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

Introduction
  Motivation
  GSNAP Features
  Examples of Complex Variant Detection

The Algorithm

Summary

Preprocessing

Method 1: Spanning Set Generation and Filtering
Method 2: Complete Set Generation and Filtering

Verification of Candidate Regions

Detecting Insertions and Deletions

Detecting Splice Junctions

Results

Simulated Reads

Transcriptional Reads

Limitations

Conclusions
Introduction
  Motivation
  GSNAP Features
  Examples of Complex Variant Detection

The Algorithm
  Summary
  Preprocessing
  Method 1: Spanning Set Generation and Filtering
  Method 2: Complete Set Generation and Filtering
  Verification of Candidate Regions
  Detecting Insertions and Deletions
  Detecting Splice Junctions
Outline

Introduction
- Motivation
- GSNAP Features
- Examples of Complex Variant Detection

The Algorithm
- Summary
- Preprocessing
  - Method 1: Spanning Set Generation and Filtering
  - Method 2: Complete Set Generation and Filtering
- Verification of Candidate Regions
- Detecting Insertions and Deletions
- Detecting Splice Junctions

Results
- Simulated Reads
- Transcriptional Reads
- Limitations
Outline

Introduction
  Motivation
  GSNAP Features
  Examples of Complex Variant Detection

The Algorithm
  Summary
  Preprocessing
  Method 1: Spanning Set Generation and Filtering
  Method 2: Complete Set Generation and Filtering
  Verification of Candidate Regions
  Detecting Insertions and Deletions
  Detecting Splice Junctions

Results
  Simulated Reads
  Transcriptional Reads
  Limitations

Conclusions
Outline

Introduction
Motivation
GSNAP Features
Examples of Complex Variant Detection

The Algorithm
Summary
Preprocessing
Method 1: Spanning Set Generation and Filtering
Method 2: Complete Set Generation and Filtering
Verification of Candidate Regions
Detecting Insertions and Deletions
Detecting Splice Junctions

Results
Simulated Reads
Transcriptional Reads
Limitations

Conclusions
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

Introduction
  Motivation
  GSNAP Features
    Examples of Complex Variant Detection

The Algorithm
  Summary
  Preprocessing
    Method 1: Spanning Set Generation and Filtering
    Method 2: Complete Set Generation and Filtering
  Verification of Candidate Regions
  Detecting Insertions and Deletions
  Detecting Splice Junctions

Results
  Simulated Reads
  Transcriptional Reads
  Limitations

Conclusions
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 (e.g., 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 (e.g., RefSeq)
  - probabilistic model
- SNP tolerant (given a user-provided database e.g., dbSNP)
- Can align bisulfite-treated DNA (for studying methylation 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 (e.g., RefSeq)
  - probabilistic model
- SNP tolerant (given a user-provided database e.g., dbSNP)
- Can align bisulfite-treated DNA (for studying methylation state)
- Still pretty fast
Outline

Introduction
  Motivation
  GSNAP Features
  Examples of Complex Variant Detection

The Algorithm
  Summary
  Preprocessing
  Method 1: Spanning Set Generation and Filtering
  Method 2: Complete Set Generation and Filtering
  Verification of Candidate Regions
  Detecting Insertions and Deletions
  Detecting Splice Junctions

Results
  Simulated Reads
  Transcriptional Reads
  Limitations

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

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

TCCTTGCCCTAGACCATTCTCCCACCCAGATGGACTTCTCTGAGAGGAGGCCACCCCTGAC

rs60255495

TCCTgGCCCTAGACCATTCTCC--------------------------CCTCCAGGGAGC
CCTTGCCgAGACCATTCTCC--------------------------CCTCCAGGGAGCC
TTGCTAGACCATTCTCC--------------------------CCTCCAGGGAGCagA
CTAGACCATTCTCC--------------------------CCTCCAGGGAGCCACCCT

tACCATTCTCC--------------------------CCTCCAGGGAGCCACCCCTGAC
Splicing identified using the probabilistic model

