Hybrid correction of long reads with a variable-order de Bruijn graph

Pierre Morisse, Thierry Lecroq and Arnaud Lefebvre
pierre.morisse2@univ-rouen.fr

Laboratoire d'Informatique, de Traitement de l'Information et des Systèmes

DSB 2018 - Helsinki

May 16, 2018

Plan

Next Generation Sequencing

- In 2005, Next Generation Sequencing (NGS) technologies started to develop
- Production of millions of short sequences (100-300 bases), called reads, useful to resolve various problems
- These reads contain sequencing errors ($\sim 1\%$)
- $\Rightarrow$ Efficient algorithms are required to process these reads
- NGS data analysis became an important research field

Next Generation Sequencing

- In 2005, Next Generation Sequencing (NGS) technologies started to develop
- Production of millions of short sequences (100-300 bases), called reads, useful to resolve various problems
- These reads contain sequencing errors ($\sim 1\%$)
- $\Rightarrow$ Efficient algorithms are required to process these reads
- NGS data analysis became an important research field

Next Generation Sequencing

- In 2005, Next Generation Sequencing (NGS) technologies started to develop
- Production of millions of short sequences (100-300 bases), called reads, useful to resolve various problems
- These reads contain sequencing errors ($\sim$ 1%)
- $\Rightarrow$ Efficient algorithms are required to process these reads
- NGS data analysis became an important research field

Next Generation Sequencing

- In 2005, Next Generation Sequencing (NGS) technologies started to develop
- Production of millions of short sequences (100-300 bases), called reads, useful to resolve various problems
- These reads contain sequencing errors ($\sim$ 1%)
- $\Rightarrow$ Efficient algorithms are required to process these reads
- NGS data analysis became an important research field

Next Generation Sequencing

- In 2005, Next Generation Sequencing (NGS) technologies started to develop
- Production of millions of short sequences (100-300 bases), called reads, useful to resolve various problems
- These reads contain sequencing errors ($\sim$ 1%)
- $\Rightarrow$ Efficient algorithms are required to process these reads
- NGS data analysis became an important research field

Third Generation Sequencing

- More recently, Third Generation Sequencing technologies started to develop
- Two main technologies: Pacific Biosciences and Oxford Nanopore
- Allow the sequencing of longer reads (several thousand of bases)
- Very useful to resolve assembly problems for large and complex genomes
- Much higher error rate, around 15% for Pacific Biosciences and up to 30% for Oxford Nanopore

Third Generation Sequencing

- More recently, Third Generation Sequencing technologies started to develop
- Two main technologies: Pacific Biosciences and Oxford Nanopore
- Allow the sequencing of longer reads (several thousand of bases)
- Very useful to resolve assembly problems for large and complex genomes
- Much higher error rate, around 15% for Pacific Biosciences and up to 30% for Oxford Nanopore

Third Generation Sequencing

- More recently, Third Generation Sequencing technologies started to develop
- Two main technologies: Pacific Biosciences and Oxford Nanopore
- Allow the sequencing of longer reads (several thousand of bases)
- Very useful to resolve assembly problems for large and complex genomes
- Much higher error rate, around 15% for Pacific Biosciences and up to 30% for Oxford Nanopore

Third Generation Sequencing

- More recently, Third Generation Sequencing technologies started to develop
- Two main technologies: Pacific Biosciences and Oxford Nanopore
- Allow the sequencing of longer reads (several thousand of bases)
- Very useful to resolve assembly problems for large and complex genomes
- Much higher error rate, around 15% for Pacific Biosciences and up to 30% for Oxford Nanopore

Third Generation Sequencing

- More recently, Third Generation Sequencing technologies started to develop
- Two main technologies: Pacific Biosciences and Oxford Nanopore
- Allow the sequencing of longer reads (several thousand of bases)
- Very useful to resolve assembly problems for large and complex genomes
- Much higher error rate, around 15% for Pacific Biosciences and up to 30% for Oxford Nanopore

Problem

- Due to their high error rate, error correction of long reads is mandatory
- Various methods already exist for the correction of short reads, but are not applicable to long reads
- Forces the development of new error correction methods
- Two main categories: self-correction and hybrid correction

Problem

- Due to their high error rate, error correction of long reads is mandatory
- Various methods already exist for the correction of short reads, but are not applicable to long reads
- Forces the development of new error correction methods
- Two main categories: self-correction and hybrid correction

Problem

- Due to their high error rate, error correction of long reads is mandatory
- Various methods already exist for the correction of short reads, but are not applicable to long reads
- Forces the development of new error correction methods
- Two main categories: self-correction and hybrid correction

Problem

- Due to their high error rate, error correction of long reads is mandatory
- Various methods already exist for the correction of short reads, but are not applicable to long reads
- Forces the development of new error correction methods
- Two main categories: self-correction and hybrid correction

Solution

We developed a new hybrid method, HG-CoLoR, that:

- Combines different approaches from the state-of-the-art:
  - Short reads alignment
  - Use of a short reads de Bruijn graph
- Follows a seed-and-extend approach:
  1. Align the short reads to the long reads $\Rightarrow$ find seeds
  2. Link the seeds together using a variable-order de Bruijn graph

Solution

We developed a new hybrid method, HG-CoLoR, that:

- Combines different approaches from the state-of-the-art:
  - Short reads alignment
  - Use of a short reads de Bruijn graph
- Follows a seed-and-extend approach:
  1. Align the short reads to the long reads $\Rightarrow$ find seeds
  2. Link the seeds together using a variable-order de Bruijn graph

Solution

We developed a new hybrid method, HG-CoLoR, that:

- Combines different approaches from the state-of-the-art:
  - Short reads alignment
  - Use of a short reads de Bruijn graph
- Follows a seed-and-extend approach:
  1. Align the short reads to the long reads $\Rightarrow$ find seeds
  2. Link the seeds together using a variable-order de Bruijn graph

Solution

We developed a new hybrid method, HG-CoLoR, that:

- Combines different approaches from the state-of-the-art:
  - Short reads alignment
  - Use of a short reads de Bruijn graph
- Follows a seed-and-extend approach:
  1. Align the short reads to the long reads $\Rightarrow$ find seeds
  2. Link the seeds together using a variable-order de Bruijn graph

Solution

We developed a new hybrid method, HG-CoLoR, that:

- Combines different approaches from the state-of-the-art:
  - Short reads alignment
  - Use of a short reads de Bruijn graph
- Follows a seed-and-extend approach:
  1. Align the short reads to the long reads $\Rightarrow$ find seeds
  2. Link the seeds together using a variable-order de Bruijn graph

de Bruijn graph

Problem

- de Bruijn graphs are widely used for assembly and correction...
- ...But face difficulties:
  - Large $k \Rightarrow$ Missing edges in insufficiently covered regions
  - Small $k \Rightarrow$ Too many branches

Solutions

- Build multiple de Bruijn graphs of different orders
- Requires a different graph for each order
- Consumes large amounts of time and memory

de Bruijn graph

Problem

- de Bruijn graphs are widely used for assembly and correction...
- ...But face difficulties:
  - Large $k \Rightarrow$ Missing edges in insufficiently covered regions
  - Small $k \Rightarrow$ Too many branches

Solutions

- Build multiple de Bruijn graphs of different orders
- Requires a different graph for each order
- Consumes large amounts of time and memory

de Bruijn graph

Problem

- de Bruijn graphs are widely used for assembly and correction...
- ...But face difficulties:
  - Large $k \Rightarrow$ Missing edges in insufficiently covered regions
  - Small $k \Rightarrow$ Too many branches

Solutions

- Build multiple de Bruijn graphs of different orders
- Requires a different graph for each order
- Consumes large amounts of time and memory

de Bruijn graph

Problem

- de Bruijn graphs are widely used for assembly and correction...
- ...But face difficulties:
  - Large $k \Rightarrow$ Missing edges in insufficiently covered regions
  - Small $k \Rightarrow$ Too many branches

Solutions

- Build multiple de Bruijn graphs of different orders
- Requires a different graph for each order
- Consumes large amounts of time and memory

de Bruijn graph

Problem

- de Bruijn graphs are widely used for assembly and correction...
- ...But face difficulties:
  - Large $k \Rightarrow$ Missing edges in insufficiently covered regions
  - Small $k \Rightarrow$ Too many branches

Solutions

- Build multiple de Bruijn graphs of different orders
- Requires a different graph for each order
- Consumes large amounts of time and memory

de Bruijn graph

Problem

- de Bruijn graphs are widely used for assembly and correction...
- ...But face difficulties:
  - Large $k \Rightarrow$ Missing edges in insufficiently covered regions
  - Small $k \Rightarrow$ Too many branches

Solutions

- Build multiple de Bruijn graphs of different orders
- Requires a different graph for each order
- Consumes large amounts of time and memory

de Bruijn graph

Problem

- de Bruijn graphs are widely used for assembly and correction...
- ...But face difficulties:
  - Large $k \Rightarrow$ Missing edges in insufficiently covered regions
  - Small $k \Rightarrow$ Too many branches

Solutions

- Build multiple de Bruijn graphs of different orders
- Requires a different graph for each order
- Consumes large amounts of time and memory

Variable-order de Bruijn graph

Idea

Represent all the de Bruijn graphs, up to a maximum order K, in a single data structure

Problem

Theoretical methods, or unsatisfying implementations

Variable-order de Bruijn graph

Idea

Represent all the de Bruijn graphs, up to a maximum order K, in a single data structure

Problem

Theoretical methods, or unsatisfying implementations

Variable-order de Bruijn graph

Our representation

- Relies on PgSA [Kowalski et al., 2015]
- The K-mers from the reads are stored in the index
- The index is queried in order to retrieve the edges

Variable-order de Bruijn graph

Our representation

- Relies on PgSA [Kowalski et al., 2015]
- The K-mers from the reads are stored in the index
- The index is queried in order to retrieve the edges

Variable-order de Bruijn graph

Our representation

- Relies on PgSA [Kowalski et al., 2015]
- The K-mers from the reads are stored in the index
- The index is queried in order to retrieve the edges

PgSA

PgSA [Kowalski et al., 2015] is a data structure that allows the indexing of a set of reads, in order to answer the following queries on the reads, for a given string f:

1. In which reads does f occur?
2. In how many reads does f occur?
3. What are the occurrences positions of f?
4. What is the number of occurrences of f?
5. In which reads does f occur only once?
6. In how many reads does f occur only once?
7. What are the occurrences positions of f in the reads where it occurs only once?

PgSA

PgSA [Kowalski et al., 2015] is a data structure that allows the indexing of a set of reads, in order to answer the following queries on the reads, for a given string f:

1. In which reads does f occur?
2. In how many reads does f occur?
3. What are the occurrences positions of f?
4. What is the number of occurrences of f?
5. In which reads does f occur only once?
6. In how many reads does f occur only once?
7. What are the occurrences positions of f in the reads where it occurs only once?

PgSA

PgSA [Kowalski et al., 2015] is a data structure that allows the indexing of a set of reads, in order to answer the following queries on the reads, for a given string f:

1. In which reads does f occur?
2. In how many reads does f occur?
3. What are the occurrences positions of f?
4. What is the number of occurrences of f?
5. In which reads does f occur only once?
6. In how many reads does f occur only once?
7. What are the occurrences positions of f in the reads where it occurs only once?

PgSA

PgSA [Kowalski et al., 2015] is a data structure that allows the indexing of a set of reads, in order to answer the following queries on the reads, for a given string f:

1. In which reads does f occur?
2. In how many reads does f occur?
3. What are the occurrences positions of f?
4. What is the number of occurrences of f?
5. In which reads does f occur only once?
6. In how many reads does f occur only once?
7. What are the occurrences positions of f in the reads where it occurs only once?

PgSA

PgSA [Kowalski et al., 2015] is a data structure that allows the indexing of a set of reads, in order to answer the following queries on the reads, for a given string f:

1. In which reads does f occur?
2. In how many reads does f occur?
3. What are the occurrences positions of f?
4. What is the number of occurrences of f?
5. In which reads does f occur only once?
6. In how many reads does f occur only once?
7. What are the occurrences positions of f in the reads where it occurs only once?

PgSA

PgSA [Kowalski et al., 2015] is a data structure that allows the indexing of a set of reads, in order to answer the following queries on the reads, for a given string f:

1. In which reads does f occur?
2. In how many reads does f occur?
3. What are the occurrences positions of f?
4. What is the number of occurrences of f?
5. In which reads does f occur only once?
6. In how many reads does f occur only once?
7. What are the occurrences positions of f in the reads where it occurs only once?

PgSA

PgSA [Kowalski et al., 2015] is a data structure that allows the indexing of a set of reads, in order to answer the following queries on the reads, for a given string f:

1. In which reads does f occur?
2. In how many reads does f occur?
3. What are the occurrences positions of f?
4. What is the number of occurrences of f?
5. In which reads does f occur only once?
6. In how many reads does f occur only once?
7. What are the occurrences positions of f in the reads where it occurs only once?

PgSA

PgSA [Kowalski et al., 2015] is a data structure that allows the indexing of a set of reads, in order to answer the following queries on the reads, for a given string f:

1. In which reads does f occur?
2. In how many reads does f occur?
3. What are the occurrences positions of f?
4. What is the number of occurrences of f?
5. In which reads does f occur only once?
6. In how many reads does f occur only once?
7. What are the occurrences positions of f in the reads where it occurs only once?

PgSA

Index construction

- Concatenation of all the reads ⇒ Pseudogenome (Pg)
- Construction of the sparse Suffix Array (SA) of the obtained pseudogenome
- Construction of an auxiliary array

Queries

Queries are handled by a binary search over the suffix array, and with the help of the auxiliary array

PgSA

Index construction

- Concatenation of all the reads ⇒ Pseudogenome (Pg)
- Construction of the sparse Suffix Array (SA) of the obtained pseudogenome
- Construction of an auxiliary array

Queries

Queries are handled by a binary search over the suffix array, and with the help of the auxiliary array

PgSA

Index construction

- Concatenation of all the reads ⇒ Pseudogenome (Pg)
- Construction of the sparse Suffix Array (SA) of the obtained pseudogenome
- Construction of an auxiliary array

Queries

Queries are handled by a binary search over the suffix array, and with the help of the auxiliary array

PgSA

Index construction

- Concatenation of all the reads ⇒ Pseudogenome (Pg)
- Construction of the sparse Suffix Array (SA) of the obtained pseudogenome
- Construction of an auxiliary array

Queries

Queries are handled by a binary search over the suffix array, and with the help of the auxiliary array

Our representation

- Define a maximum order K
- Extract the K-mers of the reads
- Build the PgSA index of the K-mers
- Query the index, looping over the third query (what are the occurrences positions of f?), to retrieve the edges

Our representation

- Define a maximum order K
- Extract the K-mers of the reads
- Build the PgSA index of the K-mers
- Query the index, looping over the third query (what are the occurrences positions of f?), to retrieve the edges

Our representation

- Define a maximum order K
- Extract the K-mers of the reads
- Build the PgSA index of the K-mers
- Query the index, looping over the third query (what are the occurrences positions of f?), to retrieve the edges

Our representation

- Define a maximum order K
- Extract the K-mers of the reads
- Build the PgSA index of the K-mers
- Query the index, looping over the third query (what are the occurrences positions of f?), to retrieve the edges

Our representation

Example

Let $S = \{\text{AGCTTACA}, \text{CTTACGTA}\}$

To build the variable-order de Bruijn graph of maximum order $K = 6$, we index the 6-mers of the reads:

K-mers set
1: AGCTTA
2: CTTACA
3: CTTACG
4: GCTTAC
5: TACGTA
6: TTACGT

Our representation

Example

Let $S = \{\text{AGCTTACA}, \text{CTTACGTA}\}$

To build the variable-order de Bruijn graph of maximum order $K = 6$, we index the 6-mers of the reads:

K-mers set

1: AGCTTA
2: CTTACA
3: CTTACG
4: GCTTAC
5: TACGTA
6: TTACGT

Our representation

Example

Let $S = \{\text{AGCTTACA}, \text{CTTACGTA}\}$

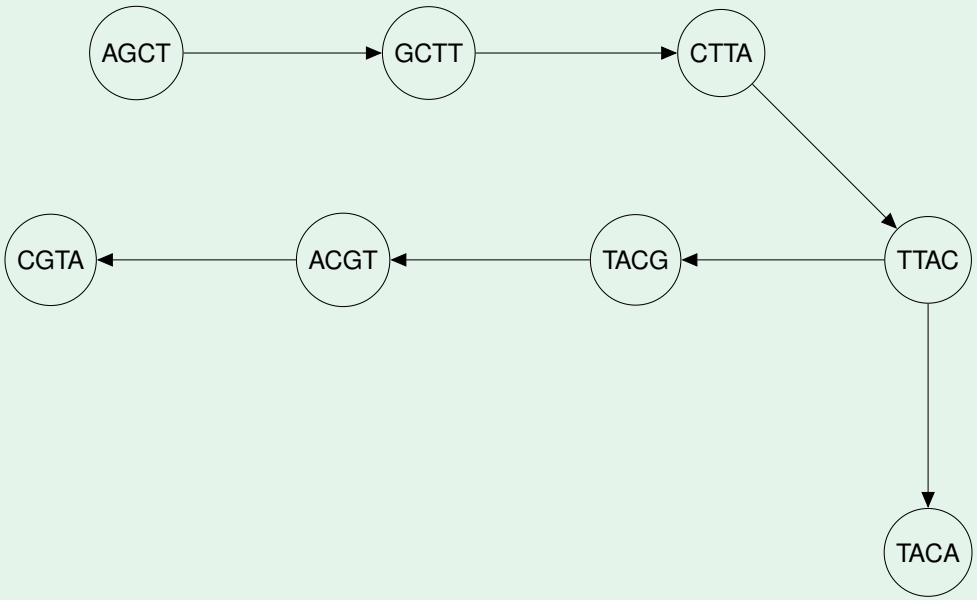
To build the variable-order de Bruijn graph of maximum order $K = 6$, we index the 6-mers of the reads:

