequencing error: high number of 1 bp indels
  - Real indels: extending a gap more likely than opening a new one
- Sequencing error + repeats cause affine gap costs to fail even for real indels

AAAGAATTCA  
A-A-A-T-CA

vs.

AAAGAATTCA  
AAA----TCA

- Linear: gap cost always the same, only one matrix required ( $n^2$ )
- Affine: separate penalty for opening and for extending a gap
- Implementation: 3 matrices instead of one (still  $n^2$ )
- Using affine gap costs is considered state of the art
- Problem with PacBio: two different gap models required
  - Sequencing error: high number of 1 bp indels
  - Real indels: extending a gap more likely than opening a new one
- Sequencing error + repeats cause affine gap costs to fail even for real indels



AAAGAATTCA  
A-A-A-T-CA

vs.

AAAGAATTCA  
AAA----TCA

- Linear: gap cost always the same, only one matrix required ( $n^2$ )
- Affine: separate penalty for opening and for extending a gap
- Implementation: 3 matrices instead of one (still  $n^2$ )
- Using affine gap costs is considered state of the art
- Problem with PacBio: two different gap models required
  - Sequencing error: high number of 1 bp indels
  - Real indels: extending a gap more likely than opening a new one
- Sequencing error + repeats cause affine gap costs to fail even for real indels

AAAGAATTCA                      vs.                      AAAGAATTCA  
A-A-A-T-CA                      AAA----TCA

- Linear: gap cost always the same, only one matrix required ( $n^2$ )
- Affine: separate penalty for opening and for extending a gap
- Implementation: 3 matrices instead of one (still  $n^2$ )
- Using affine gap costs is considered state of the art
- Problem with PacBio: two different gap models required
  - Sequencing error: high number of 1 bp indels
  - Real indels: extending a gap more likely than opening a new one
- Sequencing error + repeats cause affine gap costs to fail even for real indels

AAAGAATTCA  
A-A-A-T-CA

vs.

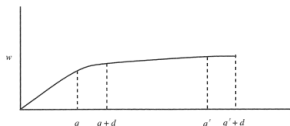
AAAGAATTCA  
AAA----TCA

- Linear: gap cost always the same, only one matrix required ( $n^2$ )
- Affine: separate penalty for opening and for extending a gap
- Implementation: 3 matrices instead of one (still  $n^2$ )
- Using affine gap costs is considered state of the art
- Problem with PacBio: two different gap models required
  - Sequencing error: high number of 1 bp indels
  - Real indels: extending a gap more likely than opening a new one
- Sequencing error + repeats cause affine gap costs to fail even for real indels

AAAGAATTCA                      vs.                      AAAGAATTCA  
A-A-A-T-CA                      AAA----TCA

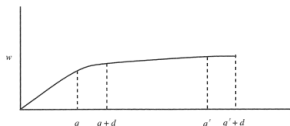
- Linear: gap cost always the same, only one matrix required ( $n^2$ )
- Affine: separate penalty for opening and for extending a gap
- Implementation: 3 matrices instead of one (still  $n^2$ )
- Using affine gap costs is considered state of the art
- Problem with PacBio: two different gap models required
  - Sequencing error: high number of 1 bp indels
  - Real indels: extending a gap more likely than opening a new one
- Sequencing error + repeats cause affine gap costs to fail even for real indels

- Cost for a gap is a convex function of gap length



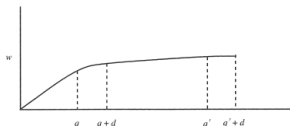
- Close to linear gap costs for 1bp gaps
- As gap gets longer penalty for "splitting" gaps increases
- Naive approach:  $O(nm^2 + n^2m)$
- Has to "look back" for each cell
- Improved algorithm using candidate lists:  $O(nm \log m)$
- Complex algorithm, no suitable implementations available
- Most probably still too slow for PacBio reads

- Cost for a gap is a convex function of gap length



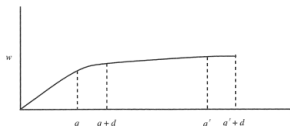
- Close to linear gap costs for 1bp gaps
- As gap gets longer penalty for "splitting" gaps increases
- Naive approach:  $O(nm^2 + n^2m)$
- Has to "look back" for each cell
- Improved algorithm using candidate lists:  $O(nm \log m)$
- Complex algorithm, no suitable implementations available
- Most probably still too slow for PacBio reads