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

**HOXA9 (NM_152739), chr -7**

<table>
<thead>
<tr>
<th>Exon 1</th>
<th>Donor prob 1.00</th>
<th>173 nt</th>
<th>Acceptor prob 1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>GGC</td>
<td>GT</td>
<td>AG</td>
<td></td>
</tr>
<tr>
<td>GCCGC</td>
<td>AGGCAG</td>
<td>TTGATAGAGAAAAAC</td>
<td></td>
</tr>
<tr>
<td>CGG</td>
<td>CGGACGCAG</td>
<td>TTGATAGAGAAAAACAA</td>
<td></td>
</tr>
</tbody>
</table>
Splicing identified using a database of known splice sites

- Splicing sites identified despite having low probabilistic scores.

**TSTA3 (NM_003313), chr -8**

<table>
<thead>
<tr>
<th>Exon 9 (donor prob 0.13)</th>
<th>228 nt</th>
<th>Exon 10 (acceptor prob 0.03)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCCATGGGACCTCCATGGGGAAGTCACC</td>
<td>GT</td>
<td>. . .</td>
</tr>
<tr>
<td>CTTCCATGGGGAAGTCACC</td>
<td>AG</td>
<td>TTTGATACAAACCA</td>
</tr>
<tr>
<td>ATGGGGAAGTCACC</td>
<td>. . .</td>
<td>TTTGATACAAACCAAGTCCG</td>
</tr>
<tr>
<td>. . .</td>
<td>. . .</td>
<td>TTTGATACAAACCAAGTGGATGGCAG</td>
</tr>
</tbody>
</table>
Interchromosomal splicing (gene fusion)

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

```
BCAS4 (NM_017843), chr +20
exon 1
CCTGACCCCCGATCCTGGGGCCGAG
CCGATCTCTGGGGCCGAG

BCAS3 (NM_017679), chr +17
exon 23
GTACCTTTTGACAGGAGCGTGACCCCT
GTACCTTTTGACAGGAGCGTGACCCCTGCTGGAG
```
SNP-tolerant alignment

- SNP-tolerance allows both genotypes to align well

![Diagram of SNP-tolerance alignment with examples for Sample 1 and Sample 2. The diagram shows sequences for exon 1 and exon 2 with SNP positions indicated by 'a' and 'g' for the second nucleotide.]
Outline

Introduction
Motivation
GSNAP Features
Examples of Complex Variant Detection

The Algorithm
Summary
Preprocessing
Method 1: Spanning Set Generation and Filtering
Method 2: Complete Set Generation and Filtering
Verification of Candidate Regions
Detecting Insertions and Deletions
Detecting Splice Junctions

Results
Simulated Reads
Transcriptional Reads
Limitations

Conclusions
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

Introduction
  Motivation
  GSNAP Features
  Examples of Complex Variant Detection

The Algorithm
  Summary

Preprocessing
  Method 1: Spanning Set Generation and Filtering
  Method 2: Complete Set Generation and Filtering
  Verification of Candidate Regions
  Detecting Insertions and Deletions
  Detecting Splice Junctions

Results
  Simulated Reads
  Transcriptional Reads
  Limitations

Conclusions
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 → 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
  - Offset

- Position file
  - 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

Sample 12-mers every 3 nt

Reference genome (with major alleles)

Compress

Compressed genome with major alleles

Substitute combinations of major/minor alleles

Known SNPs

Substitute minor alleles

FU Berlin, GSNAP, Omics WS 2011
A Hash table indexing of a reference sequence

Sample 12-mers every 3 nt

Reference genome (with major alleles)

B Hash table indexing of a reference space

Substitute combinations of major/minor alleles

Known SNPs

Substitute minor alleles

C Compressed genomes

Compressed genome with major alleles

Compressed genome with minor alleles

Determination of reference 'space'

FU Berlin, GSNAP, Omics WS 2011
Outline

Introduction
  Motivation
  GSNAP Features
  Examples of Complex Variant Detection