K-mers set
1: AGCTTA
2: CTTACA
3: CTTACG
4: GCTTAC
5: TACGTA
6: TTACGT

Our representation

Example

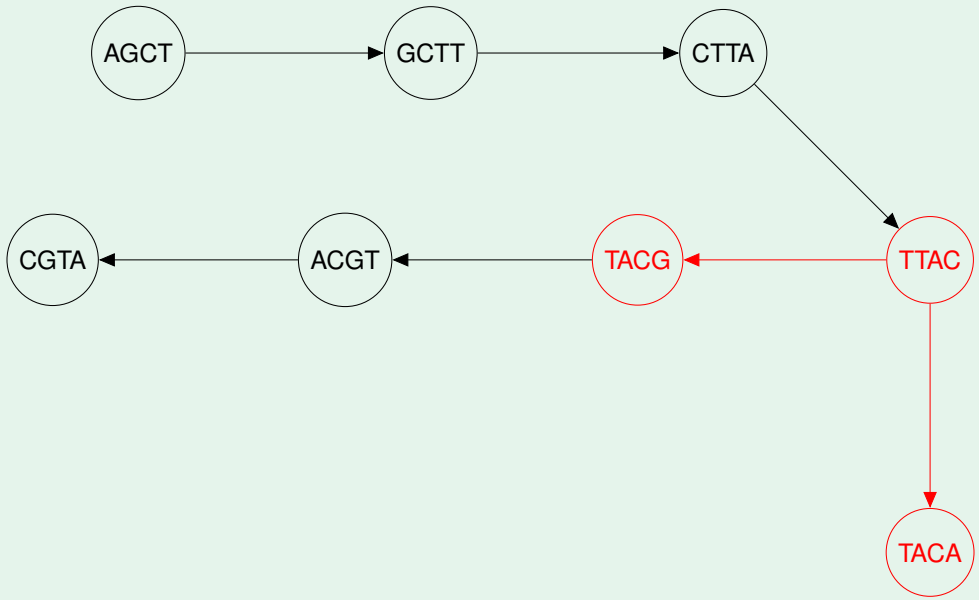
de Bruijn graph of order 4 of the previous set of reads:



Our representation

Example

de Bruijn graph of order 4 of the previous set of reads:



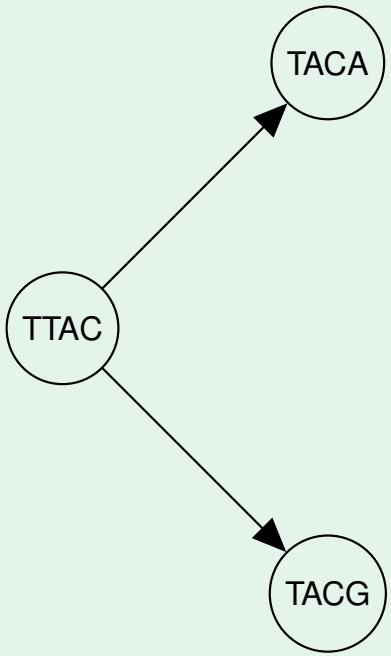
Our representation

Example

k-mers set

1: AGCTTA
2: CTTACA
3: CTTACG
4: GCTTAC
5: TACGTA
6: TTACGT

PgSA Index



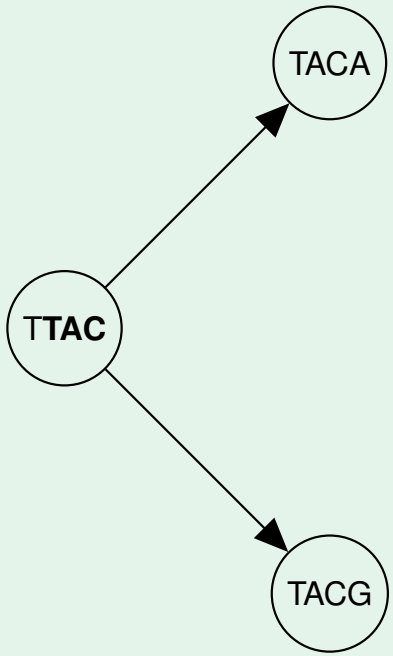
Our representation

Example

k-mers set

1: AGCTTA
2: CTTACA
3: CTTACG
4: GCTTAC
5: TACGTA
6: TTACGT

PgSA Index

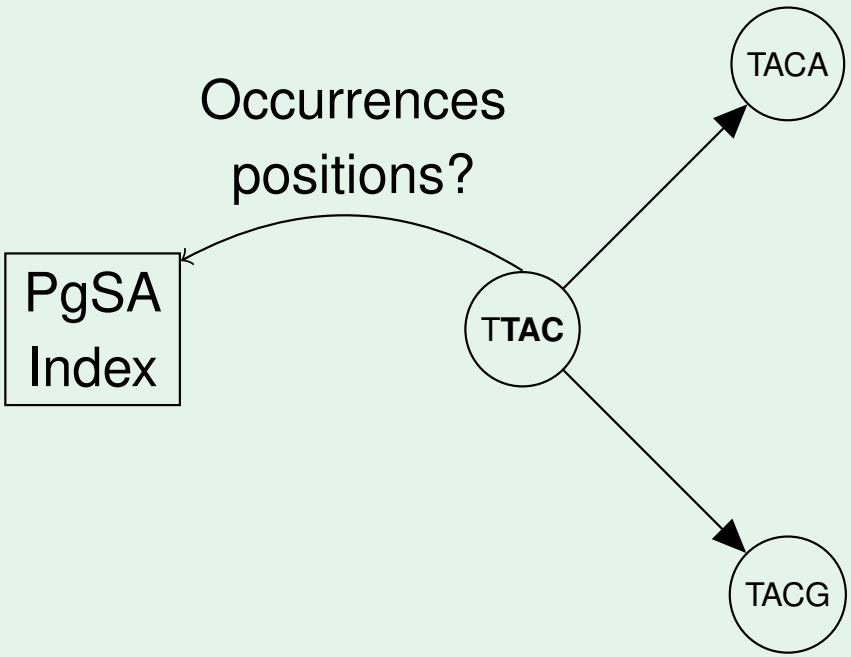


Our representation

Example

k-mers set

1: AGCTTA
2: CTTACA
3: CTTACG
4: GCTTAC
5: TACGTA
6: TTACGT

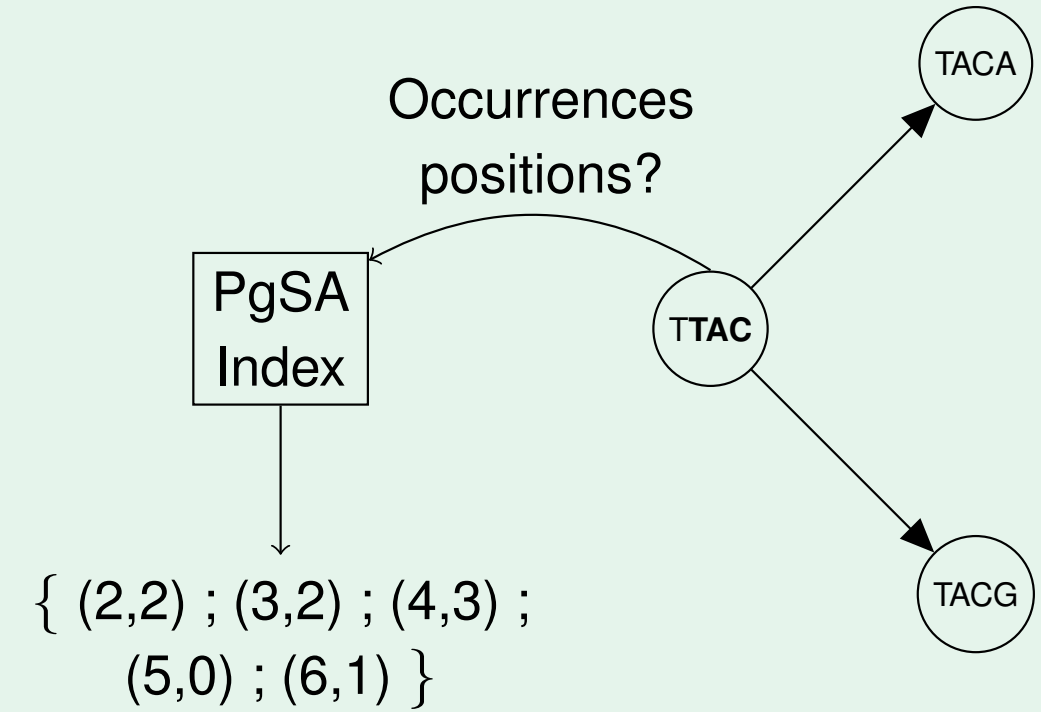


Our representation

Example

k-mers set

1: AGCTTA
2: CTTACA
3: CTTACG
4: GCTTAC
5: TACGTA
6: TTACGT



Our representation

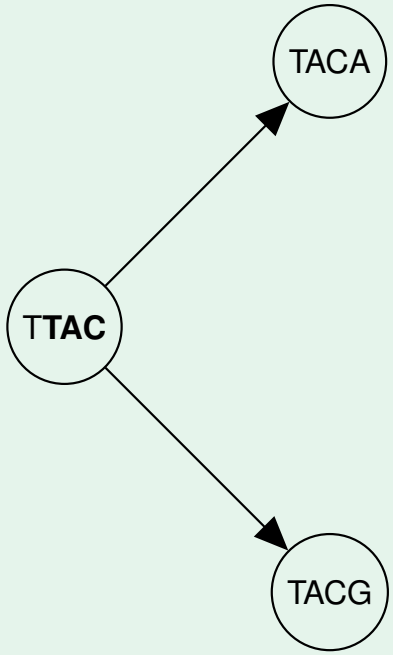
Example

k-mers set

1: AGCTTA
2: CT**TAC**A
3: CTTACG
4: GCTTAC
5: TACGTA
6: TTACGT

PgSA Index

{ **(2,2)** ; (3,2) ; (4,3) ; (5,0) ; (6,1) }



Our representation

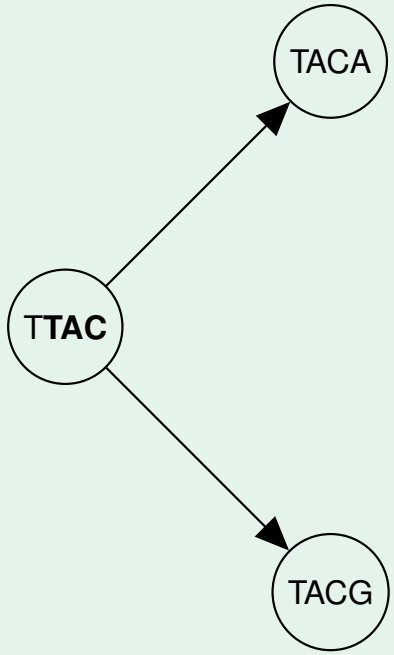
Example

k-mers set

1: AGCTTA
2: CT**TAC**A
3: CTTACG
4: GCTTAC
5: TACGTA
6: TTACGT

PgSA Index

{ **(2,2)** ; (3,2) ; (4,3) ; (5,0) ; (6,1) }



Our representation

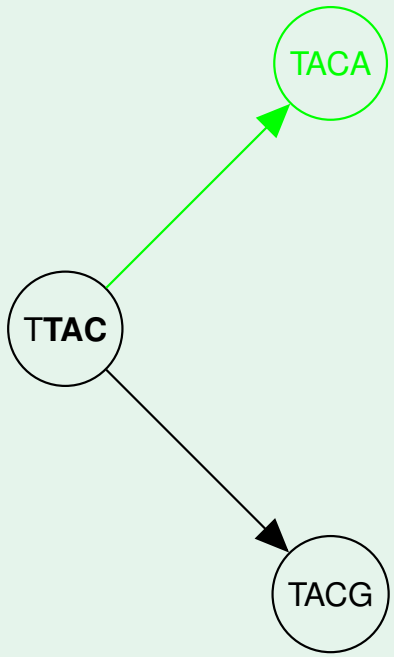
Example

k-mers set

1: AGCTTA
2: CT**TAC**A
3: CTTACG
4: GCTTAC
5: TACGTA
6: TTACGT

PgSA Index

{ (2,2) ; (3,2) ; (4,3) ; (5,0) ; (6,1) }



Our representation

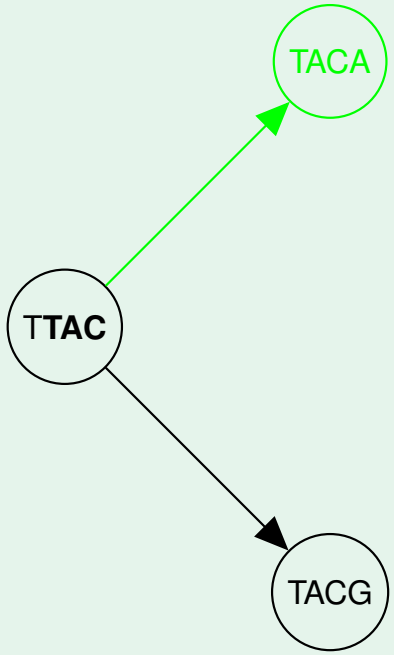
Example

k-mers set

1: AGCTTA
2: CTTACA
3: CT**TAC**G
4: GCTTAC
5: TACGTA
6: TTACGT

PgSA Index

{ (2,2) ; **(3,2)** ; (4,3) ;
(5,0) ; (6,1) }



Our representation

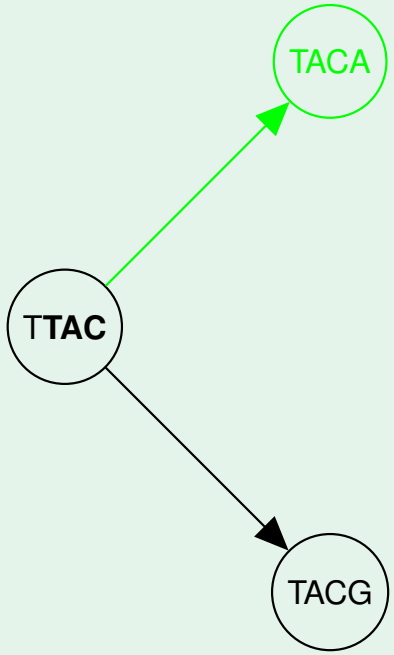
Example

k-mers set

1: AGCTTA
2: CTTACA
3: CT**TAC**G
4: GCTTAC
5: TACGTA
6: TTACGT

PgSA Index

{ (2,2) ; **(3,2)** ; (4,3) ;
(5,0) ; (6,1) }



Our representation

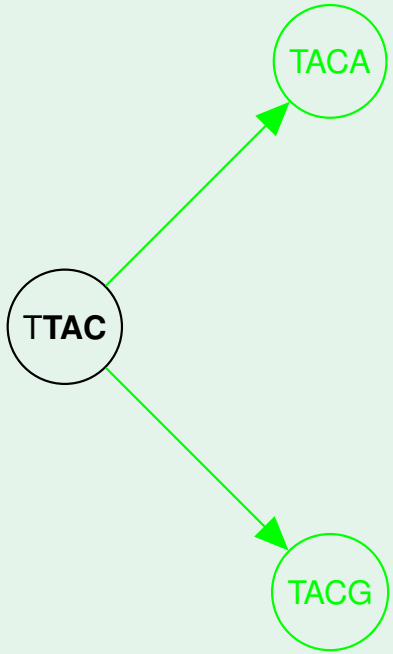
Example

k-mers set

1: AGCTTA
2: CTTACA
3: CT**TAC**G
4: GCTTAC
5: TACGTA
6: TTACGT

PgSA Index

{ (2,2) ; **(3,2)** ; (4,3) ; (5,0) ; (6,1) }



Our representation

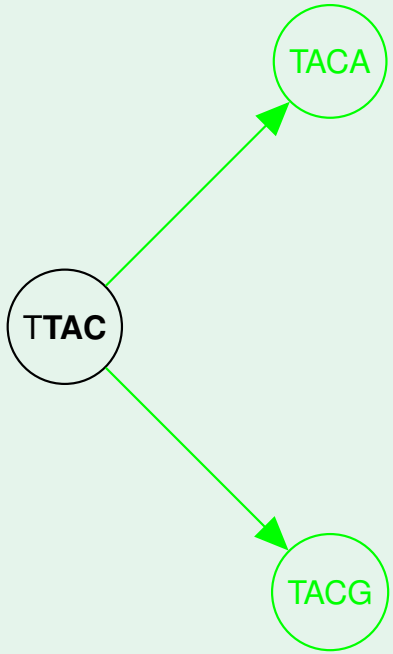
Example

k-mers set

1: AGCTTA
2: CTTACA
3: CTTACG
4: GCT**TAC**
5: TACGTA
6: TTACGT

PgSA Index

{ (2,2) ; (3,2) ; **(4,3)** ; (5,0) ; (6,1) }



Our representation

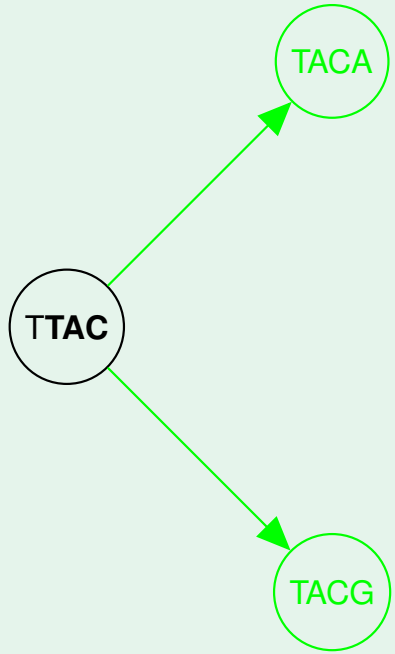
Example

k-mers set

1: AGCTTA
2: CTTACA
3: CTTACG
4: GCT**TAC**
5: TACGTA
6: TTACGT

PgSA Index

{ (2,2) ; (3,2) ; **(4,3)** ; (5,0) ; (6,1) }



Our representation

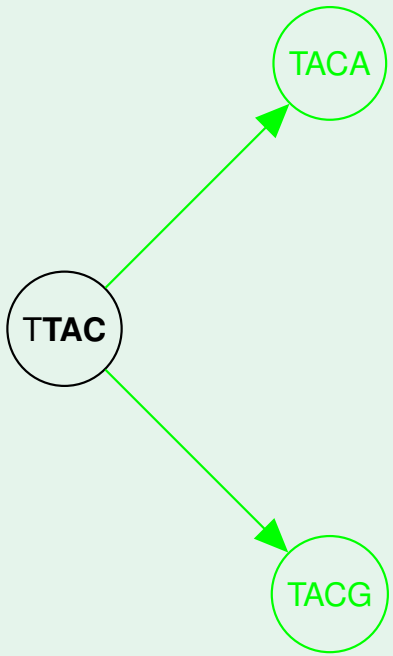
Example

k-mers set

1: AGCTTA
2: CTTACA
3: CTTACG
4: GCTTAC
5: **TAC**GTA
6: TTACGT

PgSA Index

{ (2,2) ; (3,2) ; (4,3) ;
(5,0) ; (6,1) }



Our representation

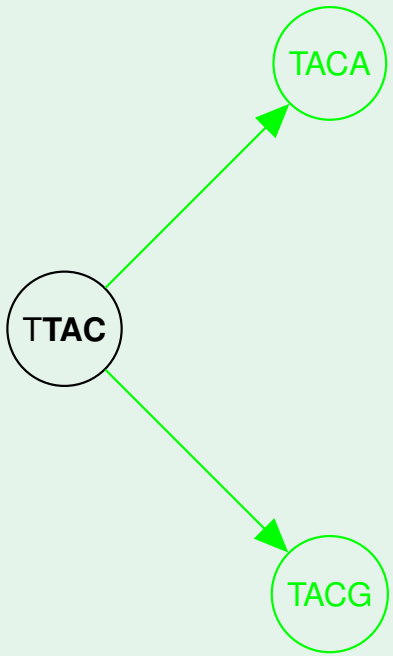
Example

k-mers set

1: AGCTTA
2: CTTACA
3: CTTACG
4: GCTTAC
5: **TAC**G**T**A
6: TTACGT

PgSA Index

{ (2,2) ; (3,2) ; (4,3) ;
(5,0) ; (6,1) }



Our representation

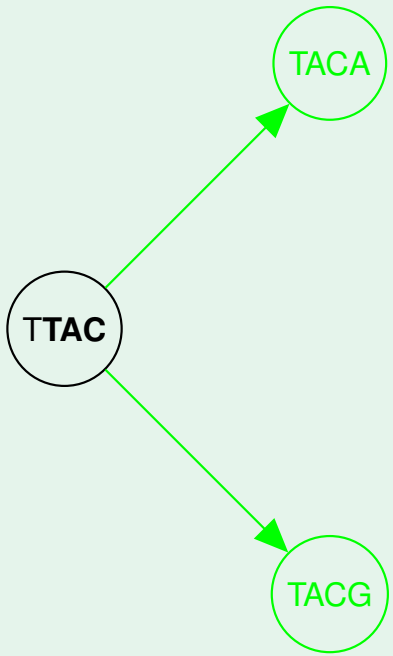
Example

k-mers set

1: AGCTTA
2: CTTACA
3: CTTACG
4: GCTTAC
5: **TAC**G**T**A
6: TTACGT

PgSA Index

{ (2,2) ; (3,2) ; (4,3) ;
(5,0) ; (6,1) }



Our representation

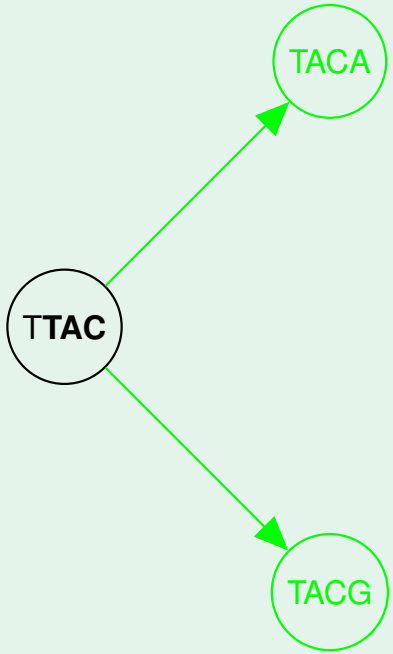
Example

k-mers set

1: AGCTTA
2: CTTACA
3: CTTACG
4: GCTTAC
5: TACGTA
6: T**TAC**GT

PgSA Index

{ (2,2) ; (3,2) ; (4,3) ;
(5,0) ; **(6,1)** }



Our representation

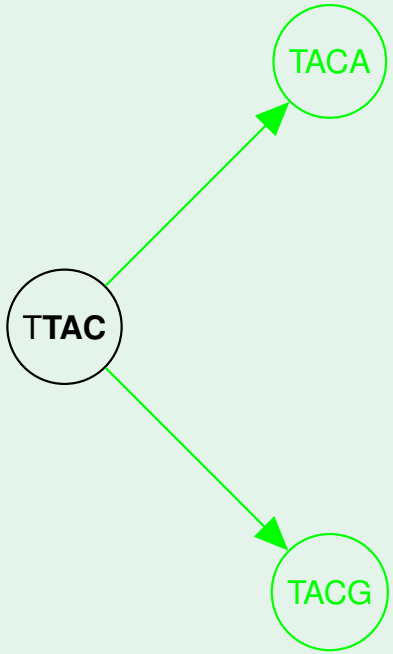
Example

k-mers set

1: AGCTTA
2: CTTACA
3: CTTACG
4: GCTTAC
5: TACGTA
6: T**TACG**T

PgSA Index

{ (2,2) ; (3,2) ; (4,3) ;
(5,0) ; **(6,1)** }



Our representation

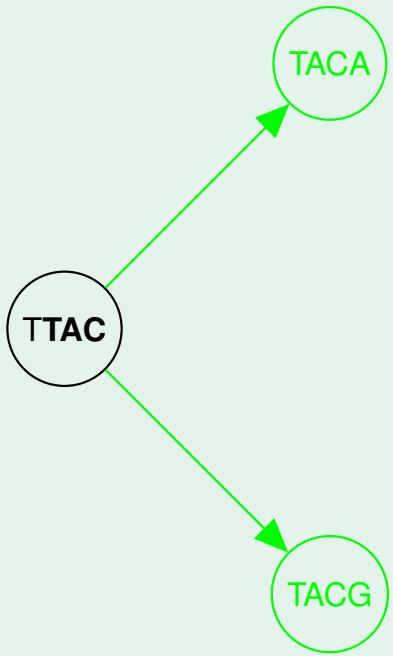
Example

k-mers set

1: AGCTTA
2: CTTACA
3: CTTACG
4: GCTTAC
5: TACGTA
6: T**TACG**T

PgSA Index

{ (2,2) ; (3,2) ; (4,3) ;
(5,0) ; **(6,1)** }



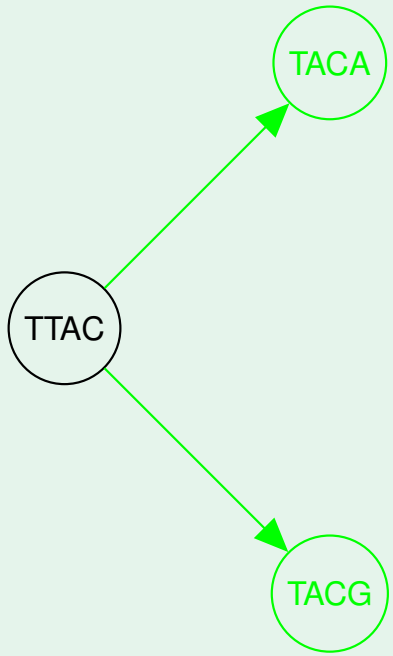
Our representation

Example

k-mers set

1: AGCTTA
2: CTTACA
3: CTTACG
4: GCTTAC
5: TACGTA
6: TTACGT

PgSA Index



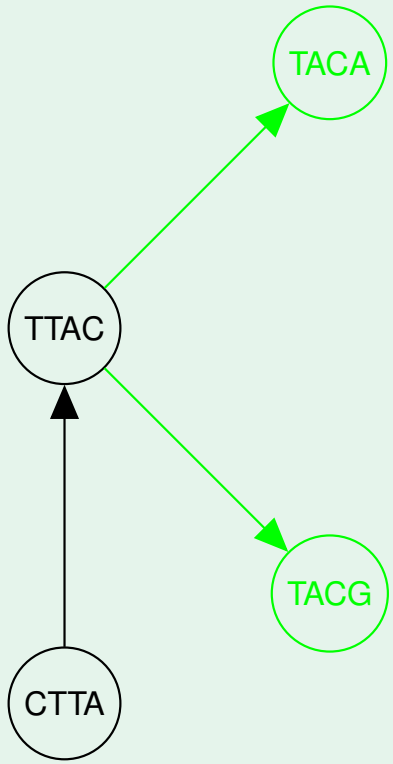
Our representation

Example

k-mers set

1: AGCTTA
2: CTTACA
3: CTTACG
4: GCTTAC
5: TACGTA
6: TTACGT

PgSA
Index



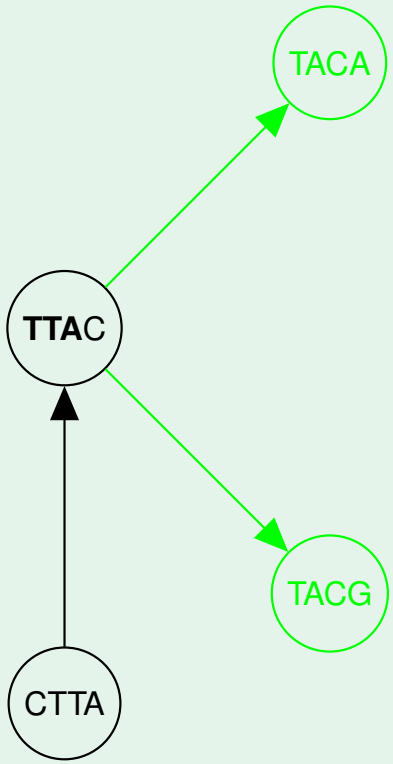
Our representation

Example

k-mers set

1: AGCTTA
2: CTTACA
3: CTTACG
4: GCTTAC
5: TACGTA
6: TTACGT

PgSA
Index

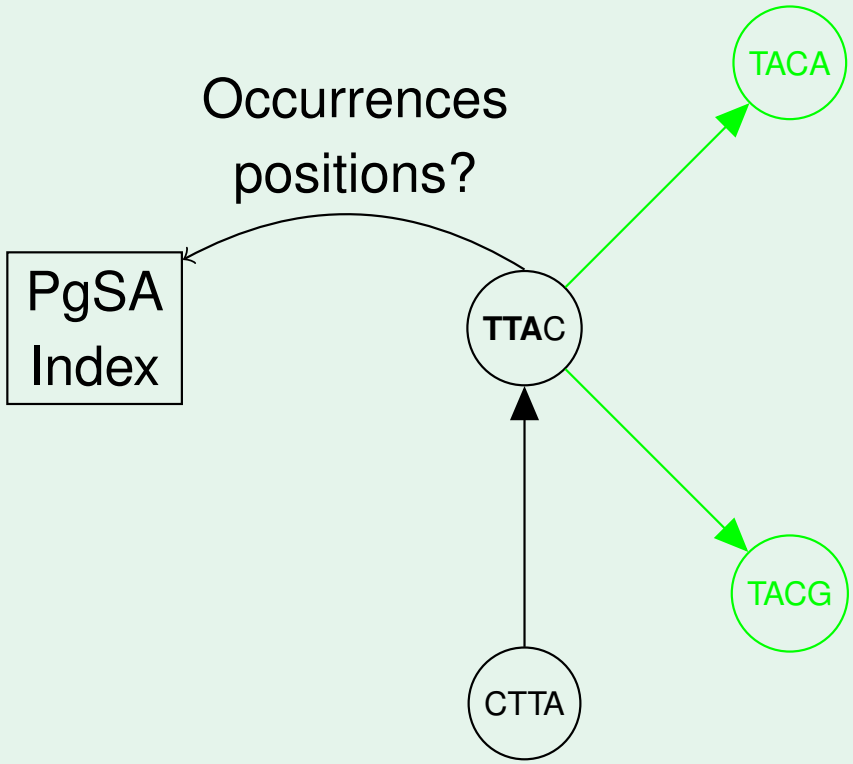


Our representation

Example

k-mers set

1: AGCTTA
2: CTTACA
3: CTTACG
4: GCTTAC
5: TACGTA
6: TTACGT



Our representation

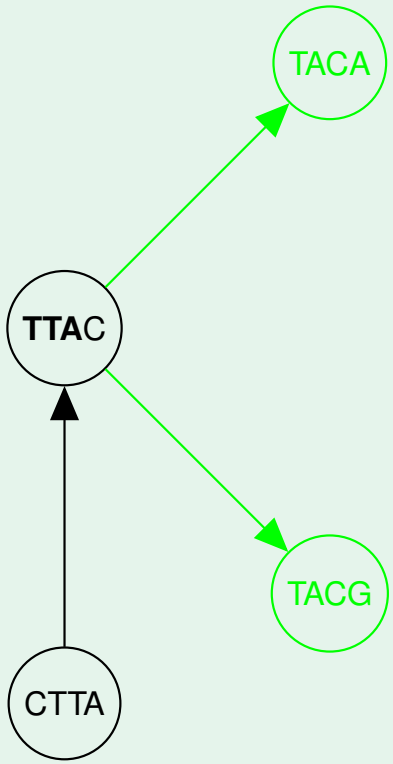
Example

k-mers set

1: AGCTTA
2: CTTACA
3: CTTACG
4: GCTTAC
5: TACGTA
6: TTACGT

PgSA Index

{ (1,3) ; (2,1) ; (3,1) ; (4,2) ; (6,0) }



Our representation

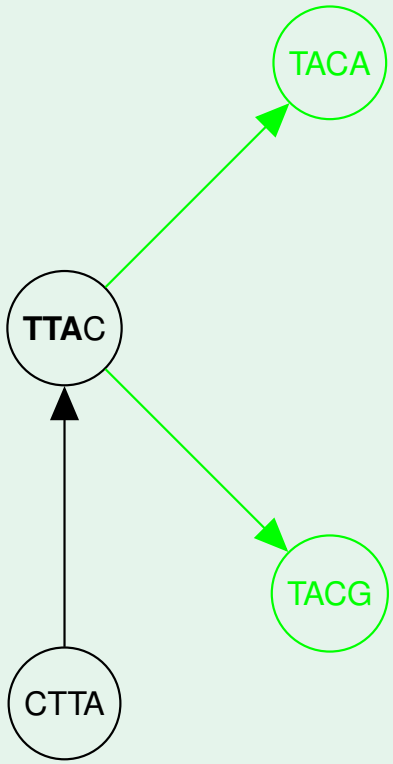
Example

k-mers set

1: AGC**TTA**
2: CTTACA
3: CTTACG
4: GCTTAC
5: TACGTA
6: TTACGT

PgSA Index

{ **(1,3)** ; (2,1) ; (3,1) ; (4,2) ; (6,0) }



Our representation

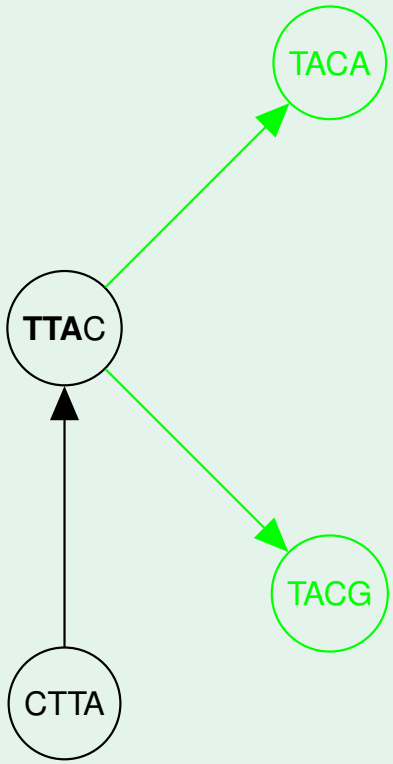
Example

k-mers set

1: AG**CTTA**
2: CTTACA
3: CTTACG
4: GCTTAC
5: TACGTA
6: TTACGT

PgSA Index

{ **(1,3)** ; (2,1) ; (3,1) ; (4,2) ; (6,0) }



Our representation

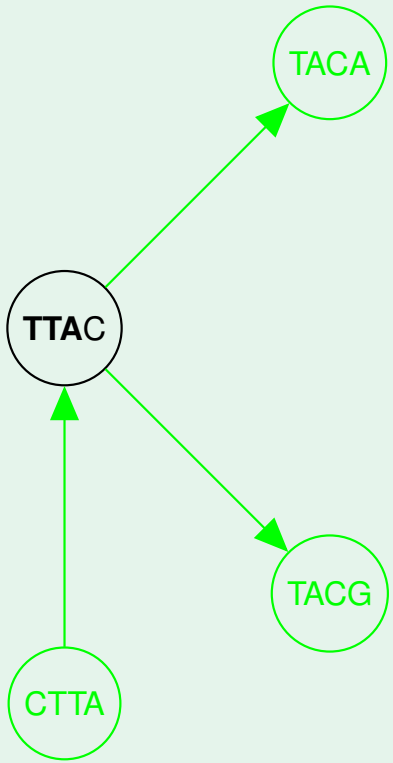
Example

k-mers set

1: AG**CTTA**
2: CTTACA
3: CTTACG
4: GCTTAC
5: TACGTA
6: TTACGT

PgSA
Index

{ **(1,3)** ; (2,1) ; (3,1) ;
(4,2) ; (6,0) }



Our representation

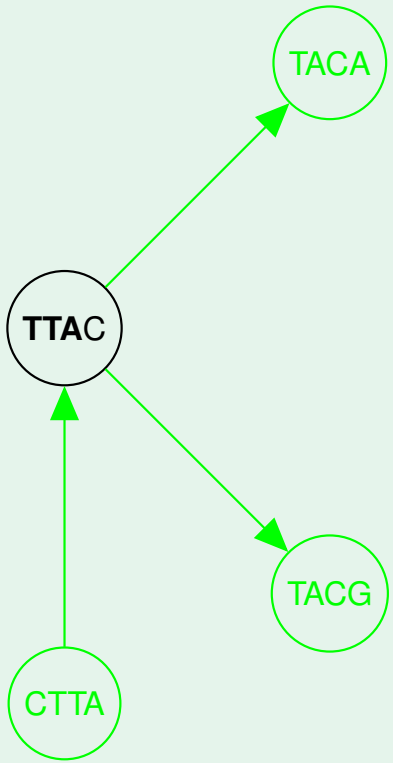
Example

k-mers set

1: AGCTTA
2: C**TTA**CA
3: CTTACG
4: GCTTAC
5: TACGTA
6: TTACGT

PgSA
Index

{ (1,3) ; **(2,1)** ; (3,1) ;
(4,2) ; (6,0) }



Our representation

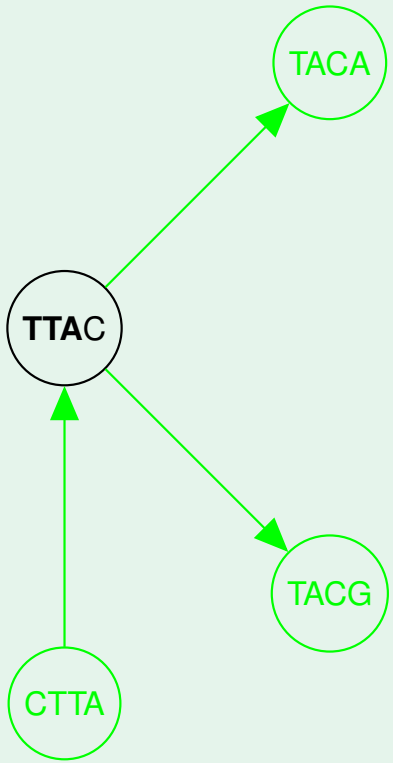
Example

k-mers set

1: AGCTTA
2: **CTTA**CA
3: CTTACG
4: GCTTAC
5: TACGTA
6: TTACGT

PgSA Index

{ (1,3) ; **(2,1)** ; (3,1) ; (4,2) ; (6,0) }



Our representation