- Cost for a gap is a convex function of gap length



- Close to linear gap costs for 1bp gaps
- As gap gets longer penalty for "splitting" gaps increases
- Naive approach:  $O(nm^2 + n^2m)$
- Has to "look back" for each cell
- Improved algorithm using candidate lists:  $O(nm \log m)$
- Complex algorithm, no suitable implementations available
- Most probably still too slow for PacBio reads

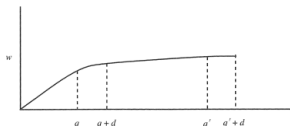
- Cost for a gap is a convex function of gap length



- Close to linear gap costs for 1bp gaps
- As gap gets longer penalty for "splitting" gaps increases
- Naive approach:  $O(nm^2 + n^2m)$
- Has to "look back" for each cell
- Improved algorithm using candidate lists:  $O(nm \log m)$
- Complex algorithm, no suitable implementations available
- Most probably still too slow for PacBio reads

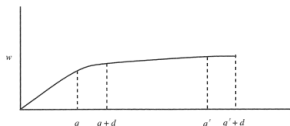


- Cost for a gap is a convex function of gap length



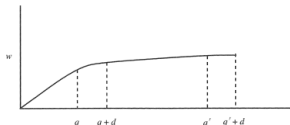
- Close to linear gap costs for 1bp gaps
- As gap gets longer penalty for "splitting" gaps increases
- Naive approach:  $O(nm^2 + n^2m)$
- Has to "look back" for each cell
- Improved algorithm using candidate lists:  $O(nm \log m)$
- Complex algorithm, no suitable implementations available
- Most probably still too slow for PacBio reads

- Cost for a gap is a convex function of gap length



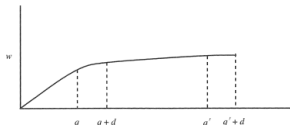
- Close to linear gap costs for 1bp gaps
- As gap gets longer penalty for "splitting" gaps increases
- Naive approach:  $O(nm^2 + n^2m)$
- Has to "look back" for each cell
- Improved algorithm using candidate lists:  $O(nm \log m)$
- Complex algorithm, no suitable implementations available
- Most probably still too slow for PacBio reads

- Cost for a gap is a convex function of gap length



- Close to linear gap costs for 1bp gaps
- As gap gets longer penalty for "splitting" gaps increases
- Naive approach:  $O(nm^2 + n^2m)$
- Has to "look back" for each cell
- Improved algorithm using candidate lists:  $O(nm \log m)$
- Complex algorithm, no suitable implementations available
- Most probably still too slow for PacBio reads

- Cost for a gap is a convex function of gap length



- Close to linear gap costs for 1bp gaps
- As gap gets longer penalty for "splitting" gaps increases
- Naive approach:  $O(nm^2 + n^2m)$
- Has to "look back" for each cell
- Improved algorithm using candidate lists:  $O(nm \log m)$
- Complex algorithm, no suitable implementations available
- Most probably still too slow for PacBio reads

- Gap decay
  - Example: cost 10, decay 0.1
  - $g(1) = 10, g(2) = 10 + 9, g(3) = 10 + 9 + 8, g(4) = 10 + 9 + 8 + 7, \dots$
  - Inspired by an implementation from the python library swalign
  - Uses backtracking pointers to track length of current gap
  - Does not guarantee to find optimal alignment!
  - Simple C++ implementation without any optimisations so far

- Gap decay
- Example: cost 10, decay 0.1
  - $g(1) = 10, g(2) = 10 + 9, g(3) = 10 + 9 + 8, g(4) = 10 + 9 + 8 + 7, \dots$
  - Inspired by an implementation from the python library swalign
  - Uses backtracking pointers to track length of current gap
  - Does not guarantee to find optimal alignment!
  - Simple C++ implementation without any optimisations so far

- Gap decay
- Example: cost 10, decay 0.1
- $g(1) = 10, g(2) = 10 + 9, g(3) = 10 + 9 + 8, g(4) = 10 + 9 + 8 + 7, \dots$
- Inspired by an implementation from the python library swalign
- Uses backtracking pointers to track length of current gap
- Does not guarantee to find optimal alignment!
- Simple C++ implementation without any optimisations so far

