GSNAP: Fast and SNP-tolerant detection of complex variants and splicing in short reads

by Thomas D. Wu and Serban Nacu

Matt Huska
Freie Universität Berlin

Computational Methods for High-Throughput Omics Data, WS 2011

Outline

Outline

Outline

Outline

Outline

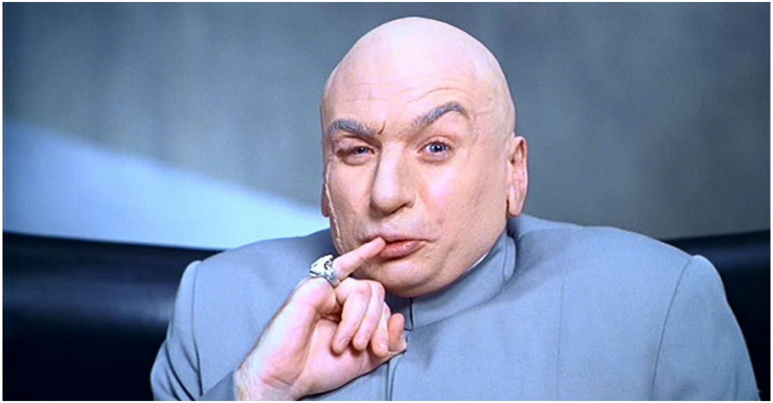
Motivation

- New technology allows us to sequence more reads in shorter time. Increasing at an incredible rate with no signs of slowing down.

Motivation

- New technology allows us to sequence more reads in shorter time. Increasing at an incredible rate with no signs of slowing down.
- "Why should we be happy with millions of reads, when we can have...

Motivation



. . .billions?"

Motivation: Why do we need another read mapping algorithm?

- The reads themselves are also getting longer. Longer reads = higher probability for complex variants within a read.

Motivation: Why do we need another read mapping algorithm?

- The reads themselves are also getting longer. Longer reads = higher probability for complex variants within a read.
- Current (Feb 2010) read mappers tend to either be very fast (BWA, Bowtie, SOAP2) or sensitive to variants (SOAP)

Motivation: Why do we need another read mapping algorithm?

- The reads themselves are also getting longer. Longer reads = higher probability for complex variants within a read.
- Current (Feb 2010) read mappers tend to either be very fast (BWA, Bowtie, SOAP2) or sensitive to variants (SOAP)
- GSNAP is intended to be fast and able to handle complex variants

Outline

Key Features of GSNAP

- Can handle short and long insertions and deletions

Key Features of GSNAP

- Can handle short and long insertions and deletions
- Detects short and long distance splicing (including interchromosomal)

Key Features of GSNAP

- Can handle short and long insertions and deletions
- Detects short and long distance splicing (including interchromosomal)
    - user-provided database of splice sites (eg. RefSeq)

Key Features of GSNAP

- Can handle short and long insertions and deletions
- Detects short and long distance splicing (including interchromosomal)
  - user-provided database of splice sites (eg. RefSeq)
  - probabilistic model

Key Features of GSNAP

- Can handle short and long insertions and deletions
- Detects short and long distance splicing (including interchromosomal)
  - user-provided database of splice sites (eg. RefSeq)
  - probabilistic model
- SNP tolerant (given a user-provided database eg. dbSNP)

Key Features of GSNAP

- Can handle short and long insertions and deletions
- Detects short and long distance splicing (including interchromosomal)
  - user-provided database of splice sites (eg. RefSeq)
  - probabilistic model
- SNP tolerant (given a user-provided database eg. dbSNP)
- Can align bisulfite-treated DNA (for studying methlyation state)

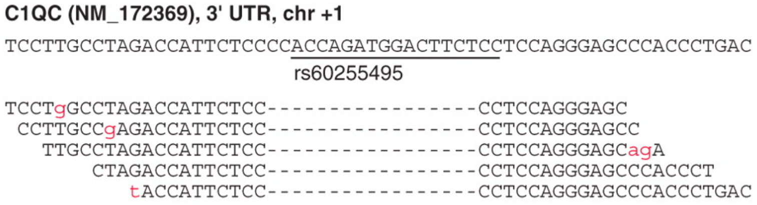
Key Features of GSNAP

- Can handle short and long insertions and deletions
- Detects short and long distance splicing (including interchromosomal)
  - user-provided database of splice sites (eg. RefSeq)
  - probabilistic model
- SNP tolerant (given a user-provided database eg. dbSNP)
- Can align bisulfite-treated DNA (for studying methlyation state)
- Still pretty fast

Outline

Long deletion of 17nt plus mismatches

- 17nt deletion matching an entry in dbSNP, including mismatches:



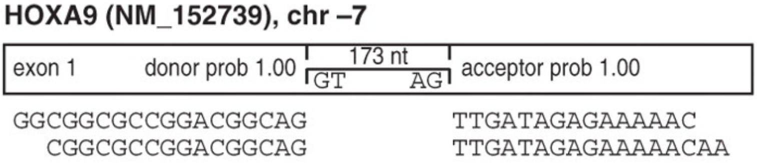
C1QC (NM_172369), 3' UTR, chr +1

```
TCCTTGCCTAGACCATTCTCCCCACCAGATGGACTTCTCCTCCAGGGAGCCCACCCTGAC
                       rs60255495

TCCTgGCCTAGACCATTCTCC-----------------CCTCCAGGGAGC
 CCTTGCCgAGACCATTCTCC-----------------CCTCCAGGGAGCC
   TTGCCTAGACCATTCTCC-----------------CCTCCAGGGAGCagA
       CTAGACCATTCTCC-----------------CCTCCAGGGAGCCCACCCT
          tACCATTCTCC-----------------CCTCCAGGGAGCCCACCCTGAC
```

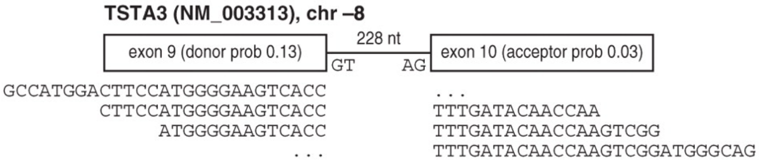
Splicing identified using the probabilistic model

- An intron within exon 1 of HOXA9. Is also experimentally supported.



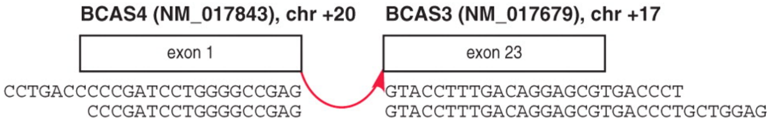
Splicing identified using a database of known splice sites

- Splicing sites identified despite having low probabilistic scores.


TSTA3 (NM_003313), chr –8
exon 9 (donor prob 0.13)
228 nt
GT
AG
exon 10 (acceptor prob 0.03)
GCCATGGACTTCCATGGGAAGTCACC
CTTCCATGGGAAGTCACC
ATGGGAAGTCACC
...
...
TTTGATACAACCAA
TTTGATACAACCAAGTCGG
TTTGATACAACCAAGTCGGATGGGCAG

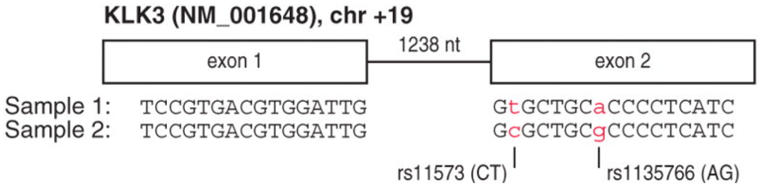
Interchromosomal splicing (gene fusion)

- Splicing between BCAS4 (chr 20) and BCAS3 (chr 17).