The Algorithm
  Summary
  Preprocessing
    Method 1: Spanning Set Generation and Filtering
    Method 2: Complete Set Generation and Filtering
  Verification of Candidate Regions
  Detecting Insertions and Deletions
  Detecting Splice Junctions

Results
  Simulated Reads
  Transcriptional Reads
  Limitations

Conclusions
Spanning Set Generation and Filtering

0.61

Query
Spanning Set Generation and Filtering

A Three spanning sets

Shift 0

Query

Shift 1

Shift 2

0..11
12..23
0..61

15..26
27..38
39..50
51..61 [11-mer]

10..21
22..33
34..45
46..57

11..22
23..34
35..46
52..61 [10-mer]

Overlap

Overlap

Overlap

0..10 [11-mer]
11..22
23.34
35..46
38..49

0..9 [10-mer]
Spanning Set Generation and Filtering

A. Three spanning sets

- **Query**
  - 0..61

- **Shift 0**
  - .011
  - 12..23
  - Overlap
  - 15..26
  - 27..38
  - 39..50
  - 51..61 [11-mer]

- **Shift 1**
  - .0.9 [10-mer]
  - 10..21
  - 22..33
  - 34..45
  - 46..57
  - Overlap
  - Overhang
  - 0..10 [11-mer]
  - 11..22
  - 23..34
  - 35..46
  - 52..61 [10-mer]

B. Overhanging 12-mer

- Union
Spanning Set Generation and Filtering

A Three spanning sets

Query

Shift 0

Shift 1 [10-mer]

Overhang

Shift 2 Overhanging 12-mer

C Overlapping 12-mers

Union

Overlap

Intersection

0..61

0..11

12..23

15..26

27..38

39..50

51..61 [11-mer]

0..9 [10-mer]

10..21

22..33

34..45

46..57

0..10 [11-mer]

11..22

23..34

35..46

52..61 [10-mer]
Spanning Set Generation and Filtering
Spanning Set Generation and Filtering

A. Three spanning sets

Shift 0

B. Overhanging 12-mer

C. Overlapping 12-mers

D. Generating elements

E. Filtering elements

Union

Overlap

0..11
12..23
15..26
27..38
39..50
51..61 [11-mer]

0.9 [10-mer]
10..21
22..33
34..45
46..57

0..10 [11-mer]
11..22
23..34
35..46
52..61 [10-mer]

38..49
50..61
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 $i$
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

A: Three spanning sets

Shift 0

Shift 1 [10-mer]

Shift 2 Overhanging 12-mer

B: Overhanging 12-mer

C: Overlapping 12-mers

D: Generating elements

E: Filtering elements

Query

Overlap

Overlap

Overlap

Overlap

Union

Intersection
Spanning Set Generation and Filtering

A. Three spanning sets

B. Overhanging 12-mer

C. Overlapping 12-mers

D. Generating elements

E. Filtering elements

Support

Union

Overlap

No support
Spanning Set Generation and Filtering

A. Three spanning sets
- Shift 0
- Shift 1
- Shift 2

B. Overhanging 12-mer
- Overhang

C. Overlapping 12-mers
- Intersection

D. Generating elements
- Support

E. Filtering elements
- Check for support

Query

Shift 0

Shift 1 [10-mer]

Shift 2 [11-mer]

Overlap
Problem: The spanning set method can only be used to detect a limited number of mismatches.
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 $N \leq \left\lfloor \frac{(L + 2)}{12} \right\rfloor$. 

For reads of length 100 (Illumina), we can allow a maximum of 6 mismatches.

For reads of length 400 (454), 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...
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 $L = 10 (\text{mod} 12)$, 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), we can allow a maximum of 6 mismatches.

For reads of length 400 (454), 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...
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...
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 $L = 10(mod12)$, so $N \leq \lceil(L + 2)/12\rceil$.