- Gap decay
- Example: cost 10, decay 0.1
- $g(1) = 10, g(2) = 10 + 9, g(3) = 10 + 9 + 8, g(4) = 10 + 9 + 8 + 7, \dots$
- Inspired by an implementation from the python library swalign
- Uses backtracking pointers to track length of current gap
- Does not guarantee to find optimal alignment!
- Simple C++ implementation without any optimisations so far

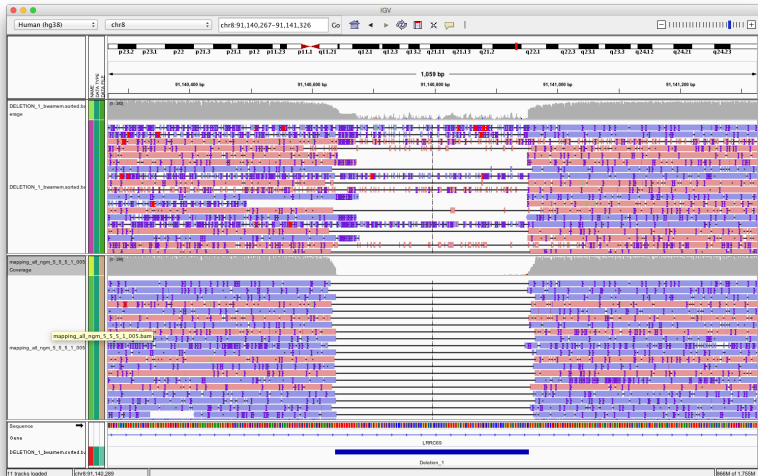


- Gap decay
- Example: cost 10, decay 0.1
- $g(1) = 10, g(2) = 10 + 9, g(3) = 10 + 9 + 8, g(4) = 10 + 9 + 8 + 7, \dots$
- Inspired by an implementation from the python library swalign
- Uses backtracking pointers to track length of current gap
- Does not guarantee to find optimal alignment!
- Simple C++ implementation without any optimisations so far

- Gap decay
- Example: cost 10, decay 0.1
- $g(1) = 10, g(2) = 10 + 9, g(3) = 10 + 9 + 8, g(4) = 10 + 9 + 8 + 7, \dots$
- Inspired by an implementation from the python library swalign
- Uses backtracking pointers to track length of current gap
- Does not guarantee to find optimal alignment!
- Simple C++ implementation without any optimisations so far

- Gap decay
- Example: cost 10, decay 0.1
- $g(1) = 10, g(2) = 10 + 9, g(3) = 10 + 9 + 8, g(4) = 10 + 9 + 8 + 7, \dots$
- Inspired by an implementation from the python library swalign
- Uses backtracking pointers to track length of current gap
- Does not guarantee to find optimal alignment!
- Simple C++ implementation without any optimisations so far

# Example realignment: deletion



## Problems:

- Runtime: naive implementation requires 0.5 to 1.5 seconds for aligning a read
- Memory consumption

## Problems:

- Runtime: naive implementation requires 0.5 to 1.5 seconds for aligning a read
- Memory consumption

- Problem: finding which part of the read should be aligned to which part of the reference
- More demanding than for short reads
- Especially in the context of structural variations: when to split an alignment?

- Problem: finding which part of the read should be aligned to which part of the reference
- More demanding than for short reads
- Especially in the context of structural variations: when to split an alignment?



- Problem: finding which part of the read should be aligned to which part of the reference
- More demanding than for short reads
- Especially in the context of structural variations: when to split an alignment?

- Split reads into non-overlapping 512 bp anchors
- Map anchors independently using NextGenMap
  - Candidate mapping region search: k-mers
  - Compute alignment scores
- Filter anchors stringently based on alignment score and number of mapping positions
- Result: tuples consisting of read position and reference position (anchor)
- Example:  
0/1000, 512/1510/, 1024/2030, 1024/512312, 1024/348, 1536/2530
- Use modified longest increasing subsequence algorithm to find the best candidate segments (compatible anchors in close proximity)
- Example 1000, 1510, 2030, 2530