BCAS4 (NM_017843), chr +20
BCAS3 (NM_017679), chr +17
exon 1
exon 23
CCTGACCCCCGATCCTGGGCCGAG
CCCGATCCTGGGCCGAG
GTACCTTTGACAGGAGCGTGACCCT
GTACCTTTGACAGGAGCGTGACCCTGCTGGAG

SNP-tolerant alignment

- SNP-tolerance allows both genotypes to align well


KLK3 (NM_001648), chr +19
exon 1
1238 nt
exon 2
Sample 1: TCCGTGACGTGGATTG GtGCTGCaCCCCTCATC
Sample 2: TCCGTGACGTGGATTG GcGCTGCgCCCCTCATC
rs11573 (CT)
rs1135766 (AG)

Outline

The Algorithm in Brief

- Index the genome using a hash table

The Algorithm in Brief

- Index the genome using a hash table
- Break up short reads into shorter elements and look each up in hash table

The Algorithm in Brief

- Index the genome using a hash table
- Break up short reads into shorter elements and look each up in hash table
- Look at resulting position lists for each element and see if they support a common target location and have a reasonable number of mismatches

The Algorithm in Brief

- Index the genome using a hash table
- Break up short reads into shorter elements and look each up in hash table
- Look at resulting position lists for each element and see if they support a common target location and have a reasonable number of mismatches
- Verify the number of mismatches by checking the whole read against the reference

Outline

Hash Table Space Requirements

- Hashing every overlapping oligomer in the entire reference sequence would take too much memory (approx. 14GB for the human genome)

Hash Table Space Requirements

- Hashing every overlapping oligomer in the entire reference sequence would take too much memory (approx. 14GB for the human genome)
- We only hash 12mers every 3nt (approx. 4GB)

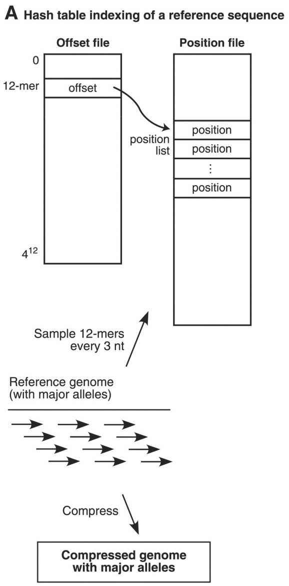
Hash Table Space Requirements

- Hashing every overlapping oligomer in the entire reference sequence would take too much memory (approx. 14GB for the human genome)
- We only hash 12mers every 3nt (approx. 4GB)
- Optional SNP-tolerance only adds a small amount to total memory requirements (3.8 GB → 4.0 GB)

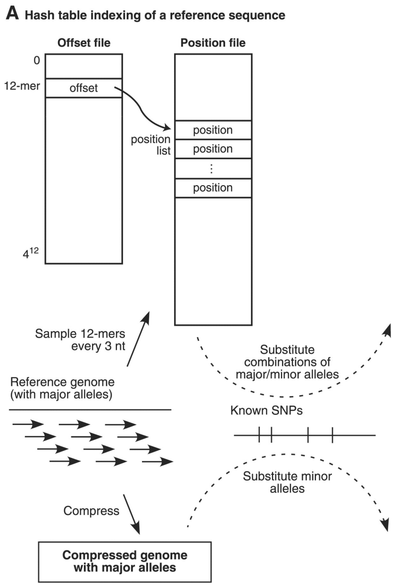
Hash Table Space Requirements

- Hashing every overlapping oligomer in the entire reference sequence would take too much memory (approx. 14GB for the human genome)
- We only hash 12mers every 3nt (approx. 4GB)
- Optional SNP-tolerance only adds a small amount to total memory requirements (3.8 GB $\rightarrow$ 4.0 GB)
- Entire table only needs to be in memory during construction. Afterwards it is mmap'd and only part is loaded into memory.

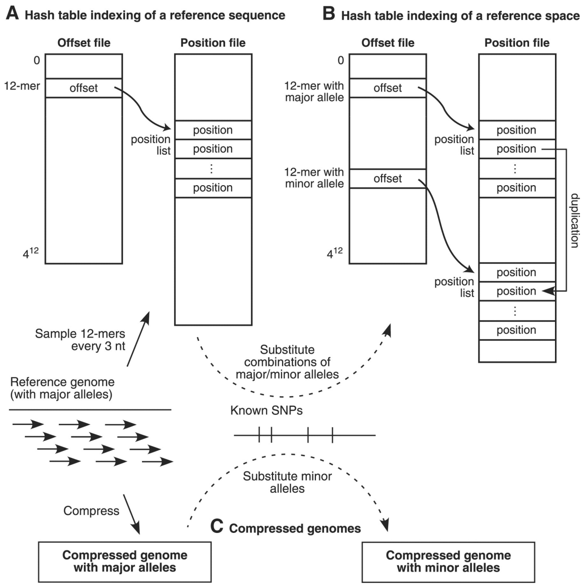
Hashing the Reference Genome


A Hash table indexing of a reference sequence
Offset file
Position file
0
12-mer
offset
4^{12}
position list
position
position
⋮
position
Sample 12-mers every 3 nt
Reference genome (with major alleles)
Compress
Compressed genome with major alleles

Including SNPs

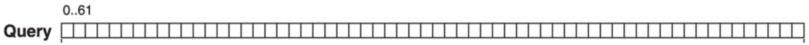

A Hash table indexing of a reference sequence
Offset file
Position file
0
12-mer
offset
4^12
position list
position
position
⋮
position
Sample 12-mers every 3 nt
Reference genome (with major alleles)
Substitute combinations of major/minor alleles
Known SNPs
Substitute minor alleles
Compress
Compressed genome with major alleles

Resulting Reference "Space"