So we need to satisfy $N > (K + 2)$, and as such we can only apply the spanning set method when $K \leq \lceil(L + 2)/12\rceil - 2$ 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

Introduction
  Motivation
  GSNAP Features
  Examples of Complex Variant Detection

The Algorithm
  Summary
  Preprocessing
  Method 1: Spanning Set Generation and Filtering
  Method 2: Complete Set Generation and Filtering
  Verification of Candidate Regions
  Detecting Insertions and Deletions
  Detecting Splice Junctions

Results
  Simulated Reads
  Transcriptional Reads
  Limitations

Conclusions
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 \( \geq 14 \) 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

A Single mismatch

Two close mismatches

Two distant mismatches

\[
\left\lfloor \frac{\Delta p + 6}{12} \right\rfloor
\]

where \( \Delta p \) is the distance between start locations of consecutive supporting 12mers.
Complete set generation and filtering

Lower bound on mismatches: \[\lfloor (\Delta p + 6)/12 \rfloor\]
where \(\Delta p\) is the distance between start locations of consecutive supporting 12mers.
Complete set example

B

0..50

[Diagram showing a set with elements marked by X]
Complete set generation and filtering

\[ \frac{8 + 6}{12} = \frac{18 + 6}{12} = \frac{13 + 6}{12} = \]
1 mismatch
2 mismatches
1 mismatch
Outline

Introduction
  Motivation
  GSNAP Features
  Examples of Complex Variant Detection

The Algorithm
  Summary
  Preprocessing
  Method 1: Spanning Set Generation and Filtering
  Method 2: Complete Set Generation and Filtering

Verification of Candidate Regions
  Detecting Insertions and Deletions
  Detecting Splice Junctions

Results
  Simulated Reads
  Transcriptional Reads
  Limitations

Conclusions
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"

A Hash table indexing of a reference sequence

B Hash table indexing of a reference space

C Compressed genomes
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

Introduction
   Motivation
   GSNAP Features
   Examples of Complex Variant Detection

The Algorithm
   Summary
   Preprocessing
   Method 1: Spanning Set Generation and Filtering
   Method 2: Complete Set Generation and Filtering
   Verification of Candidate Regions
   Detecting Insertions and Deletions
   Detecting Splice Junctions

Results
   Simulated Reads
   Transcriptional Reads
   Limitations

Conclusions
Detecting Insertions and Deletions

Complete set generation

Position lists

...
Detecting Insertions and Deletions

Complete set generation

Position lists

Generation of candidates

Merge

Candidate regions in ascending order
Detecting Insertions and Deletions

A Complete set generation

Position lists

... ...

Complete set generation

B Detection of middle indels and local splicing

Candidate 1

Candidate 2

Range of crossovers with 1 mismatch

A Generation of candidates

Candidate regions in ascending order

Possible deletion or local splice pair

Possible insertion
Detecting Insertions and Deletions

A. Generation of candidates

B. Complete set generation

C. Detection of end indels

- Mismatches found using bitwise operations

Position lists:

Candidate regions in ascending order

End region:

Insertion

Deletion

Query

Genome
Outline

Introduction
  Motivation
  GSNAP Features
  Examples of Complex Variant Detection

The Algorithm
  Summary
  Preprocessing
  Method 1: Spanning Set Generation and Filtering
  Method 2: Complete Set Generation and Filtering
  Verification of Candidate Regions
  Detecting Insertions and Deletions
  Detecting Splice Junctions

Results
  Simulated Reads
  Transcriptional Reads
  Limitations

Conclusions
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

A Complete set generation

Position lists

... Complete set generation

B Detection of middle indels and local splicing

Candidate 1

Candidate 2

Range of crossovers with 1 mismatch

A Generation of candidates

Candidate regions in ascending order

Possible deletion or local splice pair

Possible insertion
Detecting Splice Junctions

- 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

A. Generation of candidates

Candidate regions in ascending order

Complete set generation

Position lists

...
Detecting Splice Junctions

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

Introduction
  Motivation
  GSNAP Features
  Examples of Complex Variant Detection