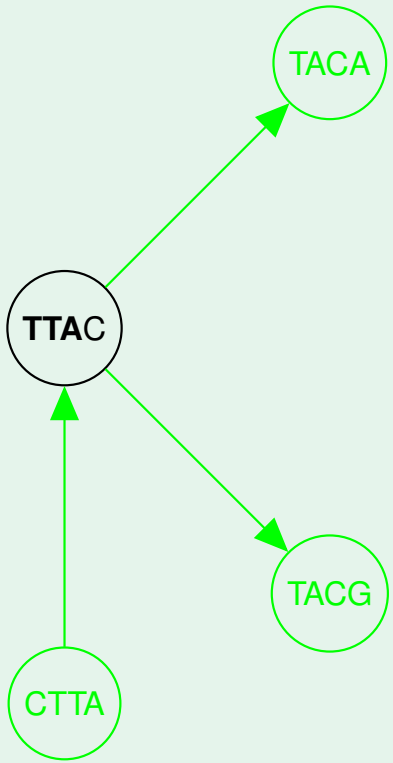
Example

k-mers set

1: AGCTTA
2: **CTTA**CA
3: CTTACG
4: GCTTAC
5: TACGTA
6: TTACGT

PgSA Index

{ (1,3) ; **(2,1)** ; (3,1) ; (4,2) ; (6,0) }



Our representation

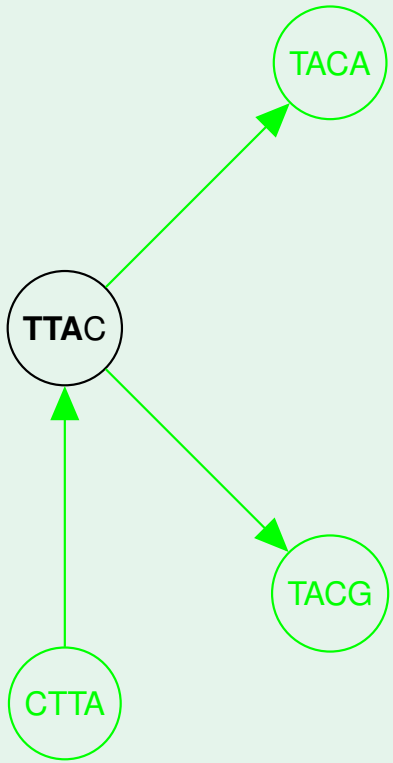
Example

k-mers set

1: AGCTTA
2: CTTACA
3: C**TTA**CG
4: GCTTAC
5: TACGTA
6: TTACGT

PgSA Index

{ (1,3) ; (2,1) ; **(3,1)** ; (4,2) ; (6,0) }



Our representation

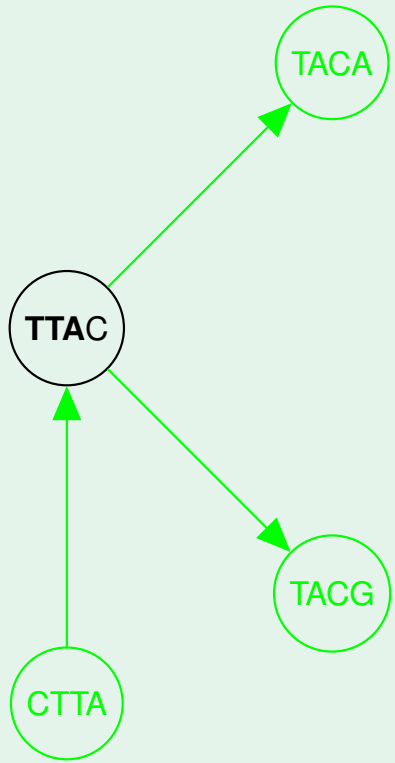
Example

k-mers set

1: AGCTTA
2: CTTACA
3: **CTTA**CG
4: GCTTAC
5: TACGTA
6: TTACGT

PgSA Index

{ (1,3) ; (2,1) ; **(3,1)** ; (4,2) ; (6,0) }



Our representation

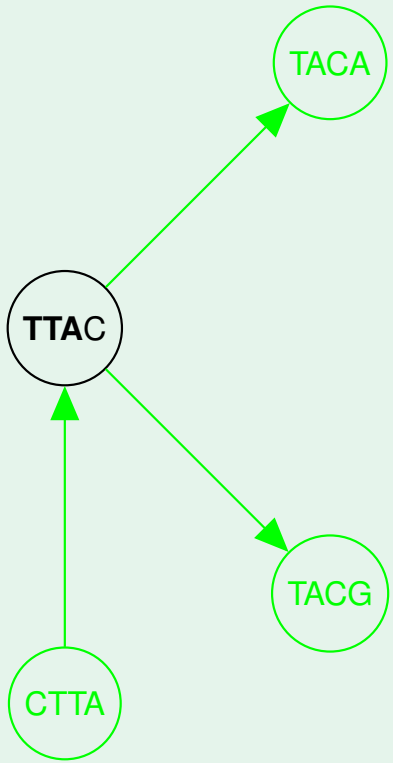
Example

k-mers set

1: AGCTTA
2: CTTACA
3: **CTTA**CG
4: GCTTAC
5: TACGTA
6: TTACGT

PgSA Index

{ (1,3) ; (2,1) ; **(3,1)** ; (4,2) ; (6,0) }



Our representation

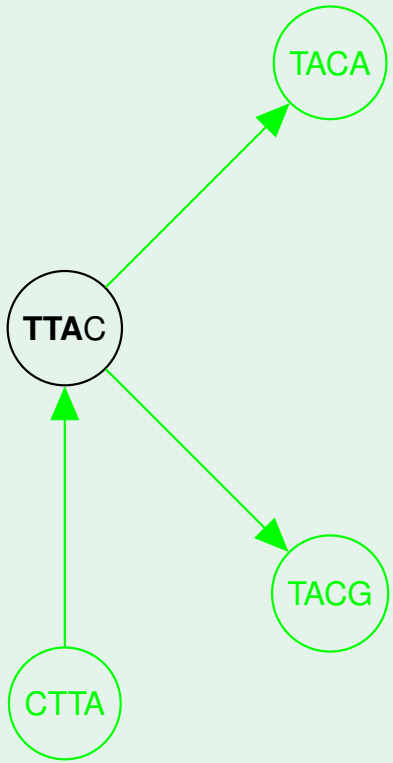
Example

k-mers set

1: AGCTTA
2: CTTACA
3: CTTACG
4: GC**TTA**C
5: TACGTA
6: TTACGT

PgSA Index

{ (1,3) ; (2,1) ; (3,1) ; **(4,2)** ; (6,0) }



Our representation

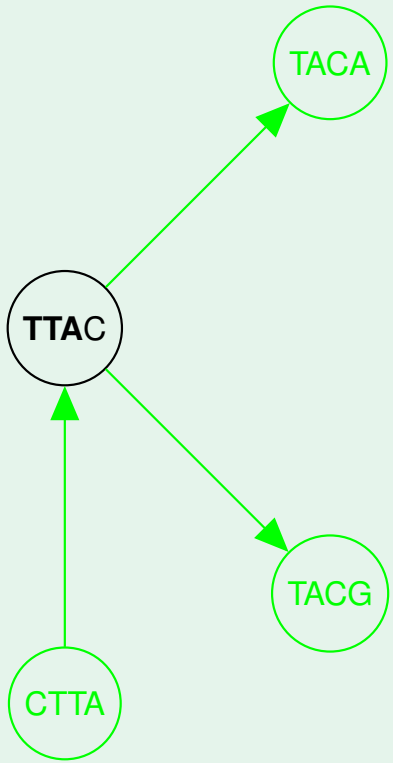
Example

k-mers set

1: AGCTTA
2: CTTACA
3: CTTACG
4: G**C**TTAC
5: TACGTA
6: TTACGT

PgSA Index

{ (1,3) ; (2,1) ; (3,1) ; **(4,2)** ; (6,0) }



Our representation

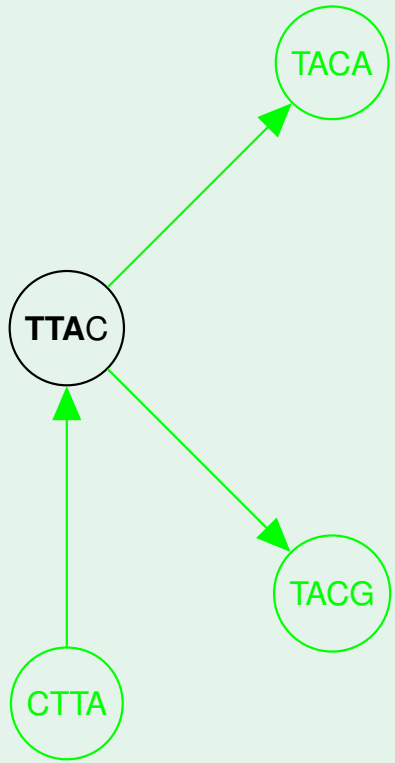
Example

k-mers set

1: AGCTTA
2: CTTACA
3: CTTACG
4: G**C**T**TA**C
5: TACGTA
6: TTACGT

PgSA Index

{ (1,3) ; (2,1) ; (3,1) ; **(4,2)** ; (6,0) }



Our representation

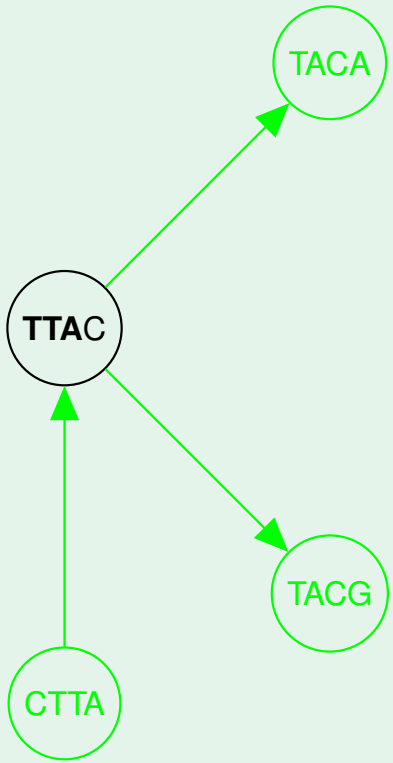
Example

k-mers set

1: AGCTTA
2: CTTACA
3: CTTACG
4: GCTTAC
5: TACGTA
6: **TTA**CGT

PgSA Index

{ (1,3) ; (2,1) ; (3,1) ; (4,2) ; **(6,0)** }



Our representation

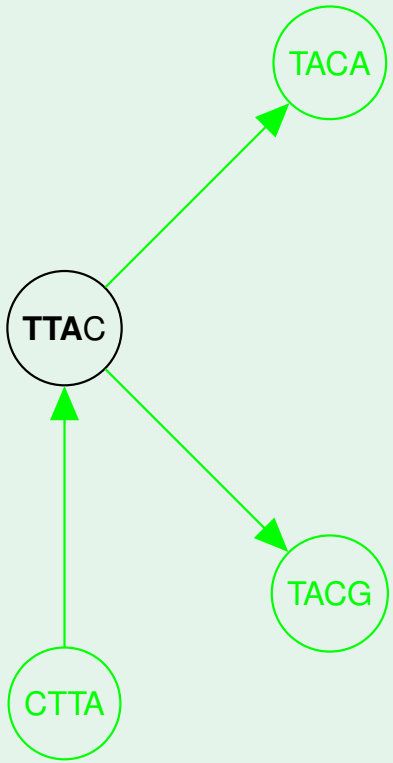
Example

k-mers set

1: AGCTTA
2: CTTACA
3: CTTACG
4: GCTTAC
5: TACGTA
6: **TTA**CGT

PgSA Index

{ (1,3) ; (2,1) ; (3,1) ; (4,2) ; **(6,0)** }



Workflow

5 steps:

1. Correct the short reads (with QuorUM [Marçais et al., 2015])
2. Filter out corrected short reads containing weak K-mers, and index solid K-mers with PgSA
3. Align the remaining short reads to the long reads, to find seeds (with BLASR [Chaisson and Tesler, 2012])
4. Merge the overlapping seeds, and link them together, by traversing the graph
5. Extend the obtained corrected long read, on the left (resp. right) of the leftmost (resp. rightmost) seed

Workflow

5 steps:

1. Correct the short reads (with QuorUM [Marçais et al., 2015])
2. Filter out corrected short reads containing weak K-mers, and index solid K-mers with PgSA
3. Align the remaining short reads to the long reads, to find seeds (with BLASR [Chaisson and Tesler, 2012])
4. Merge the overlapping seeds, and link them together, by traversing the graph
5. Extend the obtained corrected long read, on the left (resp. right) of the leftmost (resp. rightmost) seed

Workflow

5 steps:

1. Correct the short reads (with QuorUM [Marçais et al., 2015])
2. Filter out corrected short reads containing weak K-mers, and index solid K-mers with PgSA
3. Align the remaining short reads to the long reads, to find seeds (with BLASR [Chaisson and Tesler, 2012])
4. Merge the overlapping seeds, and link them together, by traversing the graph
5. Extend the obtained corrected long read, on the left (resp. right) of the leftmost (resp. rightmost) seed

Workflow

5 steps:

1. Correct the short reads (with QuorUM [Marçais et al., 2015])
2. Filter out corrected short reads containing weak K-mers, and index solid K-mers with PgSA
3. Align the remaining short reads to the long reads, to find seeds (with BLASR [Chaisson and Tesler, 2012])
4. Merge the overlapping seeds, and link them together, by traversing the graph
5. Extend the obtained corrected long read, on the left (resp. right) of the leftmost (resp. rightmost) seed

Workflow

5 steps:

1. Correct the short reads (with QuorUM [Marçais et al., 2015])
2. Filter out corrected short reads containing weak K-mers, and index solid K-mers with PgSA
3. Align the remaining short reads to the long reads, to find seeds (with BLASR [Chaisson and Tesler, 2012])
4. Merge the overlapping seeds, and link them together, by traversing the graph
5. Extend the obtained corrected long read, on the left (resp. right) of the leftmost (resp. rightmost) seed

Step 4: Seeds merging and linking

- Process the seeds with overlapping mapping positions:
  - Perfect overlap: merge
  - Otherwise: keep the best seed
- Seeds are used as anchor points on the graph
- The graph is traversed to link the seeds together, and correct uncovered regions

Step 4: Seeds merging and linking

- Process the seeds with overlapping mapping positions:
  - Perfect overlap: merge
  - Otherwise: keep the best seed
- Seeds are used as anchor points on the graph
- The graph is traversed to link the seeds together, and correct uncovered regions

Step 4: Seeds merging and linking

- Process the seeds with overlapping mapping positions:
  - Perfect overlap: merge
  - Otherwise: keep the best seed
- Seeds are used as anchor points on the graph
- The graph is traversed to link the seeds together, and correct uncovered regions

Step 4: Seeds merging and linking

- Process the seeds with overlapping mapping positions:
  - Perfect overlap: merge
  - Otherwise: keep the best seed
- Seeds are used as anchor points on the graph
- The graph is traversed to link the seeds together, and correct uncovered regions