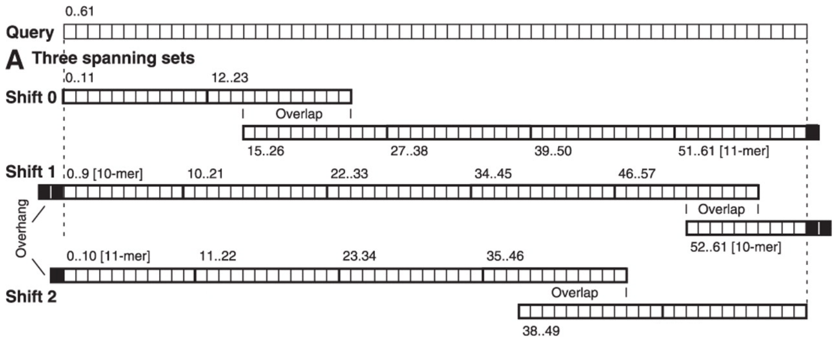


Outline

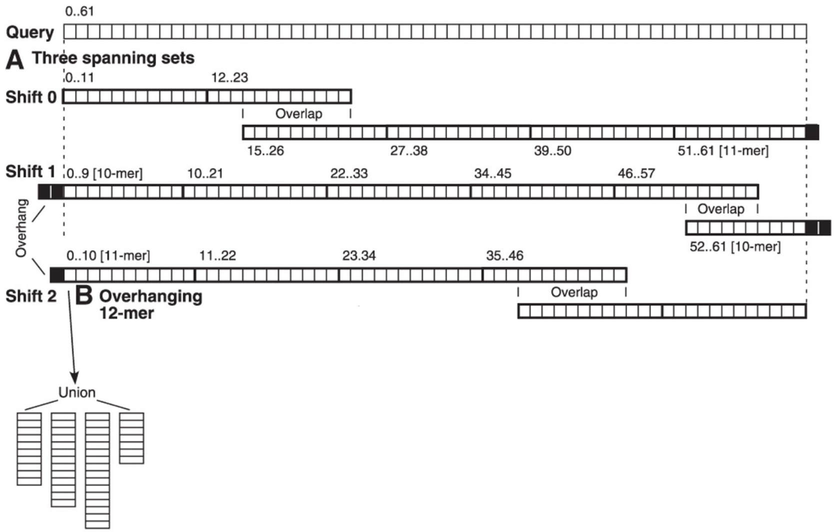
Spanning Set Generation and Filtering



Spanning Set Generation and Filtering

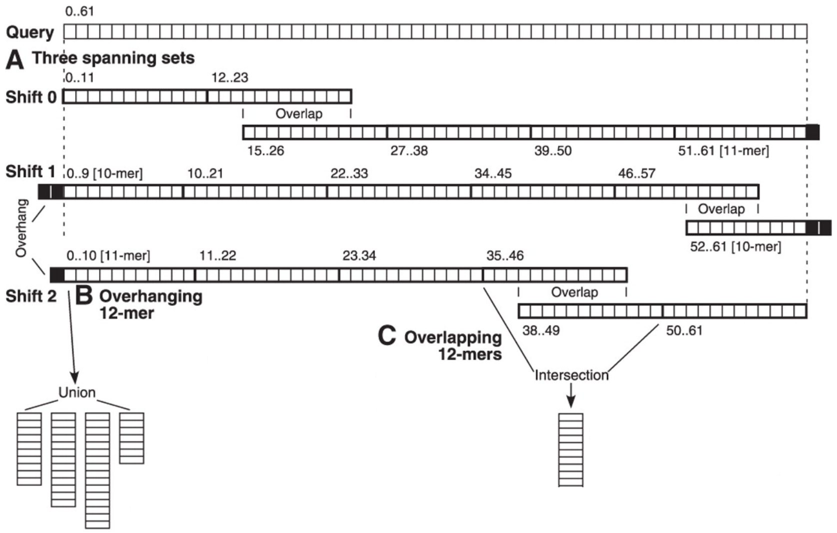


Spanning Set Generation and Filtering



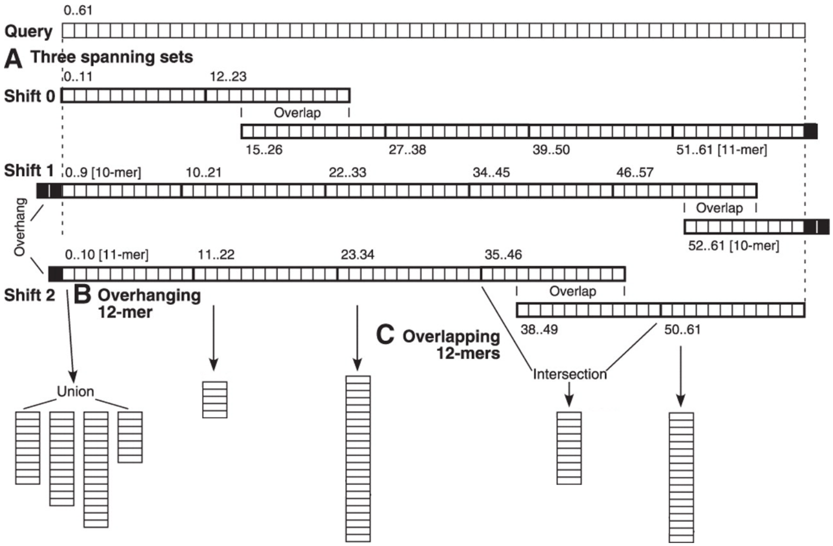
Freie Universität Berlin

Spanning Set Generation and Filtering

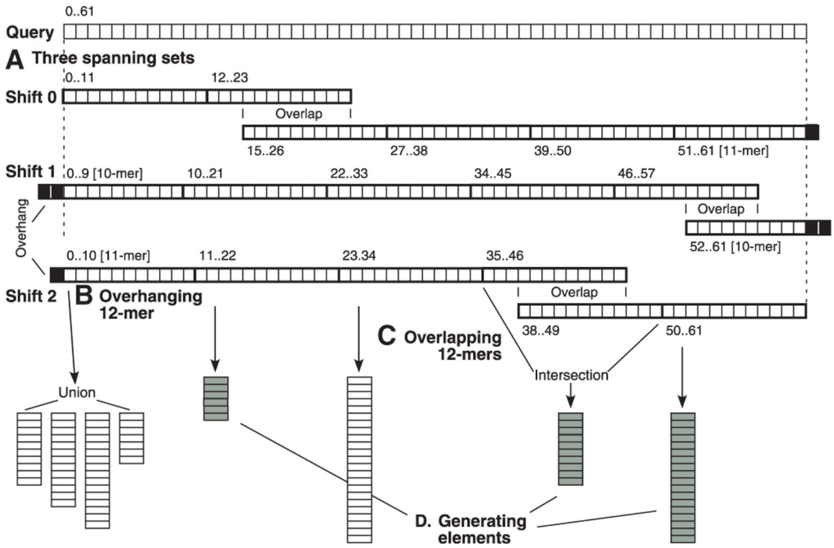


Freie Universität Berlin

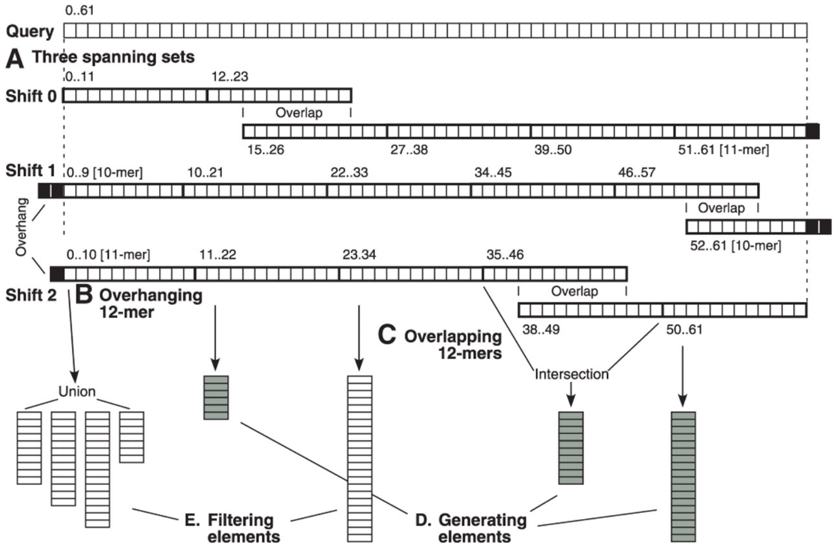
Spanning Set Generation and Filtering



Spanning Set Generation and Filtering



Spanning Set Generation and Filtering



Spanning Set Generation and Filtering

- Generating elements: used to find supporting candidate locations
  - Uses a *multiway merging* procedure (Knuth TAOCP Vol 3)
  - Slow: linear on the sum of the list lengths. $O(l \log n)$ runtime where n is the number of position lists and l is the sum of their lengths.

Spanning Set Generation and Filtering

- Generating elements: used to find supporting candidate locations
  - Uses a *multiway merging* procedure (Knuth TAOCP Vol 3)
  - Slow: linear on the sum of the list lengths. $O(l \log n)$ runtime where n is the number of position lists and l is the sum of their lengths.
- Filtering elements: filter the candidate locations found using the generating elements
  - Fast: uses a binary search. $O(\log l_i)$ for position list i

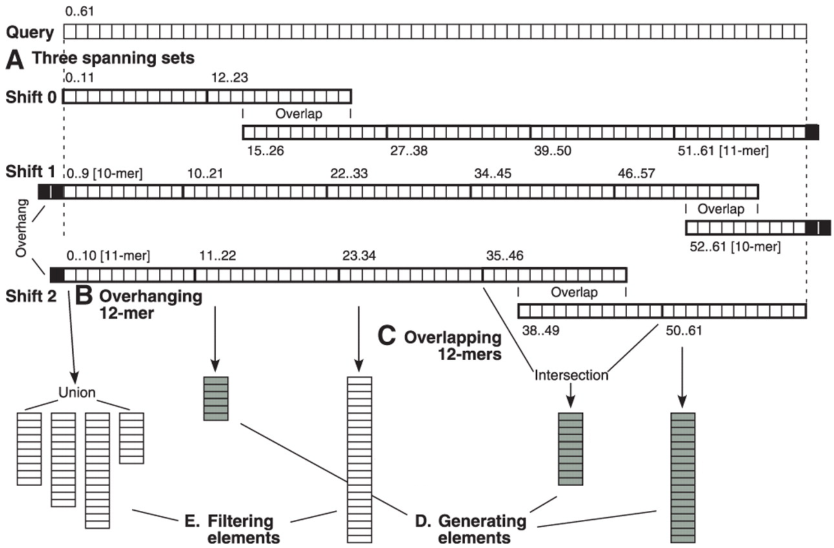
Spanning Set Generation and Filtering

- Generating elements: used to find supporting candidate locations
  - Uses a *multiway merging* procedure (Knuth TAOCP Vol 3)
  - Slow: linear on the sum of the list lengths. $O(l \log n)$ runtime where n is the number of position lists and l is the sum of their lengths.
- Filtering elements: filter the candidate locations found using the generating elements
  - Fast: uses a binary search. $O(\log l_i)$ for position list i
- Choose the elements with the shortest position lists as generating elements, and the longer ones as filtering elements.