The Algorithm
  Summary
  Preprocessing
  Method 1: Spanning Set Generation and Filtering
  Method 2: Complete Set Generation and Filtering
  Verification of Candidate Regions
  Detecting Insertions and Deletions
  Detecting Splice Junctions

Results
  Simulated Reads
  Transcriptional Reads
  Limitations

Conclusions
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

Variant: Exact

Mapping Software

GSNAP  BWA  Bowtie  SOAP2  SOAP  MAQ

Runtime (s)

0  500  1000  1500  2000  2500  3000
Simulated Reads: 36nt

Runtime (s)

Variant: 1 mm

Mapping Software

GSNAP  BWA  Bowtie  SOAP2  SOAP  MAQ

FU Berlin, GSNAP, Omics WS 2011
Simulated Reads: 36nt

Variant: 2 mm

Mapping Software

GSNAP
BWA
Bowtie
SOAP2
SOAP
MAQ
Simulated Reads: 36nt

Variant: 3 mm

Mapping Software

GSNAP
BWA
Bowtie
SOAP2
SOAP
MAQ

Runtime (s)
0 500 1000 1500 2000 2500 3000
Simulated Reads: 36nt

Variant: Ins (1...3nt)

<table>
<thead>
<tr>
<th>Mapping Software</th>
<th>Runtime (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSNAP</td>
<td>500</td>
</tr>
<tr>
<td>BWA</td>
<td>1000</td>
</tr>
<tr>
<td>Bowtie</td>
<td>1500</td>
</tr>
<tr>
<td>SOAP2</td>
<td>2000</td>
</tr>
<tr>
<td>SOAP</td>
<td>2500</td>
</tr>
<tr>
<td>MAQ</td>
<td>3000</td>
</tr>
</tbody>
</table>
Simulated Reads: 36nt

Variant: Del (1...3nt)

Mapping Software

Runtime (s)

GSNAP

BWA

Bowtie

SOAP2

SOAP

MAQ
Simulated Reads: 36nt

Variant: Ins (4...9nt)

Runtime (s)

GSNAP  BWA  Bowtie  SOAP2  SOAP  MAQ

Mapping Software
Simulated Reads: 36nt

- Simulated reads of 36nt
- Simulation of insertions (1...3nt) and deletions (1...3nt, 4...9nt, 4...30nt)
- Runtime in seconds (0, 500, 1000, 1500, 2000, 2500, 3000)

Comparison of different alignment tools:
- GSNAP
- BWA
- Bowtie
- SOAP2
- SOAP
- MAQ

 FU Berlin, GSNAP, Omics WS 2011
Simulated Reads: 70nt

![Graph showing runtime for different tools and errors](image)

- **GSNAP**
- **BWA**
- **Bowtie**
- **SOAP2**
- **SOAP**
- **MAQ**
Simulated Reads: 100nt

Runtime (s)

Exact
1 mm
2 mm
3 mm
4 mm
5 mm
Ins (1...3nt)
Del (1...3nt)
Ins (4...9nt)
Del (4...30nt)

GSNAP
BWA
Bowtie
SOAP2
SOAP
MAQ
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
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
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
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

Introduction
  Motivation
  GSNAP Features
  Examples of Complex Variant Detection

The Algorithm
  Summary
  Preprocessing
  Method 1: Spanning Set Generation and Filtering
  Method 2: Complete Set Generation and Filtering
  Verification of Candidate Regions
  Detecting Insertions and Deletions
  Detecting Splice Junctions

Results
  Simulated Reads
  Transcriptional Reads

Limitations

Conclusions
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

Introduction
  Motivation
  GSNAP Features
  Examples of Complex Variant Detection

The Algorithm
  Summary
  Preprocessing
  Method 1: Spanning Set Generation and Filtering
  Method 2: Complete Set Generation and Filtering
  Verification of Candidate Regions
  Detecting Insertions and Deletions
  Detecting Splice Junctions

Results
  Simulated Reads
  Transcriptional Reads

Limitations

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

**Knuth D.E.**

**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
Aligning Paired-End Reads

- Optional
Bisulfite-converted DNA

- Optional