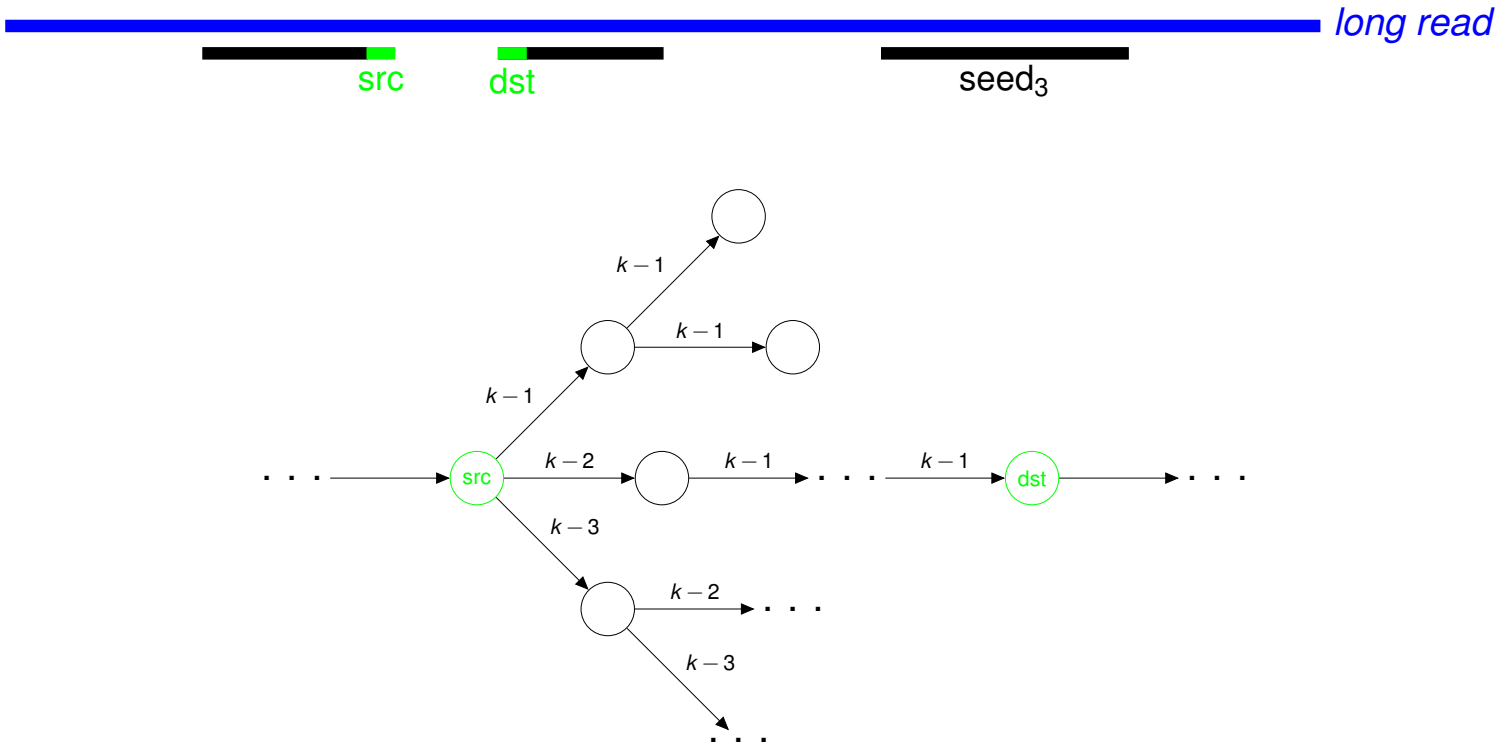
Step 4: Seeds merging and linking

- Process the seeds with overlapping mapping positions:
  - Perfect overlap: merge
  - Otherwise: keep the best seed
- Seeds are used as anchor points on the graph
- The graph is traversed to link the seeds together, and correct uncovered regions