- Split reads into non-overlapping 512 bp anchors
- Map anchors independently using NextGenMap
  - Candidate mapping region search: k-mers
  - Compute alignment scores
- Filter anchors stringently based on alignment score and number of mapping positions
- Result: tuples consisting of read position and reference position (anchor)
- Example:  
0/1000, 512/1510/, 1024/2030, 1024/512312, 1024/348, 1536/2530
- Use modified longest increasing subsequence algorithm to find the best candidate segments (compatible anchors in close proximity)
- Example 1000, 1510, 2030, 2530

- Split reads into non-overlapping 512 bp anchors
- Map anchors independently using NextGenMap
  - Candidate mapping region search: k-mers
  - Compute alignment scores
- Filter anchors stringently based on alignment score and number of mapping positions
- Result: tuples consisting of read position and reference position (anchor)
- Example:  
0/1000, 512/1510/, 1024/2030, 1024/512312, 1024/348, 1536/2530
- Use modified longest increasing subsequence algorithm to find the best candidate segments (compatible anchors in close proximity)
- Example 1000, 1510, 2030, 2530

- Split reads into non-overlapping 512 bp anchors
- Map anchors independently using NextGenMap
  - Candidate mapping region search: k-mers
  - Compute alignment scores
- Filter anchors stringently based on alignment score and number of mapping positions
- Result: tuples consisting of read position and reference position (anchor)
- Example:  
0/1000, 512/1510/, 1024/2030, 1024/512312, 1024/348, 1536/2530
- Use modified longest increasing subsequence algorithm to find the best candidate segments (compatible anchors in close proximity)
- Example 1000, 1510, 2030, 2530

- Split reads into non-overlapping 512 bp anchors
- Map anchors independently using NextGenMap
  - Candidate mapping region search: k-mers
  - Compute alignment scores
- Filter anchors stringently based on alignment score and number of mapping positions
- Result: tuples consisting of read position and reference position (anchor)
- Example:  
0/1000, 512/1510/, 1024/2030, 1024/512312, 1024/348, 1536/2530
- Use modified longest increasing subsequence algorithm to find the best candidate segments (compatible anchors in close proximity)
- Example 1000, 1510, 2030, 2530

- Split reads into non-overlapping 512 bp anchors
- Map anchors independently using NextGenMap
  - Candidate mapping region search: k-mers
  - Compute alignment scores
- Filter anchors stringently based on alignment score and number of mapping positions
- Result: tuples consisting of read position and reference position (anchor)
- Example:  
0/1000, 512/1510/, 1024/2030, 1024/512312, 1024/348, 1536/2530
- Use modified longest increasing subsequence algorithm to find the best candidate segments (compatible anchors in close proximity)
- Example 1000, 1510, 2030, 2530

- Split reads into non-overlapping 512 bp anchors
- Map anchors independently using NextGenMap
  - Candidate mapping region search: k-mers
  - Compute alignment scores
- Filter anchors stringently based on alignment score and number of mapping positions
- Result: tuples consisting of read position and reference position (anchor)
- Example:  
0/1000, 512/1510/, 1024/2030, 1024/512312, 1024/348, 1536/2530
- Use modified longest increasing subsequence algorithm to find the best candidate segments (compatible anchors in close proximity)
- Example 1000, 1510, 2030, 2530

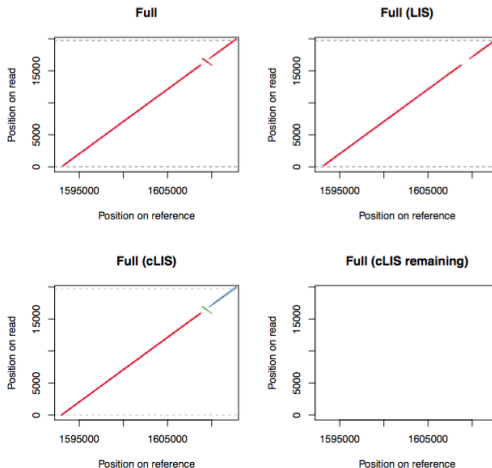


- Split reads into non-overlapping 512 bp anchors
- Map anchors independently using NextGenMap
  - Candidate mapping region search: k-mers
  - Compute alignment scores
- Filter anchors stringently based on alignment score and number of mapping positions
- Result: tuples consisting of read position and reference position (anchor)
- Example:  
0/1000, 512/1510/, 1024/2030, 1024/512312, 1024/348, 1536/2530
- Use modified longest increasing subsequence algorithm to find the best candidate segments (compatible anchors in close proximity)
- Example 1000, 1510, 2030, 2530