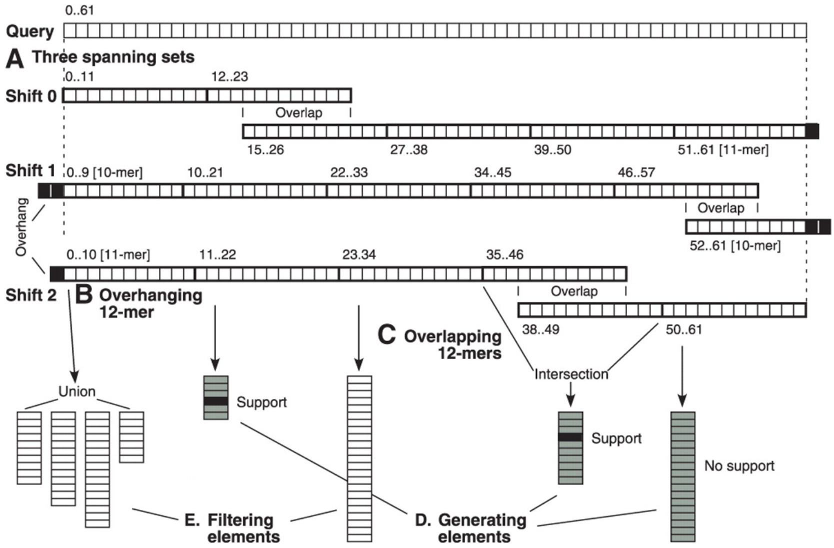
Spanning Set Generation and Filtering

- Generating elements: used to find supporting candidate locations
  - Uses a *multiway merging* procedure (Knuth TAOCP Vol 3)
  - Slow: linear on the sum of the list lengths. $O(l \log n)$ runtime where n is the number of position lists and l is the sum of their lengths.
- Filtering elements: filter the candidate locations found using the generating elements
  - Fast: uses a binary search. $O(\log l_i)$ for position list i
- Choose the elements with the shortest position lists as generating elements, and the longer ones as filtering elements.
- They choose $K + 2$ generating sets, where K is the constraint score (= max number of mismatches)

Spanning Set Generation and Filtering

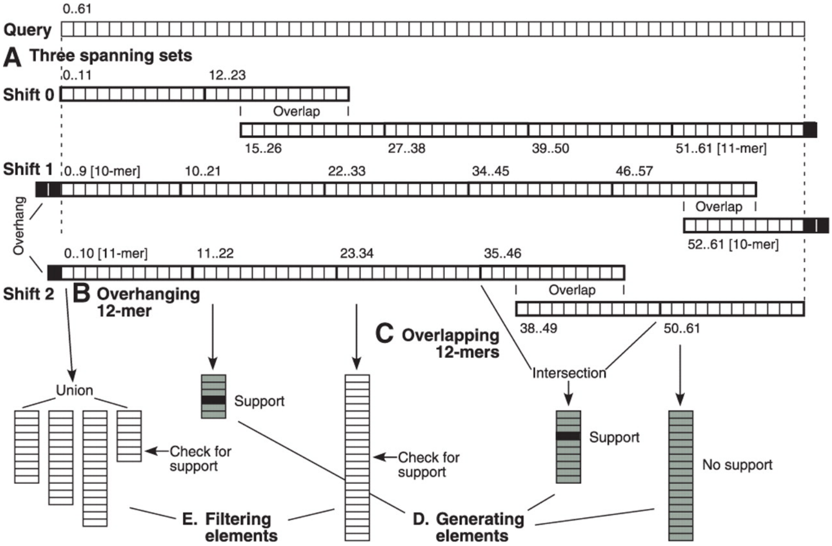


Spanning Set Generation and Filtering



Freie Universität Berlin

Spanning Set Generation and Filtering



Spanning Set Generation and Filtering

- Problem: The spanning set method can only be used to detect a limited number of mismatches.

Spanning Set Generation and Filtering

- Problem: The spanning set method can only be used to detect a limited number of mismatches.
- If we want to allow K matches, for $K > 1$, then we need at least $(K + 2)$ generating elements. The read length L limits the number of elements N, and the spanning set elements are non-overlapping in all three shifts if $L = 10(mod 12)$, so $N \leq \lfloor (L + 2)/12 \rfloor$.

Spanning Set Generation and Filtering

- Problem: The spanning set method can only be used to detect a limited number of mismatches.
- If we want to allow K matches, for $K > 1$, then we need at least $(K+2)$ generating elements. The read length L limits the number of elements N, and the spanning set elements are non-overlapping in all three shifts if $L = 10(mod 12)$, so $N \leq \lfloor (L+2)/12 \rfloor$.
- So we need to satisfy $N > (K+2)$, and as such we can only apply the spanning set method when $K \leq \lfloor (L+2)/12 \rfloor - 2$ for $L \geq 34$.

Spanning Set Generation and Filtering

- Problem: The spanning set method can only be used to detect a limited number of mismatches.
- If we want to allow K matches, for $K > 1$, then we need at least $(K + 2)$ generating elements. The read length L limits the number of elements N, and the spanning set elements are non-overlapping in all three shifts if $L = 10(mod 12)$, so $N \leq \lfloor (L + 2)/12 \rfloor$.
- So we need to satisfy $N > (K + 2)$, and as such we can only apply the spanning set method when $K \leq \lfloor (L + 2)/12 \rfloor - 2$ for $L \geq 34$.
- For reads of length 100 (Illumina), the we can allow a maximum of 6 mismatches.
- For reads of length 400 (454), the we can allow a maximum of 31 mismatches.

Spanning Set Generation and Filtering

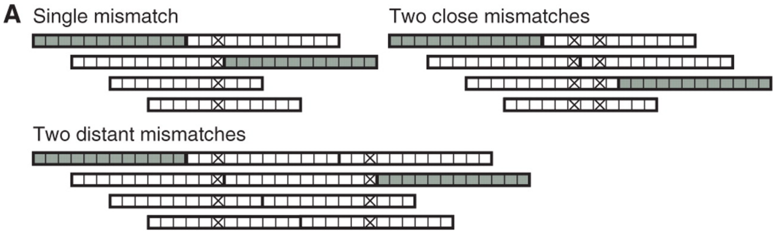
- Problem: The spanning set method can only be used to detect a limited number of mismatches.
- If we want to allow K matches, for $K > 1$, then we need at least $(K + 2)$ generating elements. The read length L limits the number of elements N, and the spanning set elements are non-overlapping in all three shifts if $L = 10(mod 12)$, so $N \leq \lfloor (L+2)/12 \rfloor$.
- So we need to satisfy $N > (K + 2)$, and as such we can only apply the spanning set method when $K \leq \lfloor (L+2)/12 \rfloor - 2$ for $L \geq 34$.
- For reads of length 100 (Illumina), the we can allow a maximum of 6 mismatches.
- For reads of length 400 (454), the we can allow a maximum of 31 mismatches.
- If we want to allow larger numbers of mismatches or the same number of mismatches in shorter reads, we need to use another method...