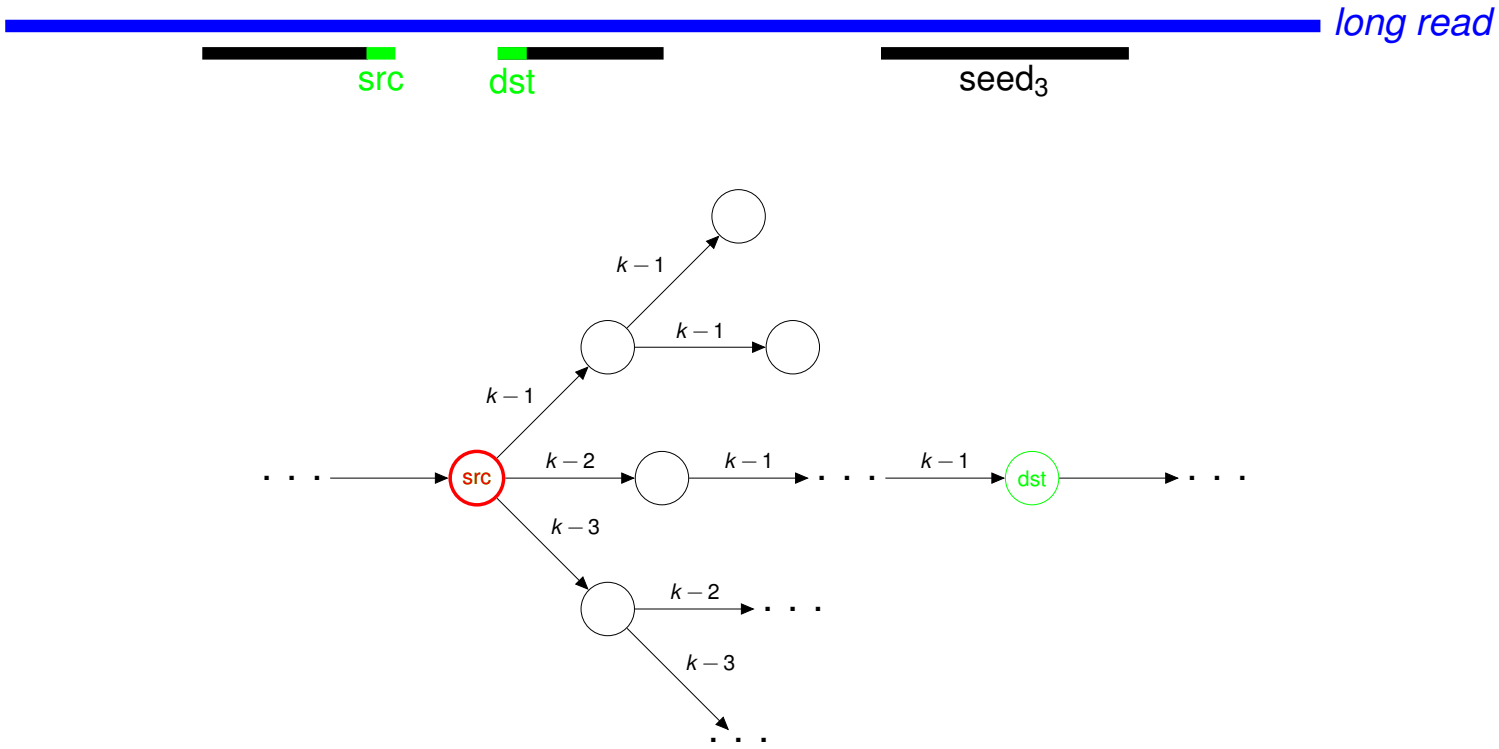
Step 4: Seeds linking

Step 4: Seeds linking

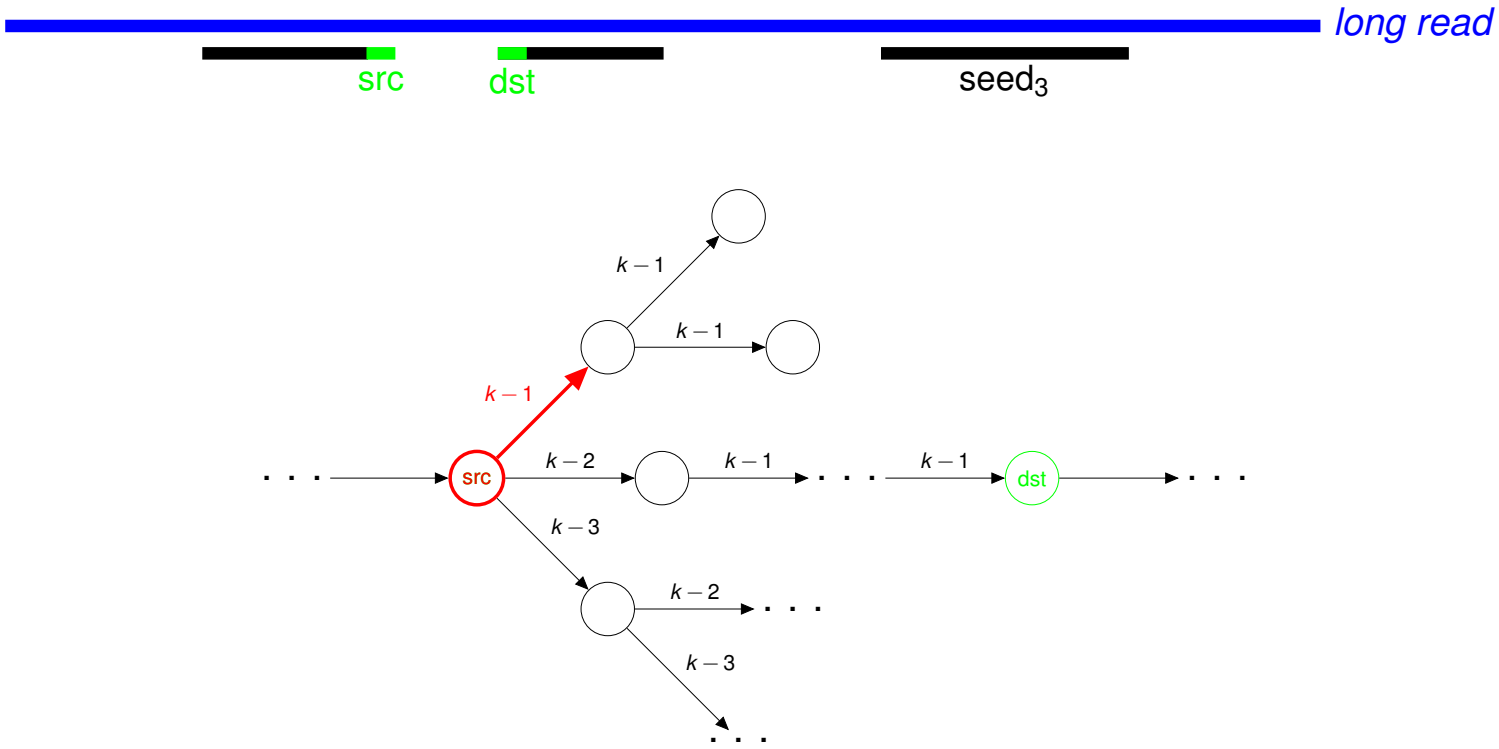
Step 4: Seeds linking



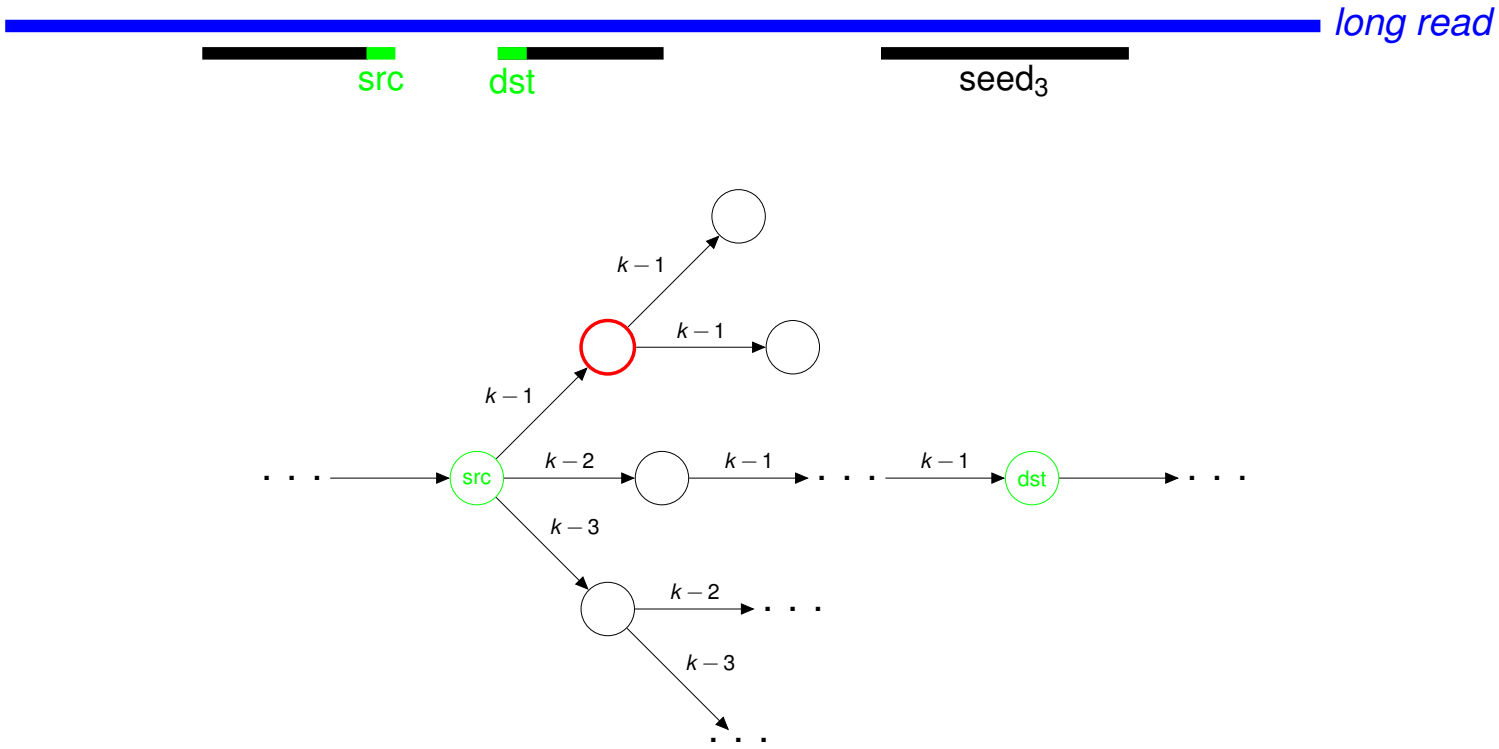
Step 4: Seeds linking



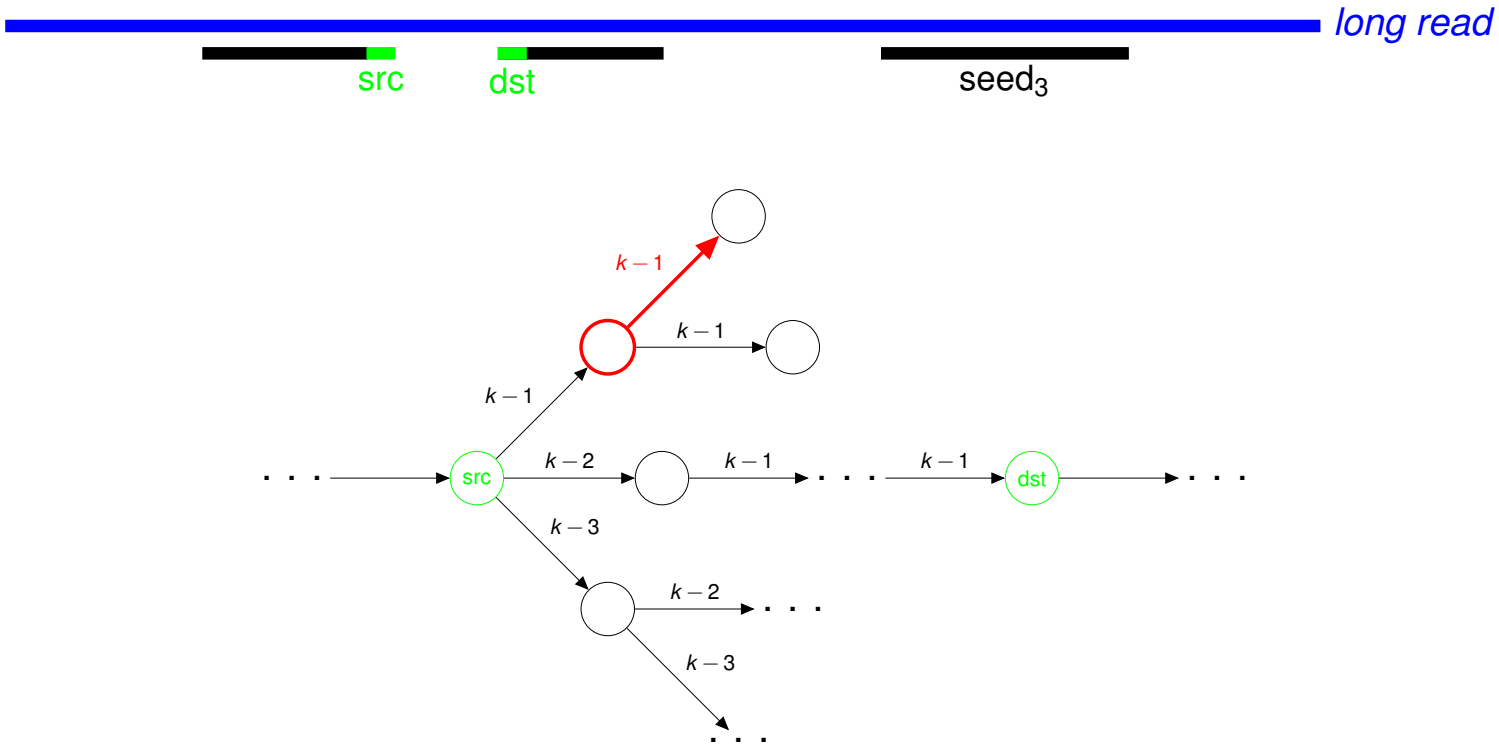
Step 4: Seeds linking



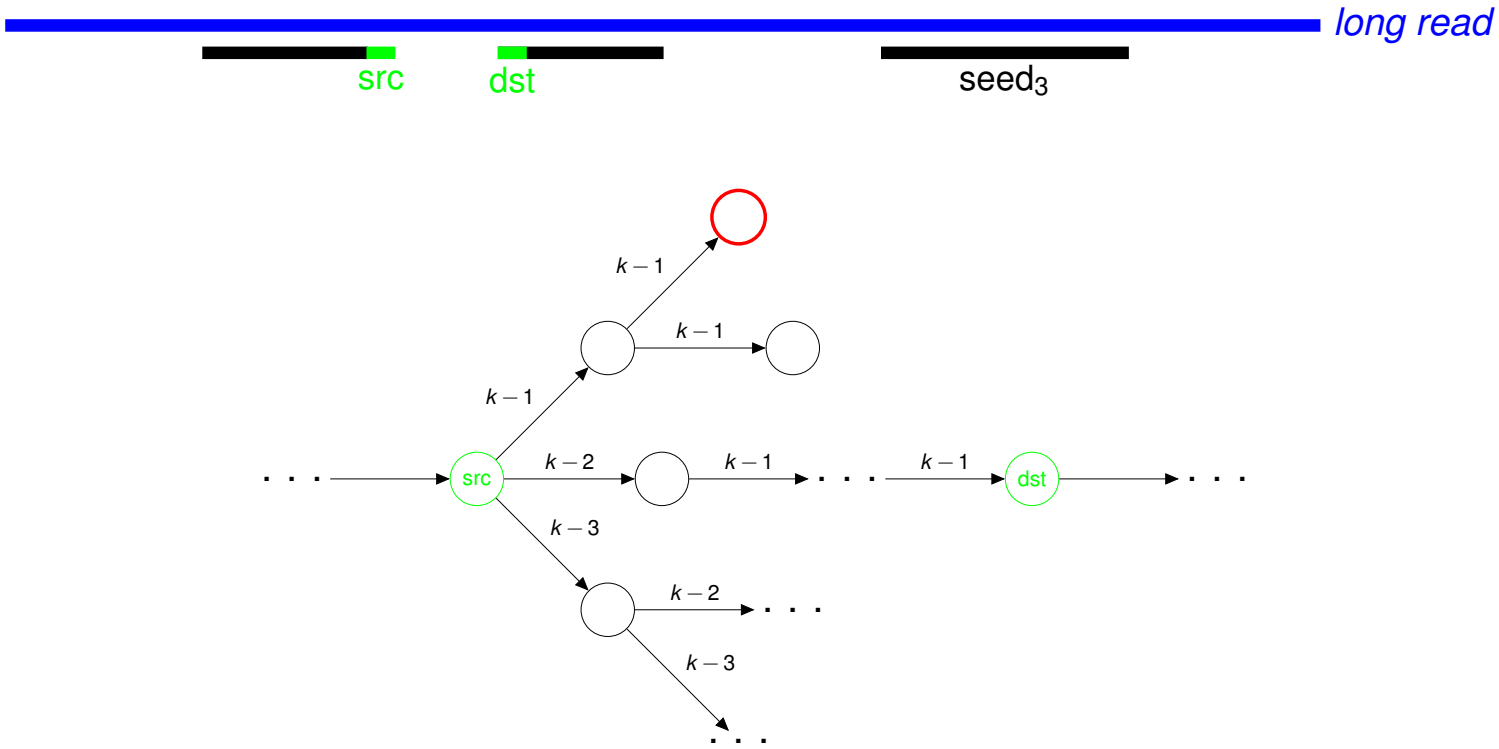
Step 4: Seeds linking



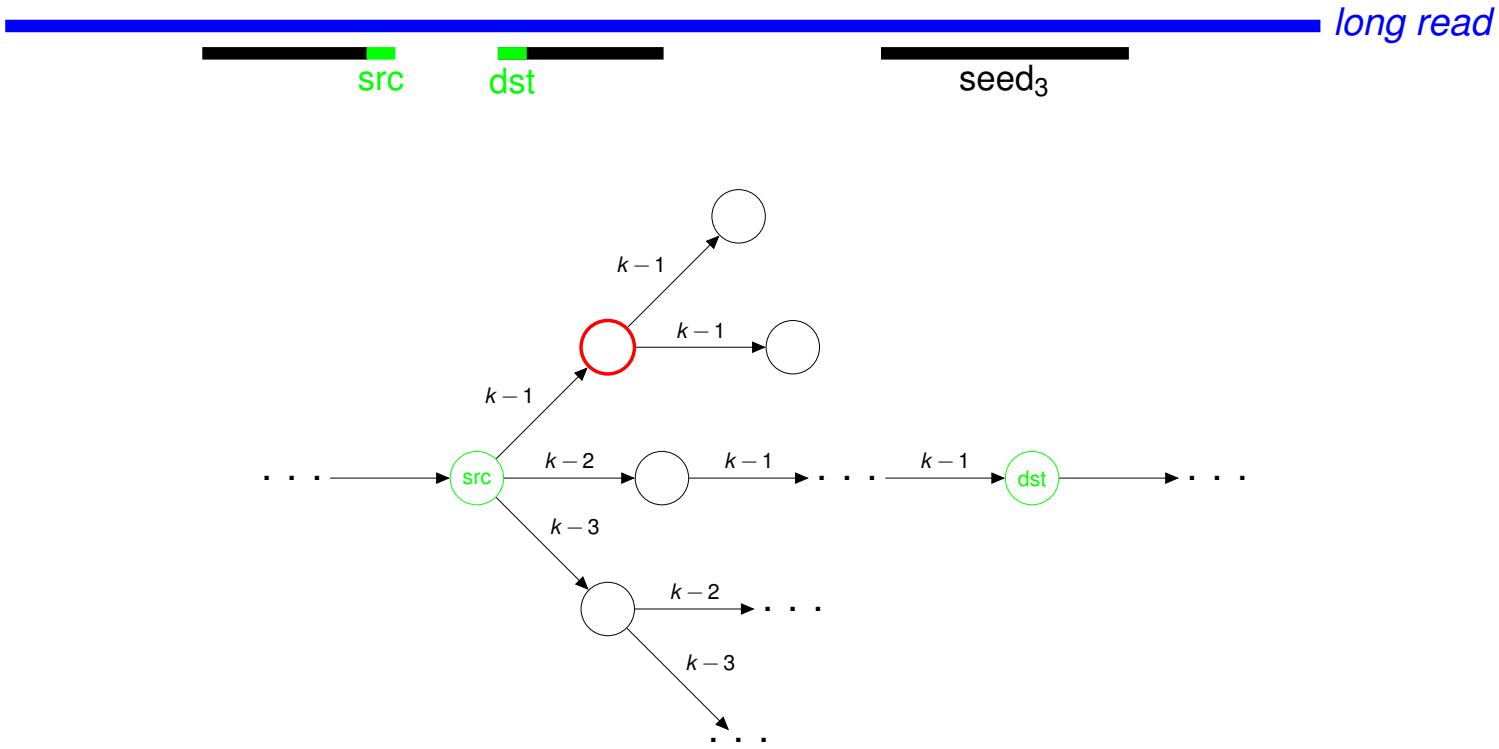
Step 4: Seeds linking



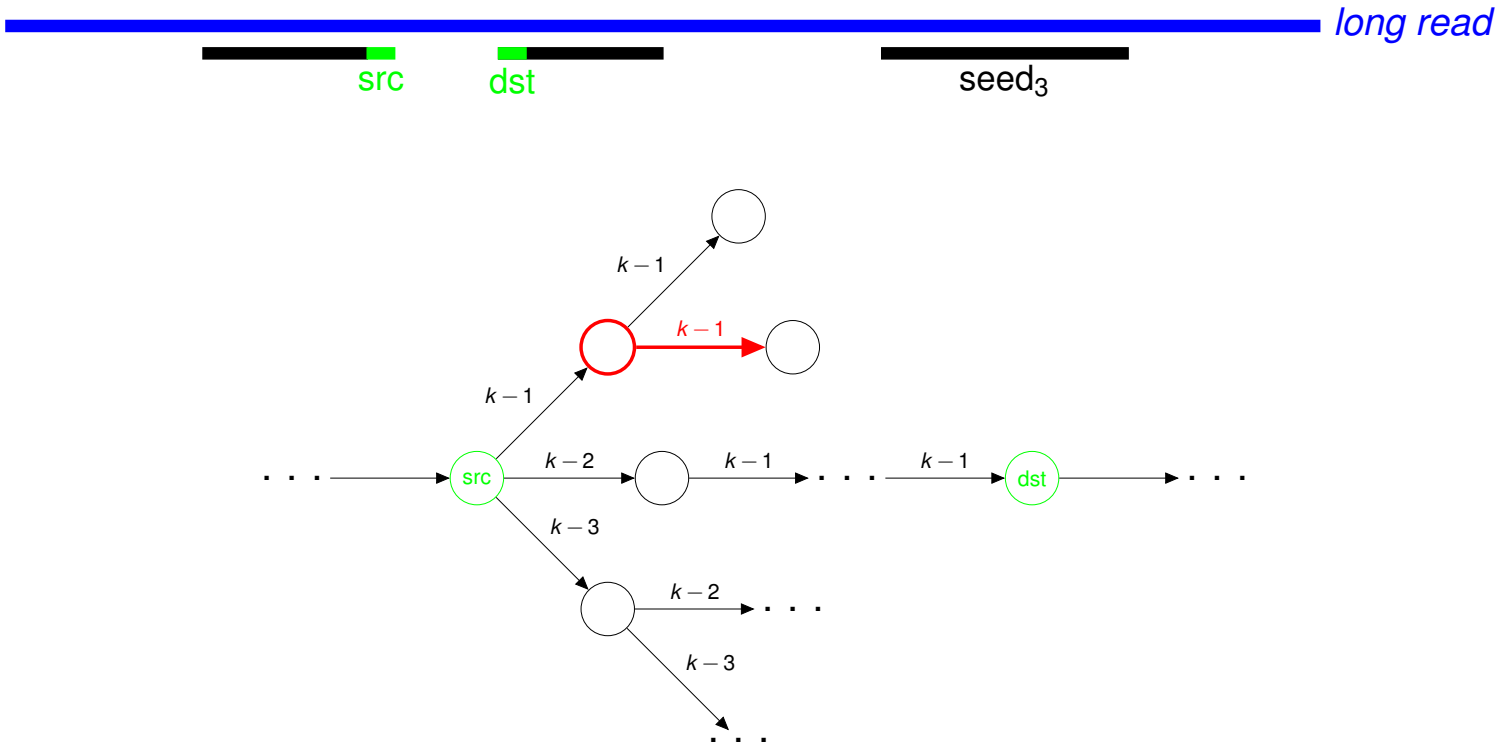
Step 4: Seeds linking



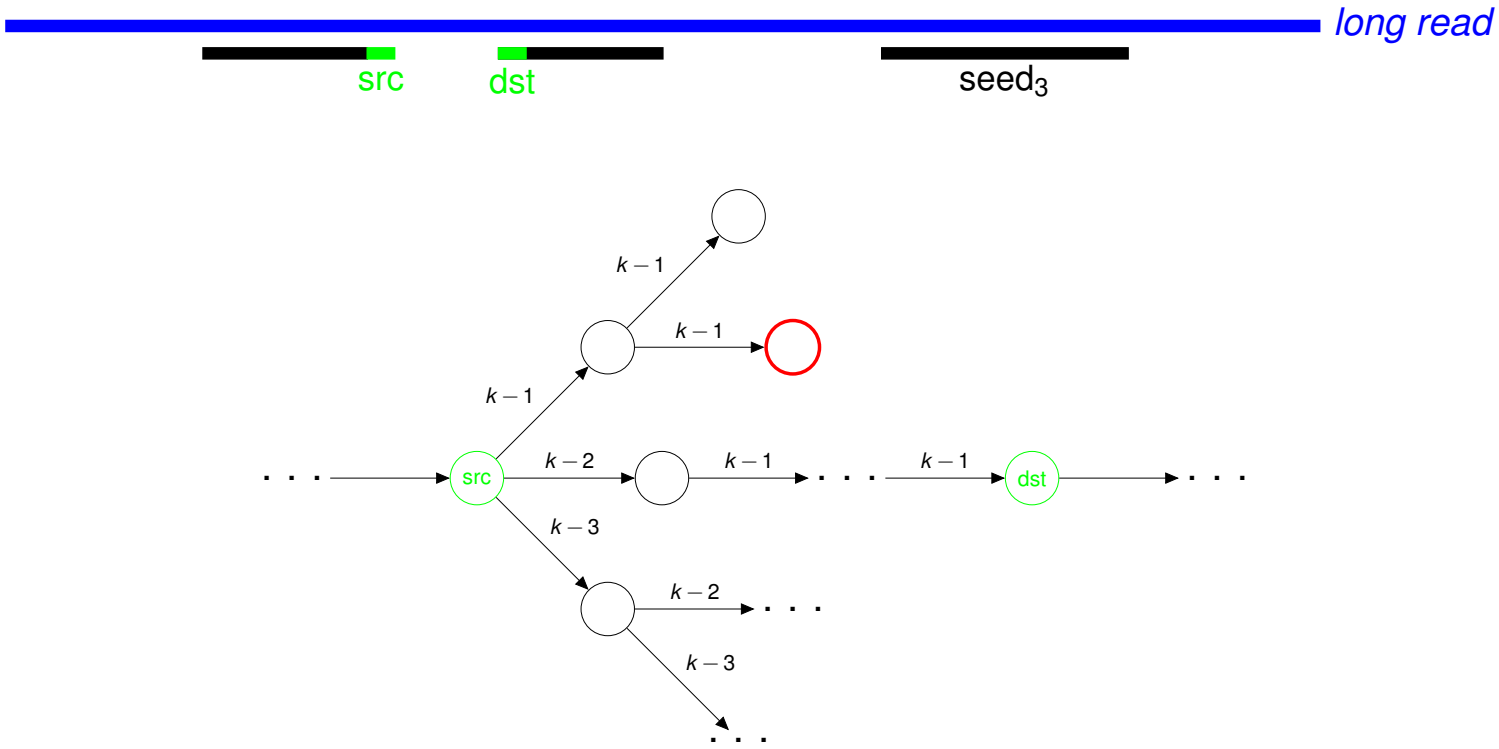
Step 4: Seeds linking



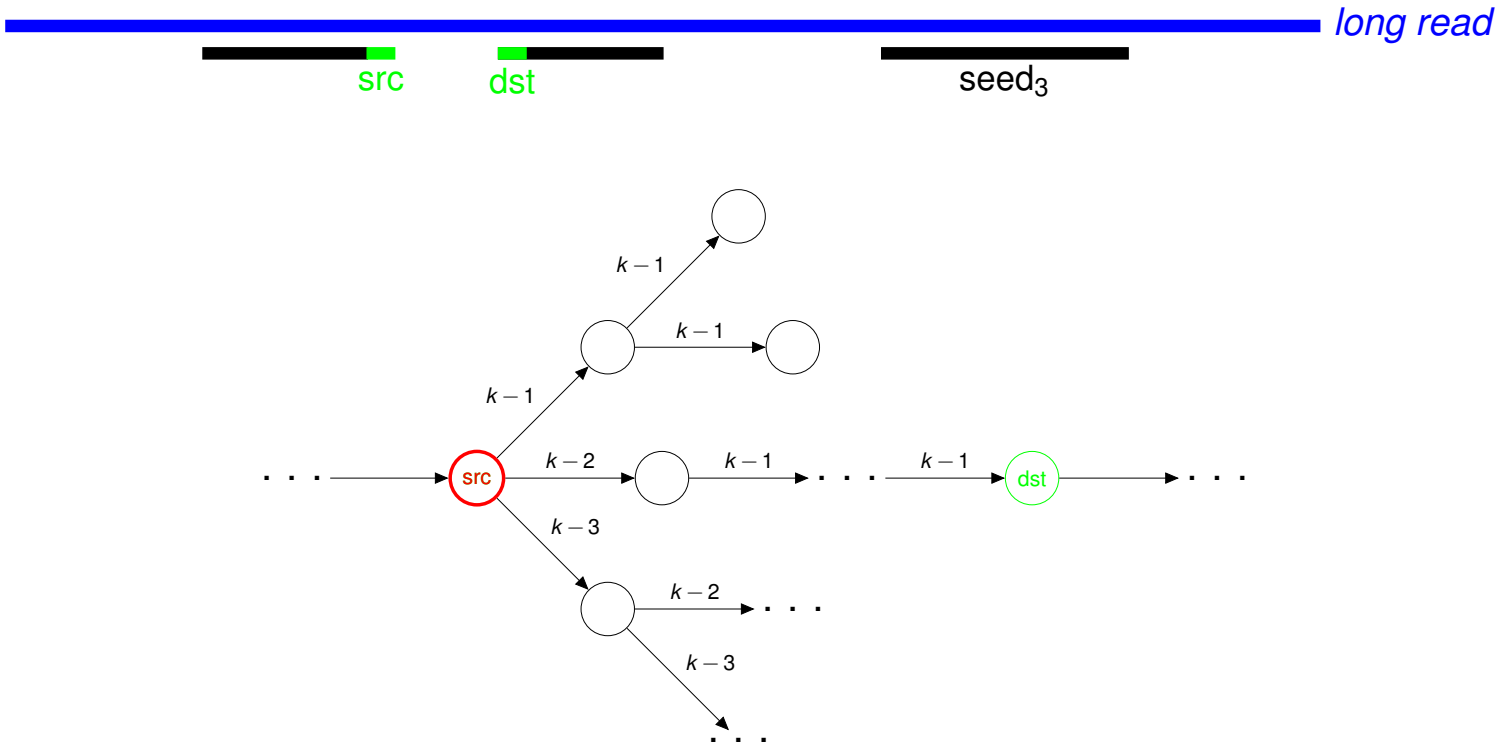
Step 4: Seeds linking



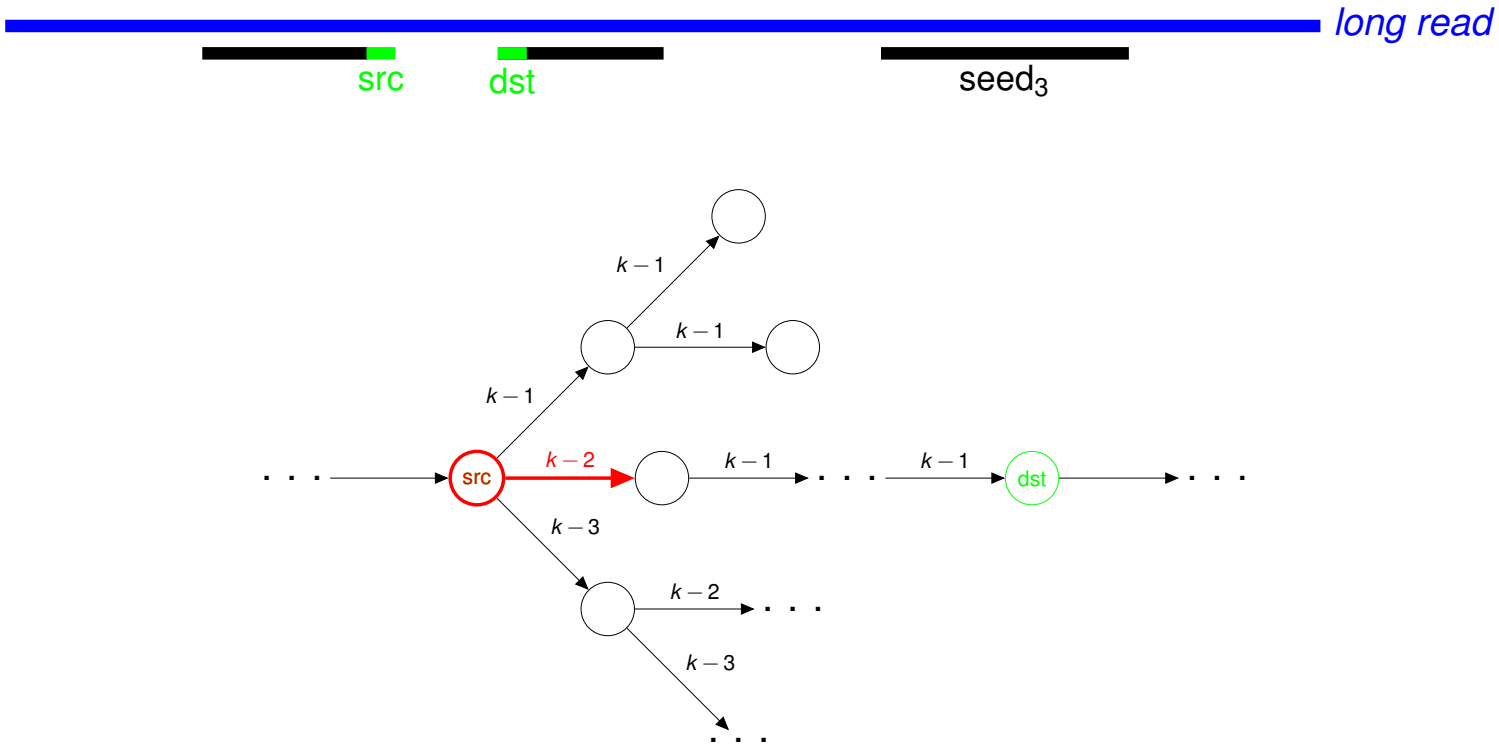
Step 4: Seeds linking



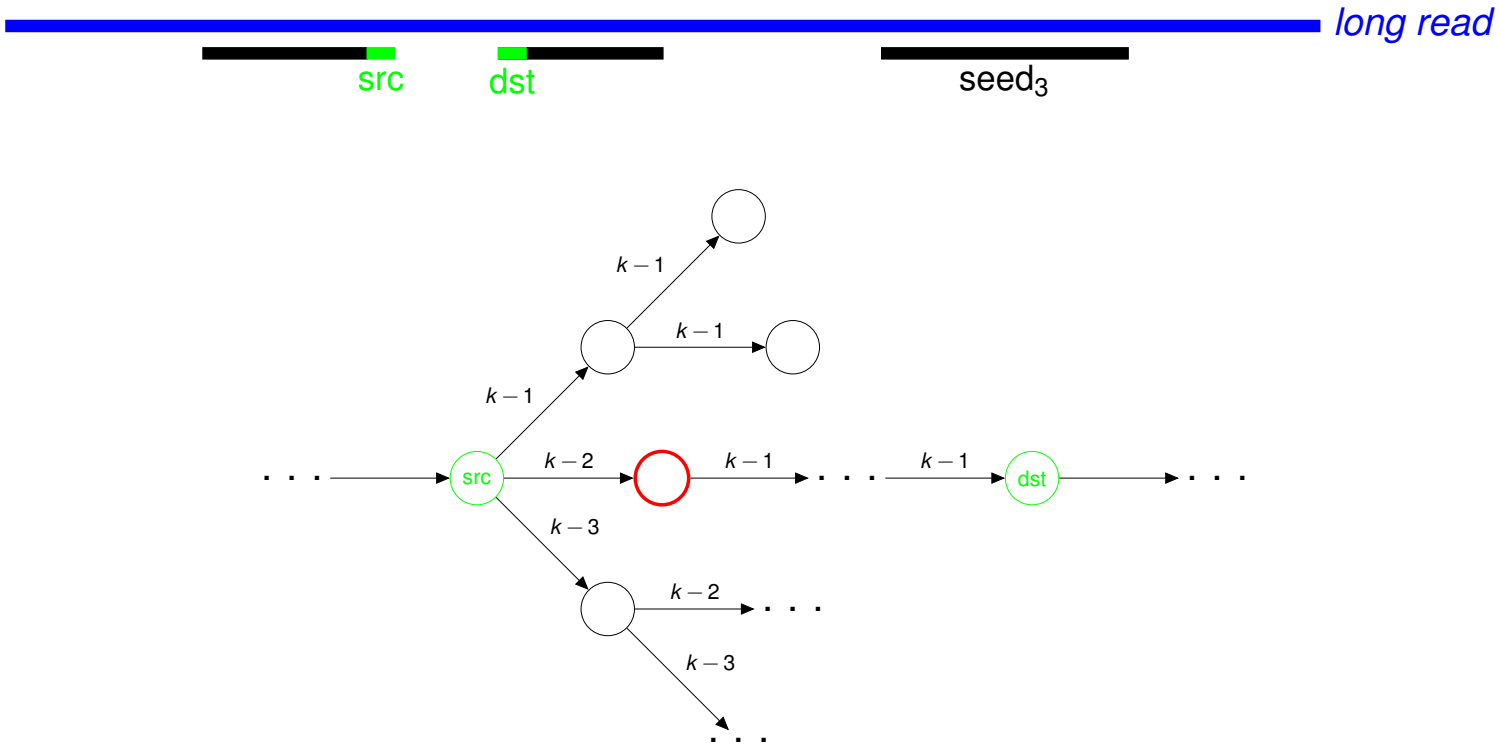
Step 4: Seeds linking



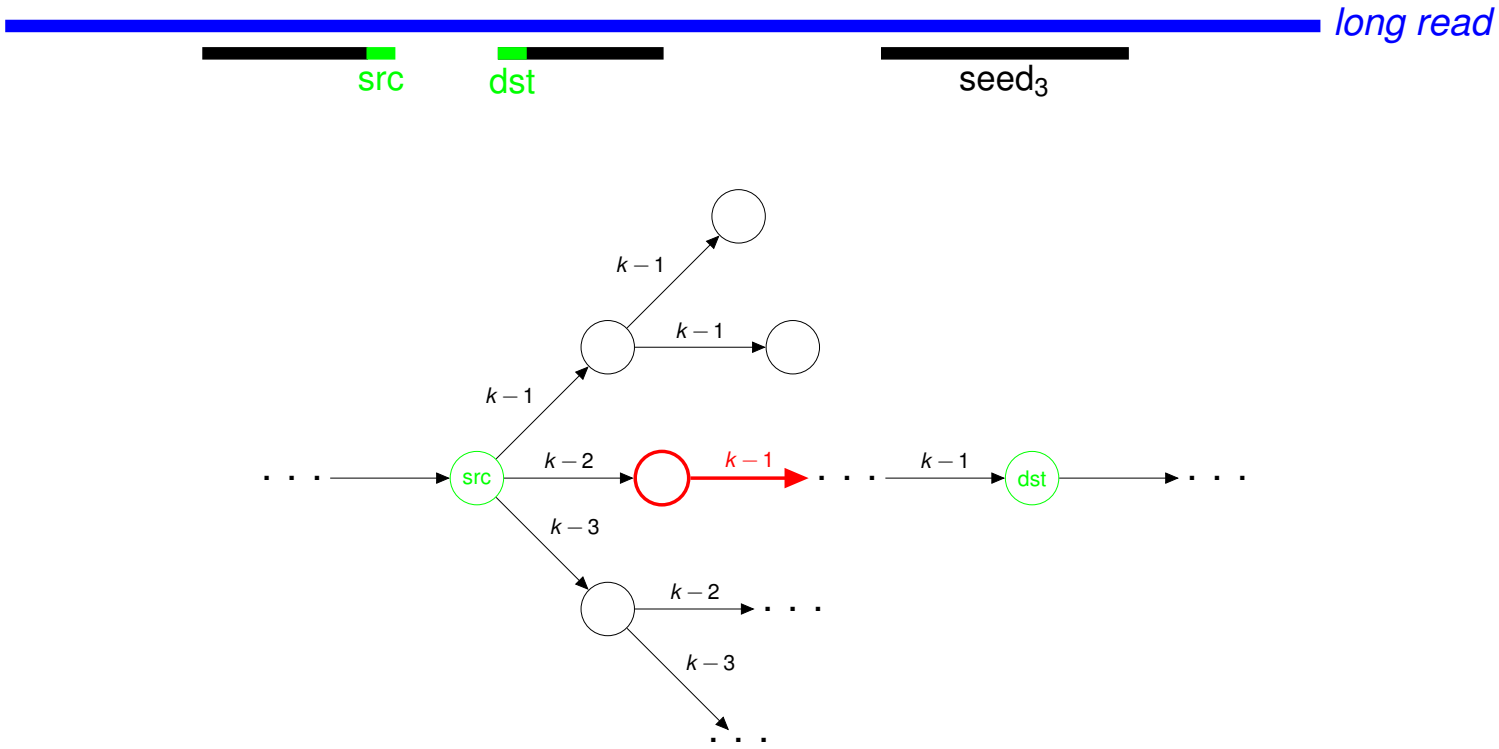
Step 4: Seeds linking



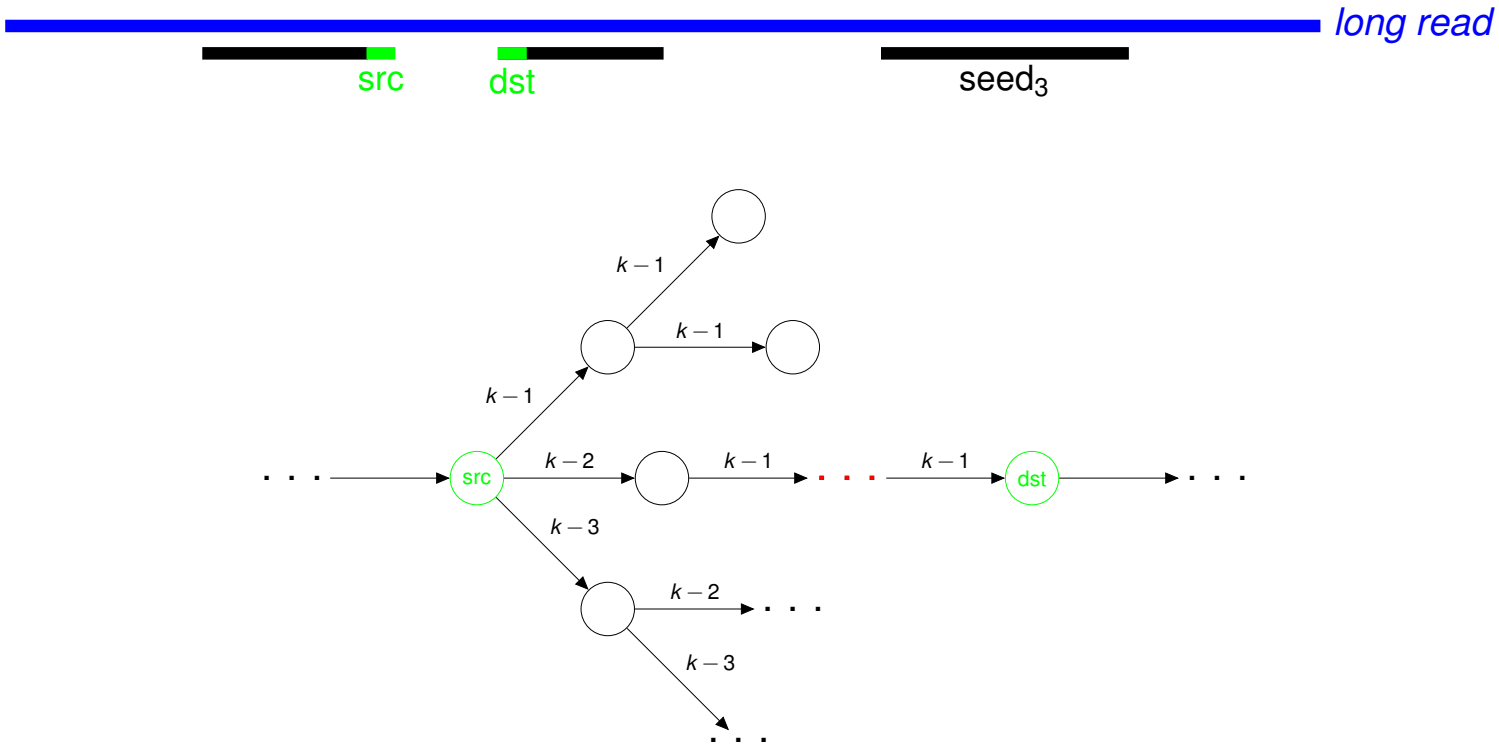
Step 4: Seeds linking



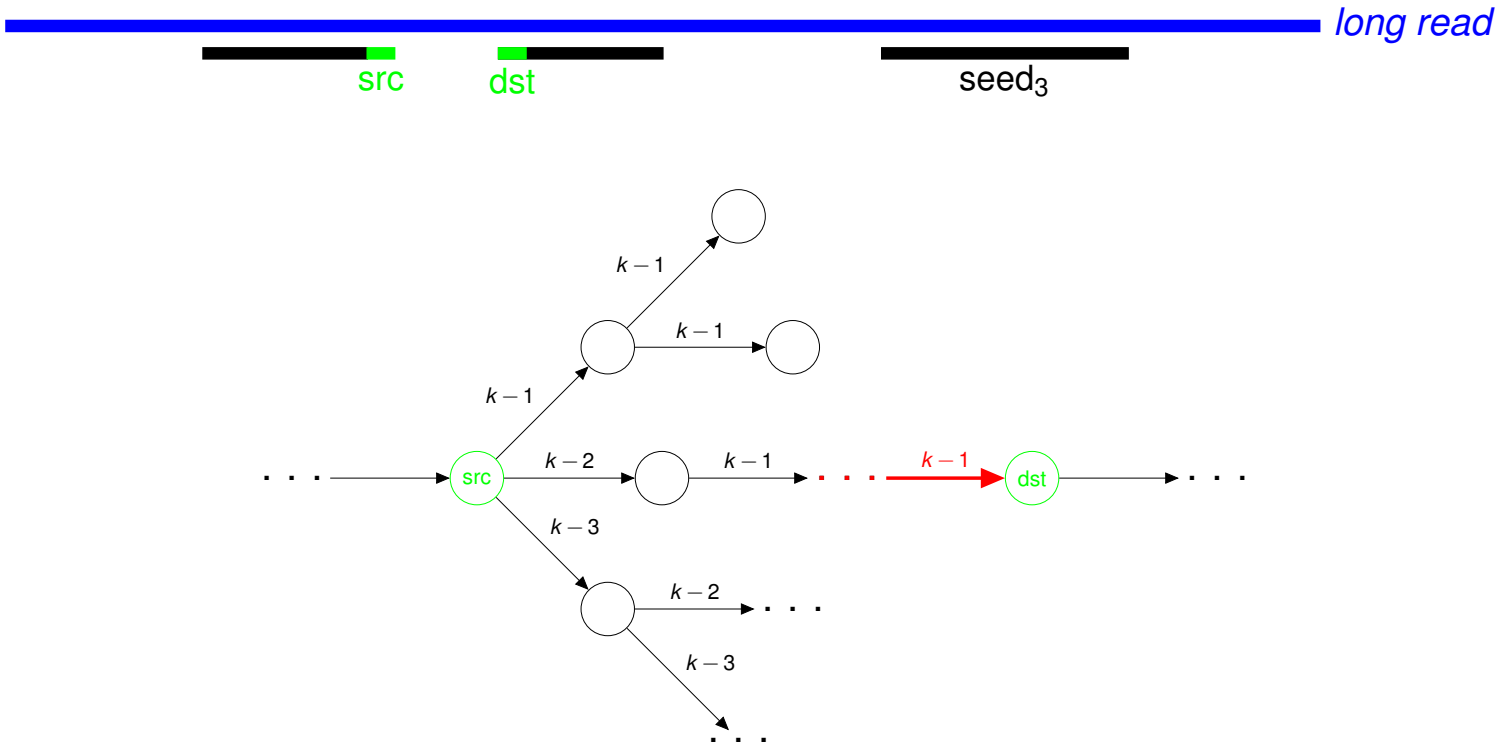
Step 4: Seeds linking



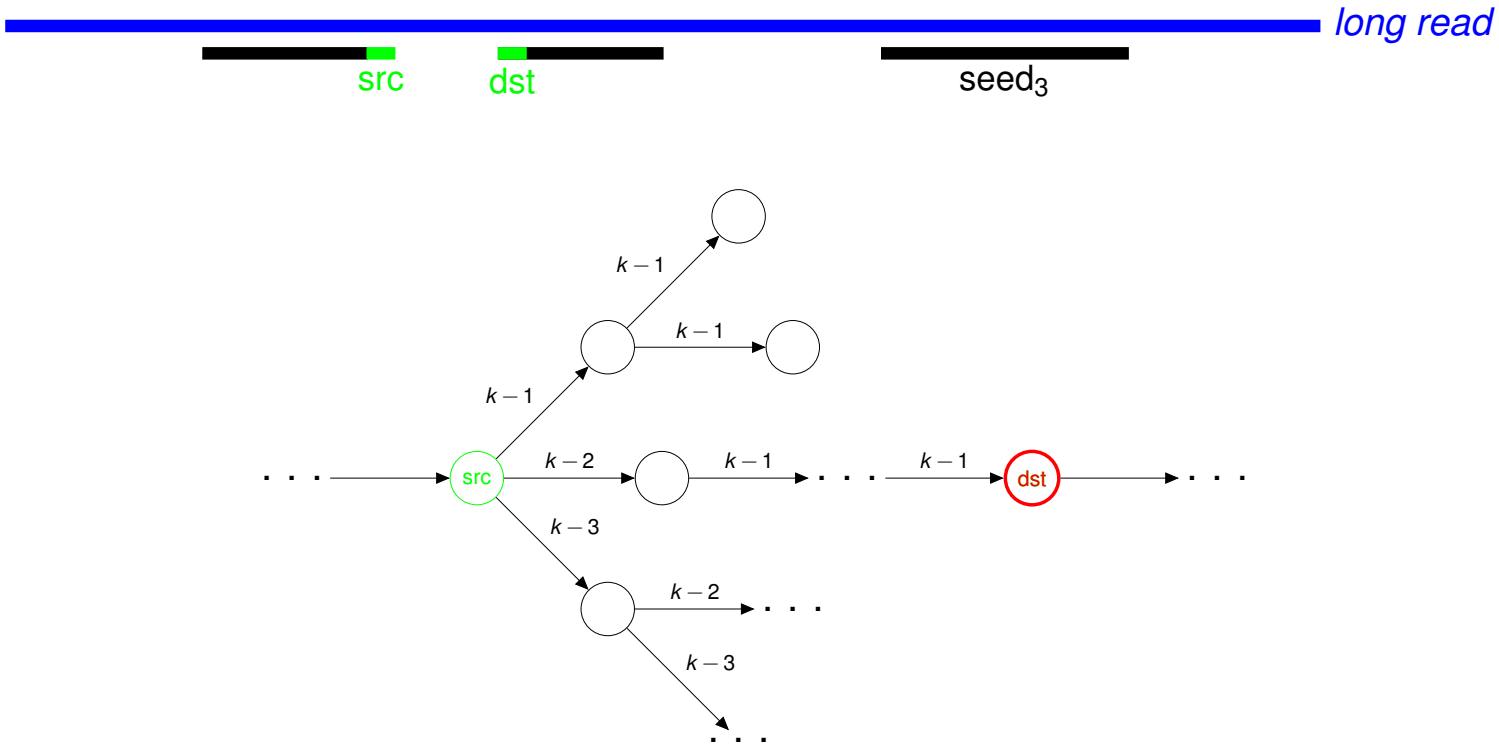
Step 4: Seeds linking



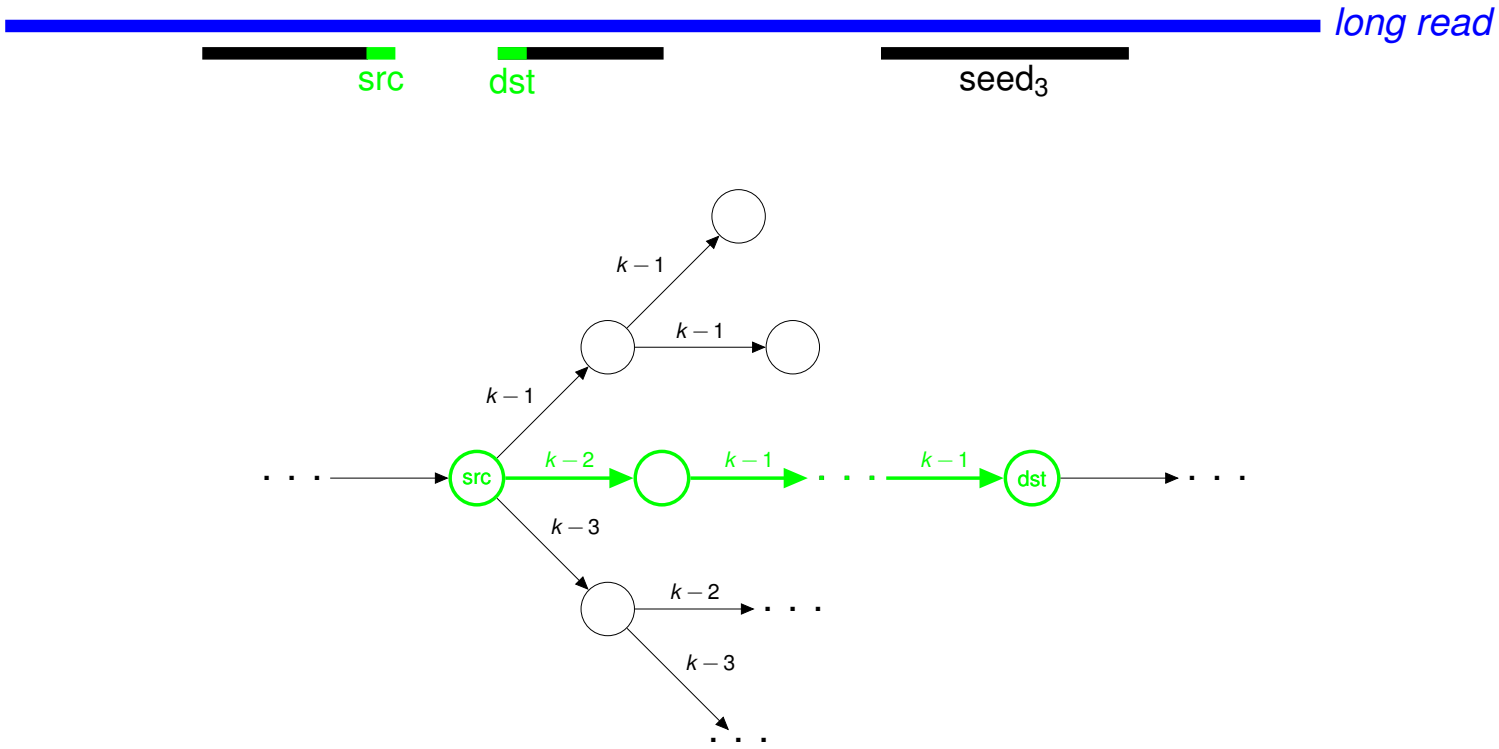
Step 4: Seeds linking



Step 4: Seeds linking



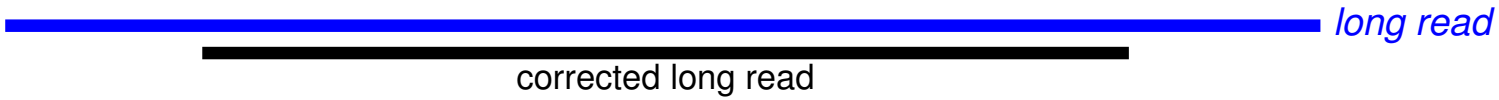
Step 4: Seeds linking



Step 4: Seeds linking