- Split reads into non-overlapping 512 bp anchors
- Map anchors independently using NextGenMap
  - Candidate mapping region search: k-mers
  - Compute alignment scores
- Filter anchors stringently based on alignment score and number of mapping positions
- Result: tuples consisting of read position and reference position (anchor)
- Example:  
0/1000, 512/1510/, 1024/2030, 1024/512312, 1024/348, 1536/2530
- Use modified longest increasing subsequence algorithm to find the best candidate segments (compatible anchors in close proximity)
- Example 1000, 1510, 2030, 2530

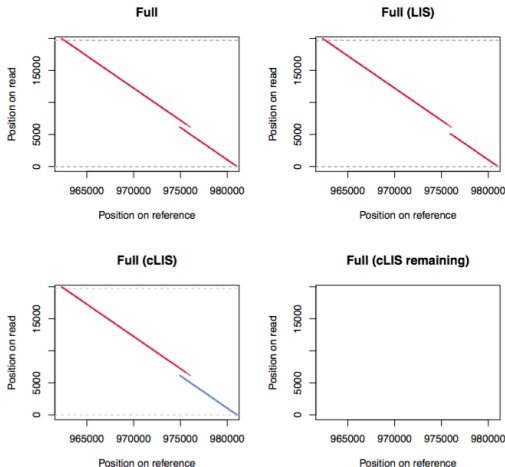
# Candidate search example (simulated)

925758/52



# Candidate search example (simulated)

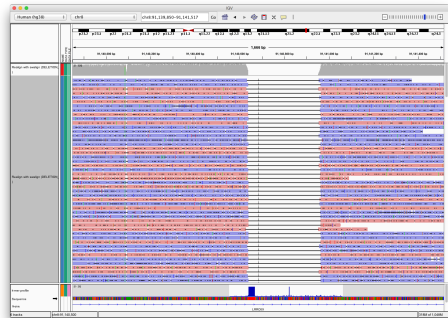
135516/92



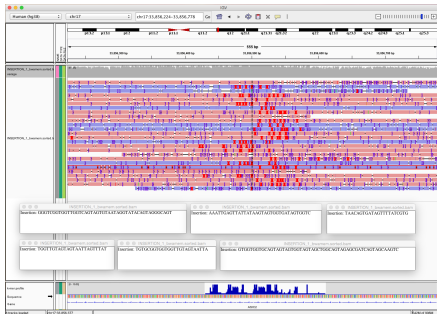
**CIBIV MFPL**  
**Center for Integrative  
Bioinformatics Vienna**



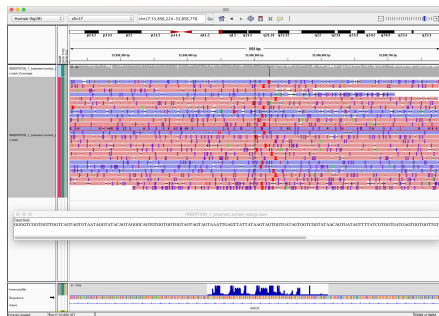
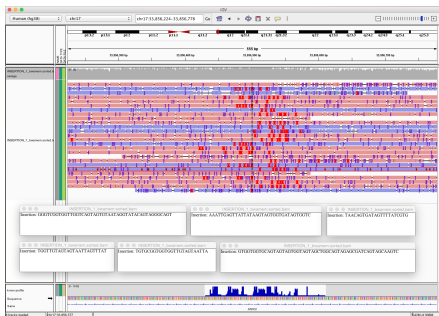
# Results: deletion