Outline

Complete set generation and filtering

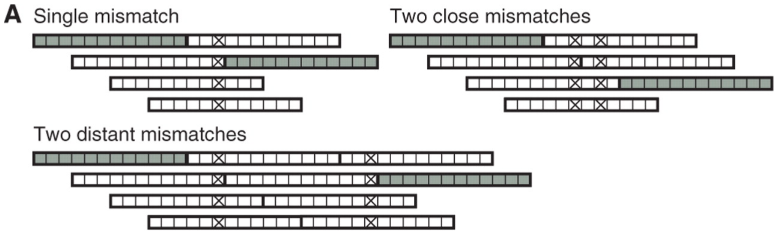
- Uses the complete set of overlapping 12mers.
- Works for any number of mismatches as long as read and target have ≥ 14 consecutive matches (12mer out of phase by up to two bases)
- Exhaustive for $K \leq \lfloor L/14 \rfloor - 1$

Complete set generation and filtering



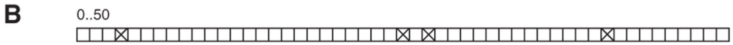
Freie Universität Berlin

Complete set generation and filtering

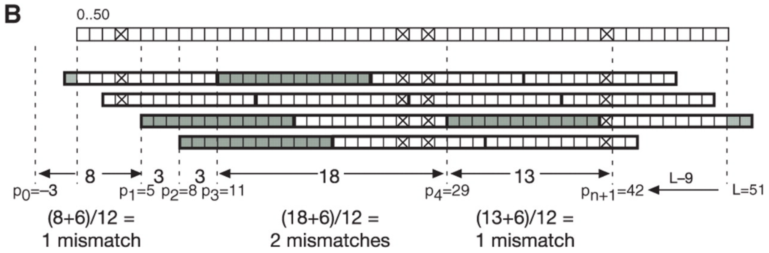


- Lower bound on mismatches: $\lfloor (\Delta p + 6)/12 \rfloor$
  where Δp is the distance between start locations of consecutive supporting 12mers.

Complete set example



Complete set generation and filtering

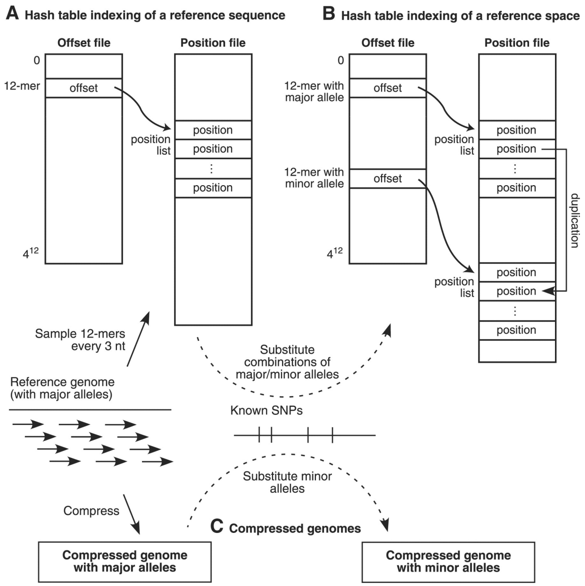


Outline

Verification of Candidate Regions

- The spanning set and complete set methods generate candidate regions for which we know a **lower bound** on the number of mismatches.
- These regions need to be verified to check the **exact number** of mismatches.

Remember: Resulting Reference "Space"



Compressed Genome

- Text usually stored as 8 bit characters

Compressed Genome

- Text usually stored as 8 bit characters
- Because we have a reduced alphabet the reference is stored as 3 bits per character: 2 bits for the nt + a flag

Compressed Genome

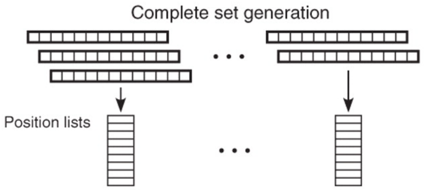
- Text usually stored as 8 bit characters
- Because we have a reduced alphabet the reference is stored as 3 bits per character: 2 bits for the nt + a flag
  - Flag in major-allele genome: indicates unknown or ambiguous nt
  - Flag in minor-allele genome: indicates a SNP

Verification of Candidate Regions

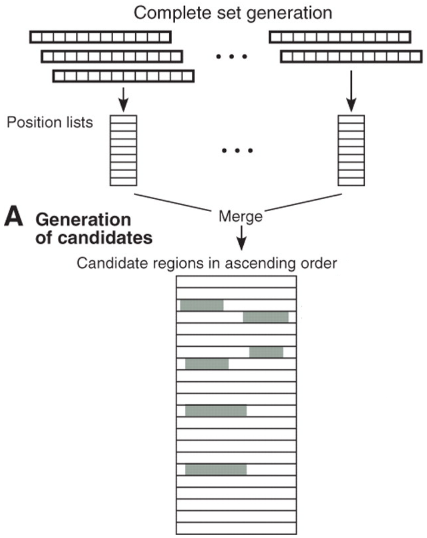
- Query sequence converted to the same compressed representation as the reference
- Shifted into position and bitwise XOR combined with the major- and minor-allele genomes separately
- Resulting arrays are bitwise AND'd, so mismatches at a SNP only occur if both alleles do not match

Outline

Detecting Insertions and Deletions

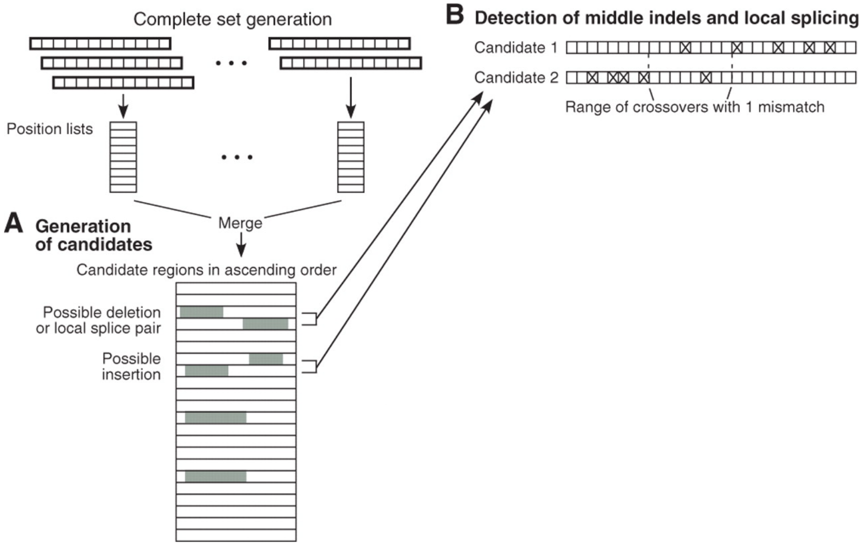


Detecting Insertions and Deletions

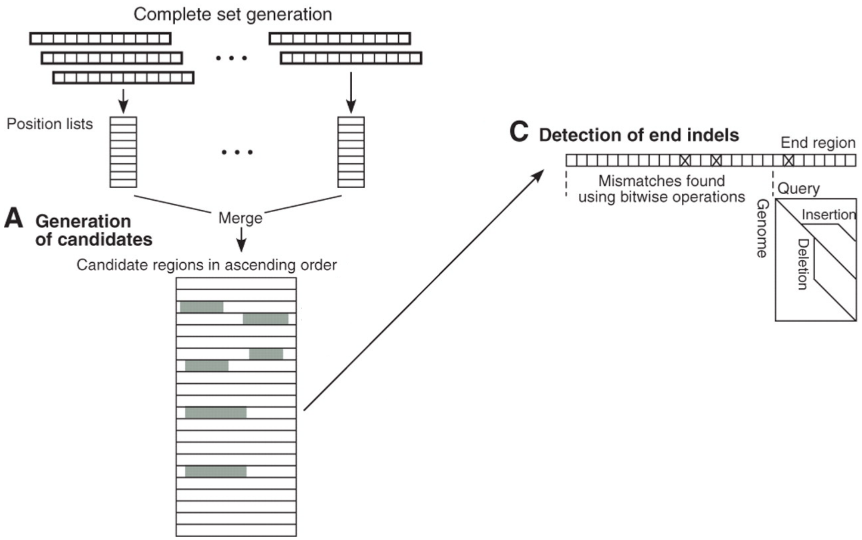


Complete set generation
Position lists
A Generation of candidates
Merge
Candidate regions in ascending order

Detecting Insertions and Deletions


Complete set generation
Position lists
A Generation of candidates
Merge
Candidate regions in ascending order
Possible deletion or local splice pair
Possible insertion
B Detection of middle indels and local splicing
Candidate 1
Candidate 2
Range of crossovers with 1 mismatch

Detecting Insertions and Deletions



Complete set generation
Position lists
A Generation of candidates
Merge
Candidate regions in ascending order
C Detection of end indels
End region
Mismatches found using bitwise operations
Query
Genome
Insertion
Deletion

Outline

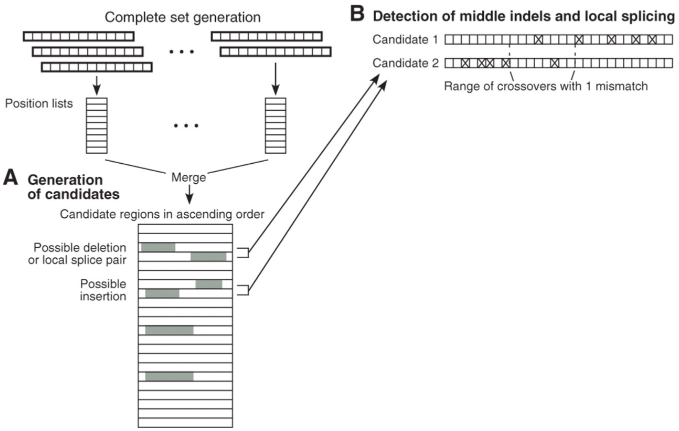
Detecting Splice Junctions

- Known splice sites: user-provided database
- Novel splice sites: maximum entropy probabilistic model from Yeo and Burge, 2004

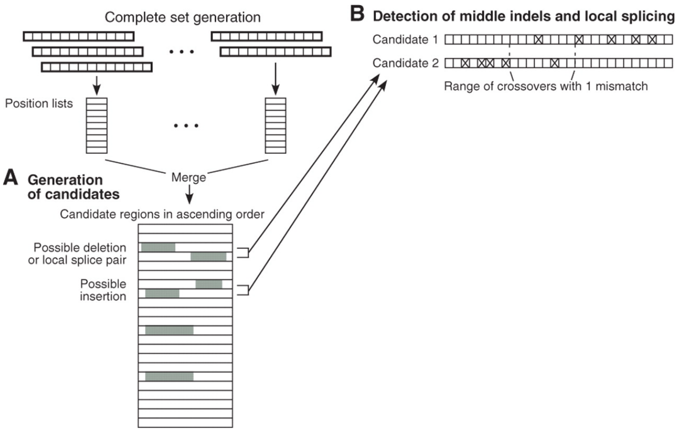
Detecting Splice Junctions

- Short-distance splice sites are on the same chromosome and < some distance apart (default: 200,000 nt)
- Method similar to the one we used to find middle deletions earlier. . .

Detecting Splice Junctions


Complete set generation
...
Position lists
...
Merge
A Generation of candidates
Candidate regions in ascending order
Possible deletion or local splice pair
Possible insertion
B Detection of middle indels and local splicing
Candidate 1
Candidate 2
Range of crossovers with 1 mismatch

Detecting Splice Junctions

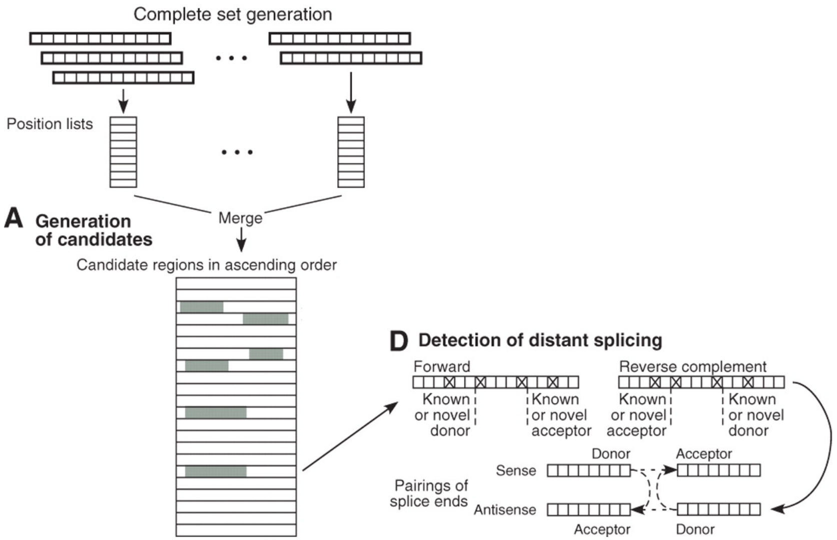

Complete set generation
Position lists
A Generation of candidates
Merge
Candidate regions in ascending order
Possible deletion or local splice pair
Possible insertion
B Detection of middle indels and local splicing
Candidate 1
Candidate 2
Range of crossovers with 1 mismatch

- Crossover area is then searched for donor or acceptor sites (either known or novel with high probability).

Detecting Splice Junctions

- Long-distance splice sites can be on different chromosomes
- Require higher probability scores for novel splice sites than short-distance splice sites
- Candidates with matching breakpoints on the read are matched

Detecting Splice Junctions



Detecting Splice Junctions

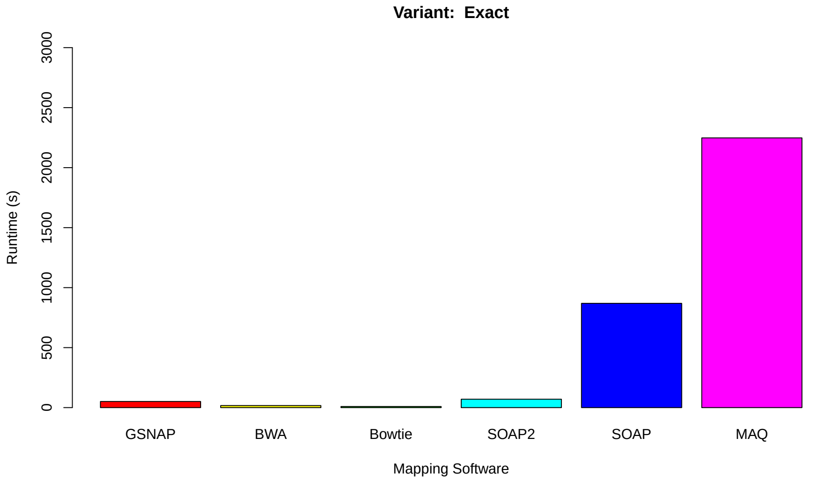
- If both splice sites can not be found then GSNAP will return one site (a "half-intron")

Outline

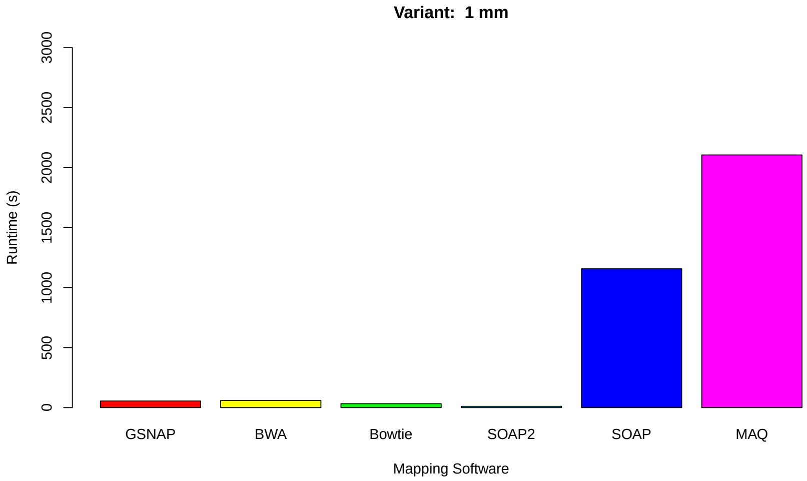
Simulated Reads

- Runtime comparison between GSNAP and other alignment tools (for 100,000 reads)
- Simulated increasingly complicated variants
  - exact matches only
  - 1 - 3 mismatches
  - short insertions and deletions
  - longer insertions and deletions