# Results: insertion



# Results: insertion





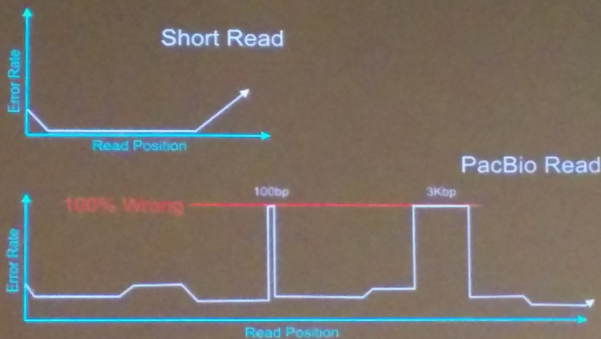
# Results: insertion



# Results: insertion



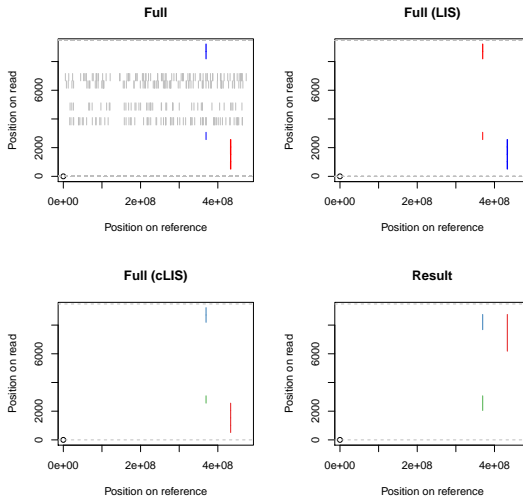
## Error Profile of a Read



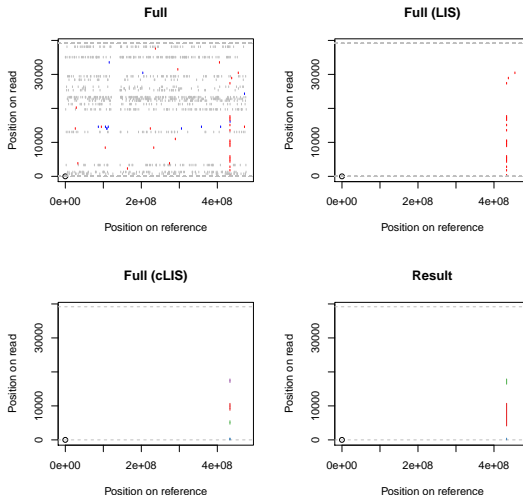
The profile is not computable from instrument QVs  
Completely different for each read

# Error rate: example

276612/961



195160/1498



- Identify segments using cLIS algorithm
- If two segments are within a certain corridor join/connect them
- Parameters: min score for anchors, max number of mapping positions per anchor, allowed distance for cLIS, max allowed corridor

Other approaches:

- k-mer search that tolerates indel errors
- HISAT approach: use large k-mers to find approximate mapping region. Use very small k-mers (4-8 bp) to perform CMR search only on this small part of the genome

- Is it possible to compute alignments for full 10 - 80 kb reads in a reasonable time? If yes, how?
- How to handle inversions properly? Especially small ones.
- Is there a simple strategy to get candidate segments from anchors that can handle the read parts with high error rates?
- How to handle structural variations that are too big even for long reads? Local alignment will always skip on part of the read?

- Is it possible to compute alignments for full 10 - 80 kb reads in a reasonable time? If yes, how?
- How to handle inversions properly? Especially small ones.
- Is there a simple strategy to get candidate segments from anchors that can handle the read parts with high error rates?
- How to handle structural variations that are too big even for long reads? Local alignment will always skip on part of the read?



- Is it possible to compute alignments for full 10 - 80 kb reads in a reasonable time? If yes, how?
- How to handle inversions properly? Especially small ones.
- Is there a simple strategy to get candidate segments from anchors that can handle the read parts with high error rates?
- How to handle structural variations that are too big even for long reads? Local alignment will always skip on part of the read?

- Is it possible to compute alignments for full 10 - 80 kb reads in a reasonable time? If yes, how?
- How to handle inversions properly? Especially small ones.
- Is there a simple strategy to get candidate segments from anchors that can handle the read parts with high error rates?
- How to handle structural variations that are too big even for long reads? Local alignment will always skip on part of the read?

We don't need the optimal alignment!

...

Heuristics to speed up alignment step?

...

Compute alignments for short parts of the reads and join them?

...

Switch to seed and extend? How to still capture structural variations?

...

Better way of identifying anchors and candidate segments?

...

Ideas based on BLAST like programs?

Thanks to:

- Fritz
- Arndt
- Mike
- Maria