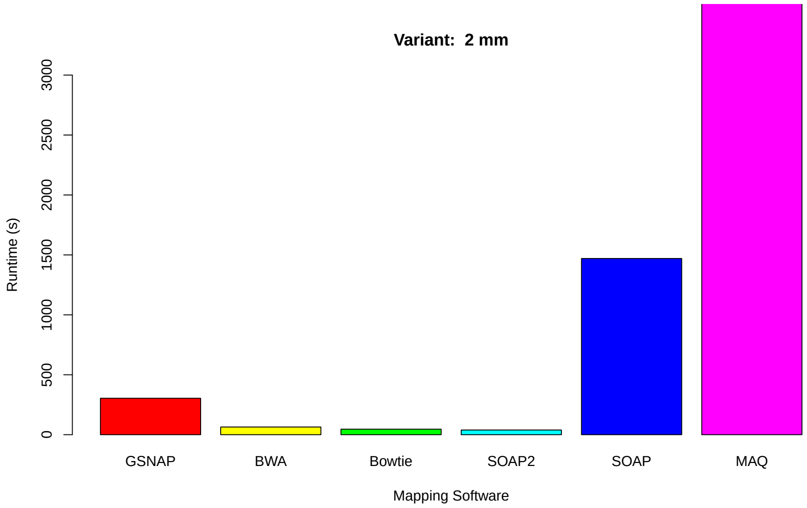
Simulated Reads: 36nt



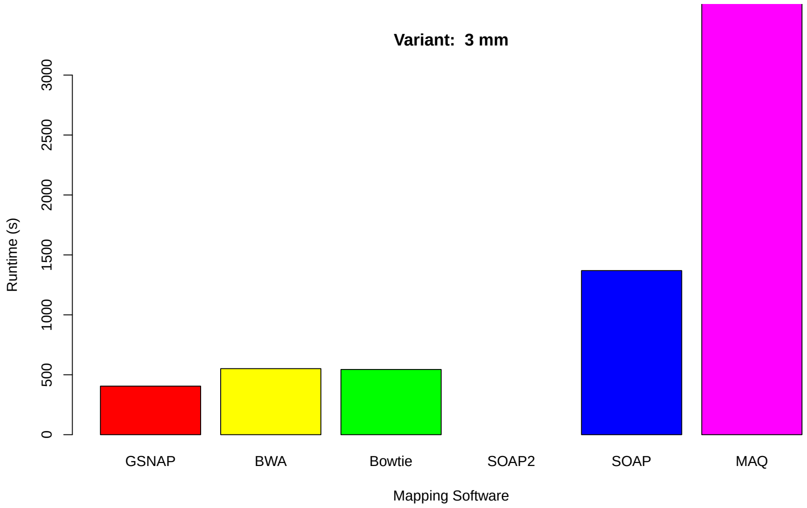
Simulated Reads: 36nt



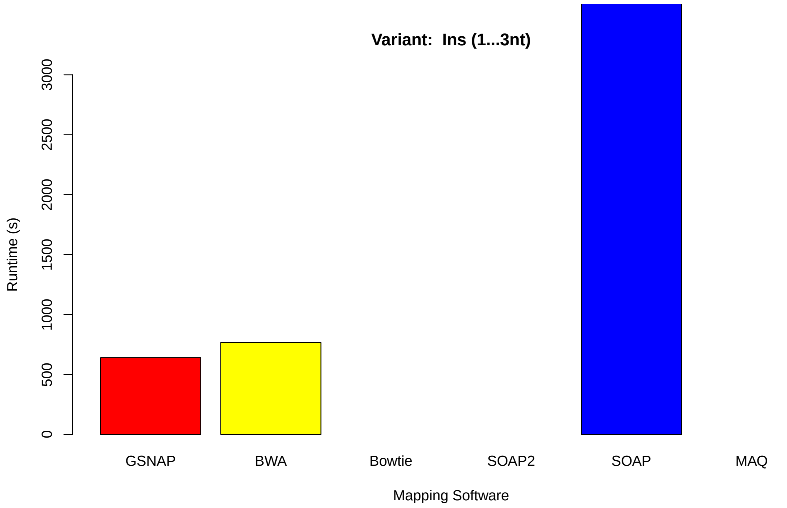
Simulated Reads: 36nt



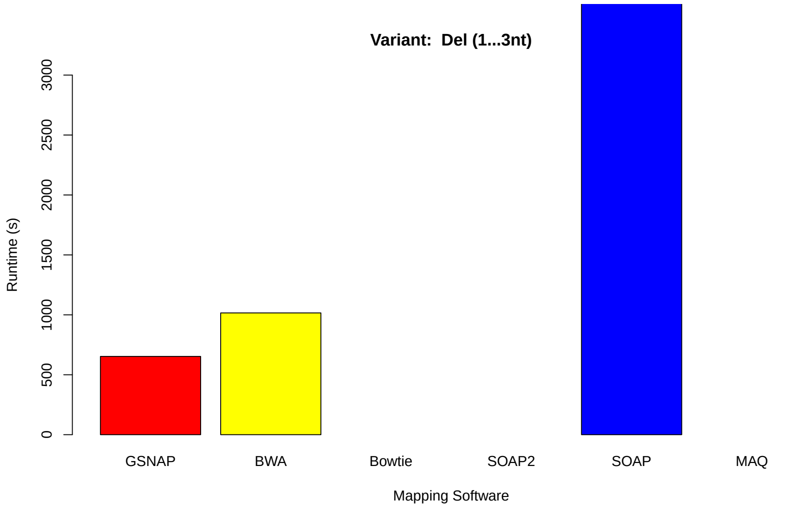
Simulated Reads: 36nt



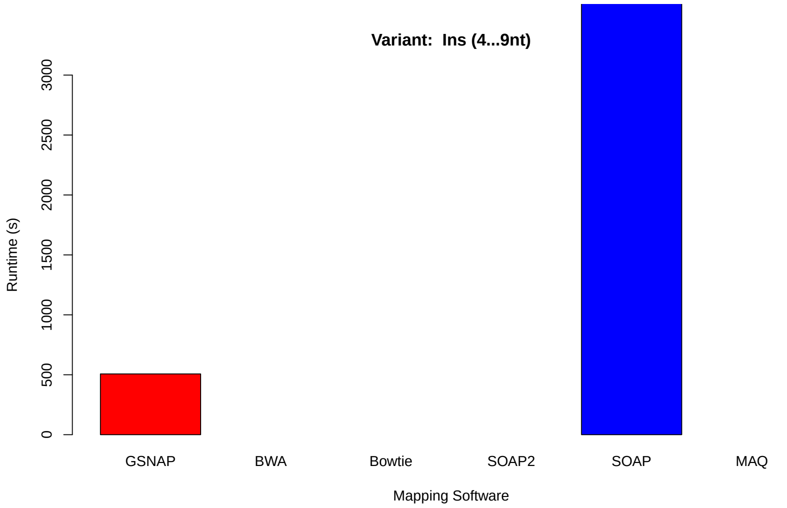
Simulated Reads: 36nt



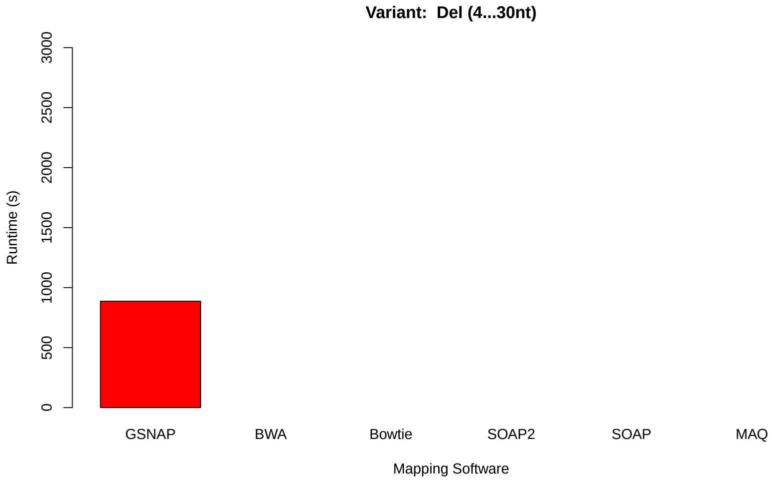
Simulated Reads: 36nt



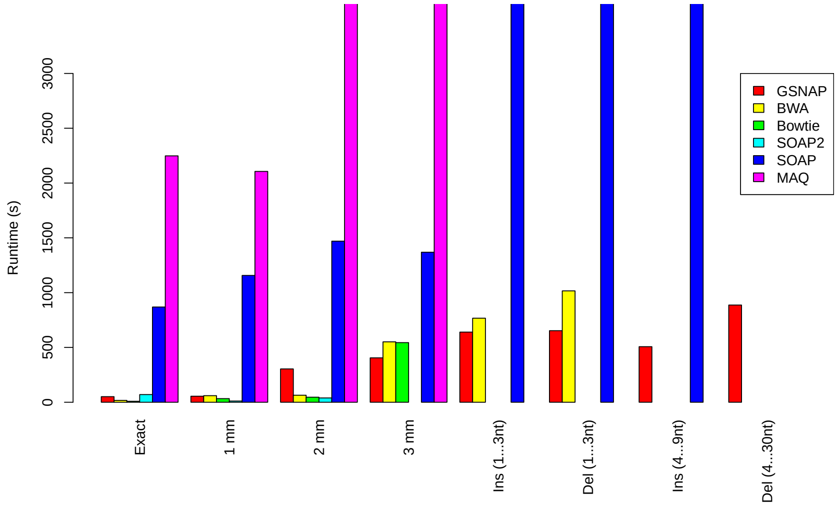
Simulated Reads: 36nt



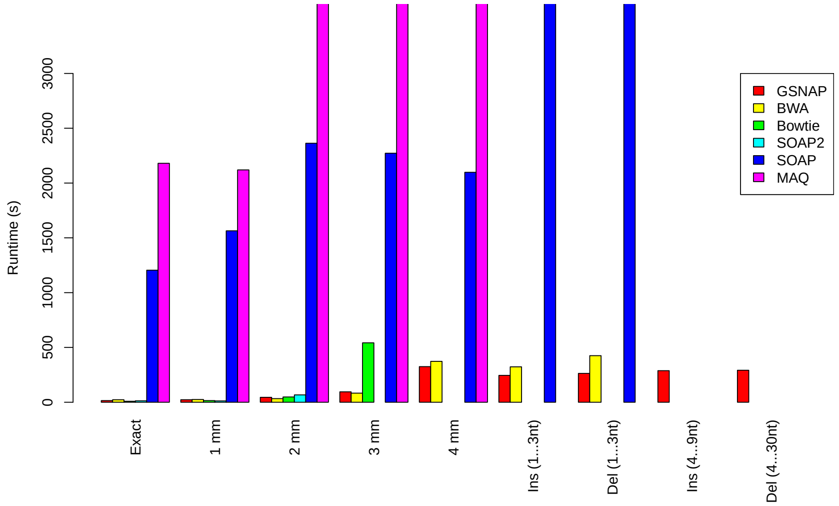
Simulated Reads: 36nt



Simulated Reads: 36nt

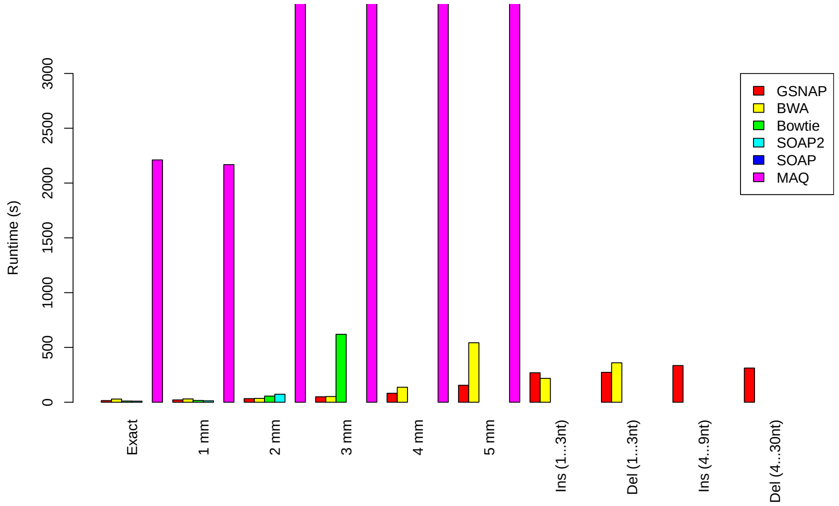


Simulated Reads: 70nt



Freie Universität Berlin

Simulated Reads: 100nt



Simulated Reads: Percent misses (on unique reads)

- Most algorithms were able to perfectly map unique reads, with the following notable exceptions:

Simulated Reads: Percent misses (on unique reads)

- Most algorithms were able to perfectly map unique reads, with the following notable exceptions:
- GSNAP: 12% misses for 36nt reads with 3 mismatches

Simulated Reads: Percent misses (on unique reads)

- Most algorithms were able to perfectly map unique reads, with the following notable exceptions:
- GSNAP: 12% misses for 36nt reads with 3 mismatches
- SOAP: 15% misses for 36nt reads with 3 mismatches, around 5% for 1-3nt indels

Simulated Reads: Percent misses (on unique reads)

- Most algorithms were able to perfectly map unique reads, with the following notable exceptions:
- GSNAP: 12% misses for 36nt reads with 3 mismatches
- SOAP: 15% misses for 36nt reads with 3 mismatches, around 5% for 1-3nt indels
- BWA: 1-5% misses in 1-3nt indels

Memory Requirements

- GSNAP: should have access to 5 GB of memory, otherwise it will run slowly
- BWA: 2.2 GB
- Bowtie: 1.1 GB (exact matches) or 2.2 GB (allowing mismatches)
- MAQ: 302 MB
- SOAP: 14 GB
- SOAP2: unknown ("only provided as a binary and did not have the required compile time flag")

Outline

Transcriptional Reads

- Only looked at the effect of splicing, indels and SNP tolerance

Transcriptional Reads

- Only looked at the effect of splicing, indels and SNP tolerance
- Including known splicing information → increased yield approx. 8%

Transcriptional Reads

- Only looked at the effect of splicing, indels and SNP tolerance
- Including known splicing information → increased yield approx. 8%
- Including SNP tolerance → minor increase in yield (0.5%) but effected about 8% of alignments

Outline

Limitations

- Matching is only exhaustive if there is at least a 14nt contiguous match, otherwise it’s a heuristic

Limitations

- Matching is only exhaustive if there is at least a 14nt contiguous match, otherwise it's a heuristic
- Limited to one indel or splice site per read

Limitations

- Matching is only exhaustive if there is at least a 14nt contiguous match, otherwise it's a heuristic
- Limited to one indel or splice site per read
- Does not use read quality scores

Limitations

- Matching is only exhaustive if there is at least a 14nt contiguous match, otherwise it's a heuristic
- Limited to one indel or splice site per read
- Does not use read quality scores
- Does not work with ABI SOLiD data

TL;DR

- Comparable to other fast read alignment algorithms in terms of speed, but can handle more complex variants and splicing

For Further Reading I

Knuth D.E.
The Art of Computer Programming: Sorting and Searching. Vol 3.
Addison-Wesley, 1973

Thomas D. Wu and Serban Nacu
Fast and SNP-tolerant detection of complex variants and splicing in short reads
Bioinformatics, 2010 Apr 1;26(7):873-81.

Yeo G and Burge CB.
Maximum entropy modeling of short sequence motifs with applications to RNA splicing signals
J Comput Biol. 2004;11(2-3):377-94.

Aligning Paired-End Reads

- Optional

Bisulfite-converted DNA

- Optional