Step 4: Seeds linking

Step 4: Seeds linking



Step 5: Tips extension

- Seeds don't always map right at the beginning and at the end of the long read
- Once all the seeds have been linked, HG-CoLoR keeps on traversing the graph
- The traversal stops when the borders of the long read or a branching path are reached

Step 5: Tips extension

- Seeds don't always map right at the beginning and at the end of the long read
- Once all the seeds have been linked, HG-CoLoR keeps on traversing the graph
- The traversal stops when the borders of the long read or a branching path are reached

Step 5: Tips extension

- Seeds don't always map right at the beginning and at the end of the long read
- Once all the seeds have been linked, HG-CoLoR keeps on traversing the graph
- The traversal stops when the borders of the long read or a branching path are reached

Remark

- Some seeds might be impossible to link together
- $\Rightarrow$ Fill in the uncorrected part of the long reads with original bases
- Propose a trim/split output to remove uncorrected bases

Remark

- Some seeds might be impossible to link together
- $\Rightarrow$ Fill in the uncorrected part of the long reads with original bases
- Propose a trim/split output to remove uncorrected bases

Remark

- Some seeds might be impossible to link together
- $\Rightarrow$ Fill in the uncorrected part of the long reads with original bases
- Propose a trim/split output to remove uncorrected bases

Comparison to the state-of-the-art

We compared HG-CoLoR to the following state-of-the-art error correction tools:

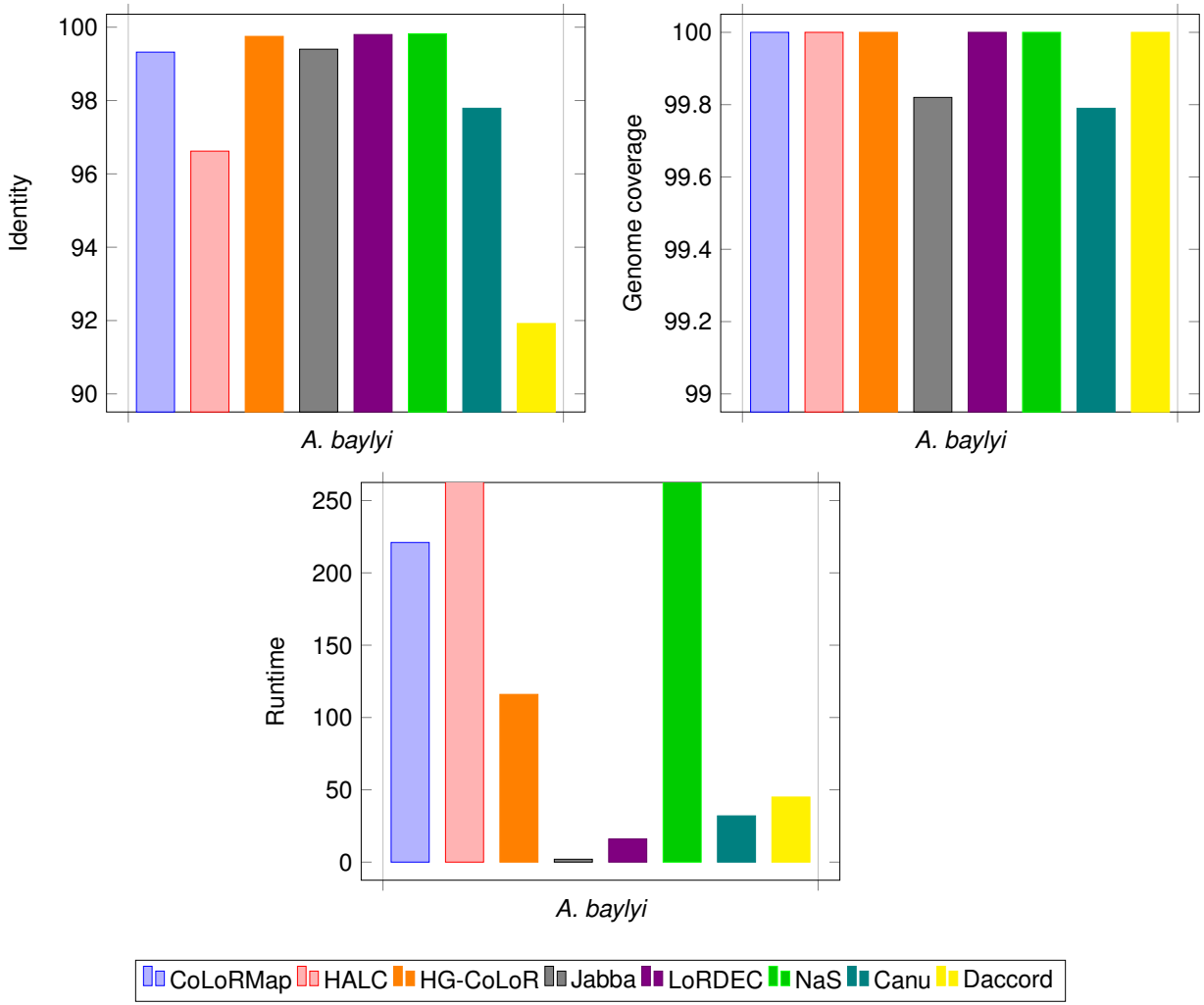
- CoLoRMap [Haghshenas et al., 2016]
- HALC [Bao and Lan, 2017]
- Jabba [Miclotte et al., 2016]
- LoRDEC [Salmela and Rivals, 2014]
- NaS [Madoui et al., 2015]
- Canu [Koren et al., 2017]
- Daccord [Tischler and Myers, 2017]

Datasets

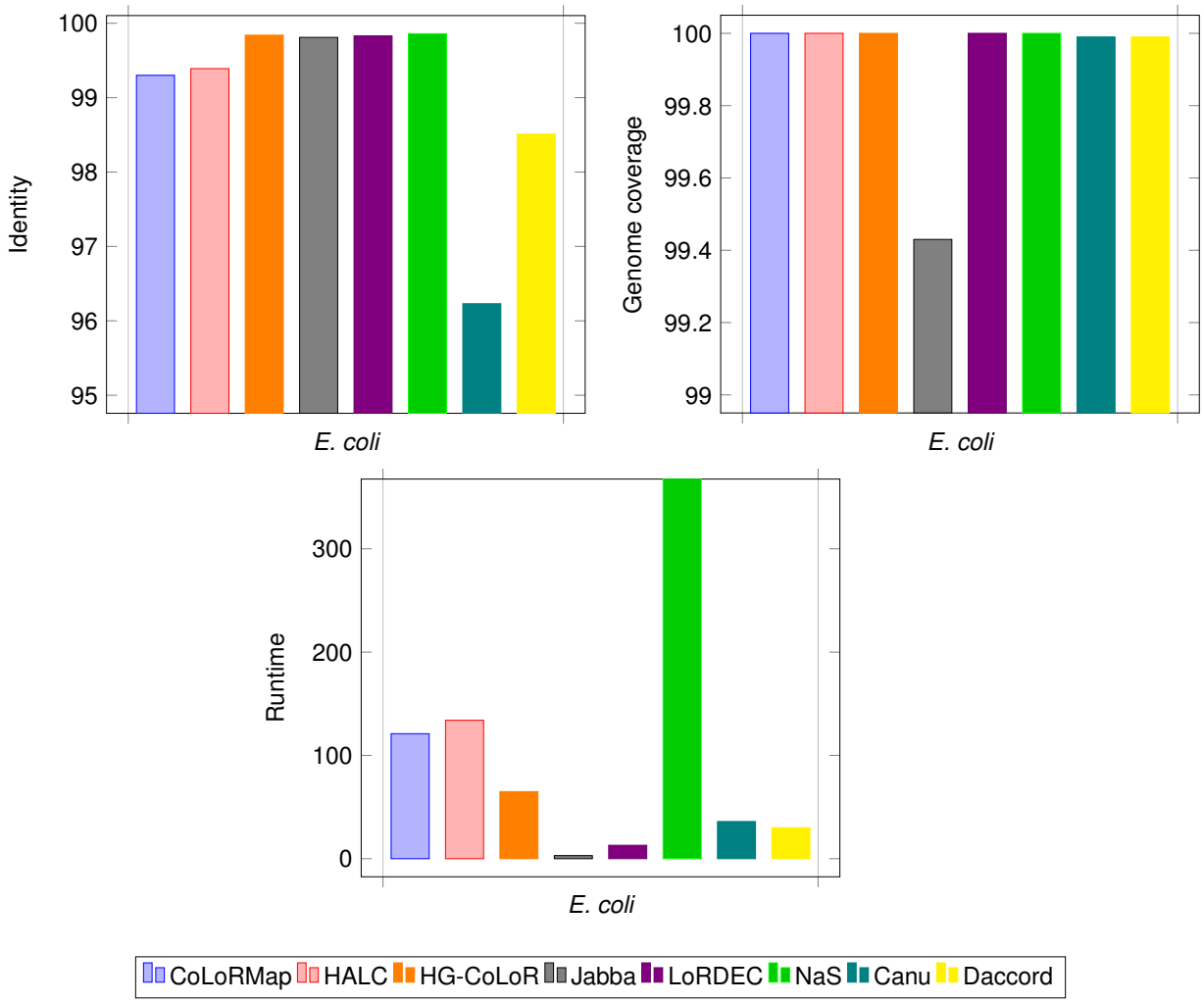
The different tools were compared on the following datasets:

Dataset	*A. baylyi*	*E. coli*	*S. cerevisiae*	*C. elegans*
Reference organism				
Strain	ADP1	K-12 substr. MG1655	W303	Bristol N2
Reference sequence	CR543861	NC_000913	scf7180000000{084-13}	GCA_000002985.3
Genome size	3.6 Mbp	4.6 Mbp	12.2 Mbp	100 Mbp
Real Oxford Nanopore data				
Coverage	106x	29x	95x	20x
Error rate	30%	20%	44%	29%
Illumina data				
Coverage	50x	50x	50x	50x
Read length	250	300	250	250

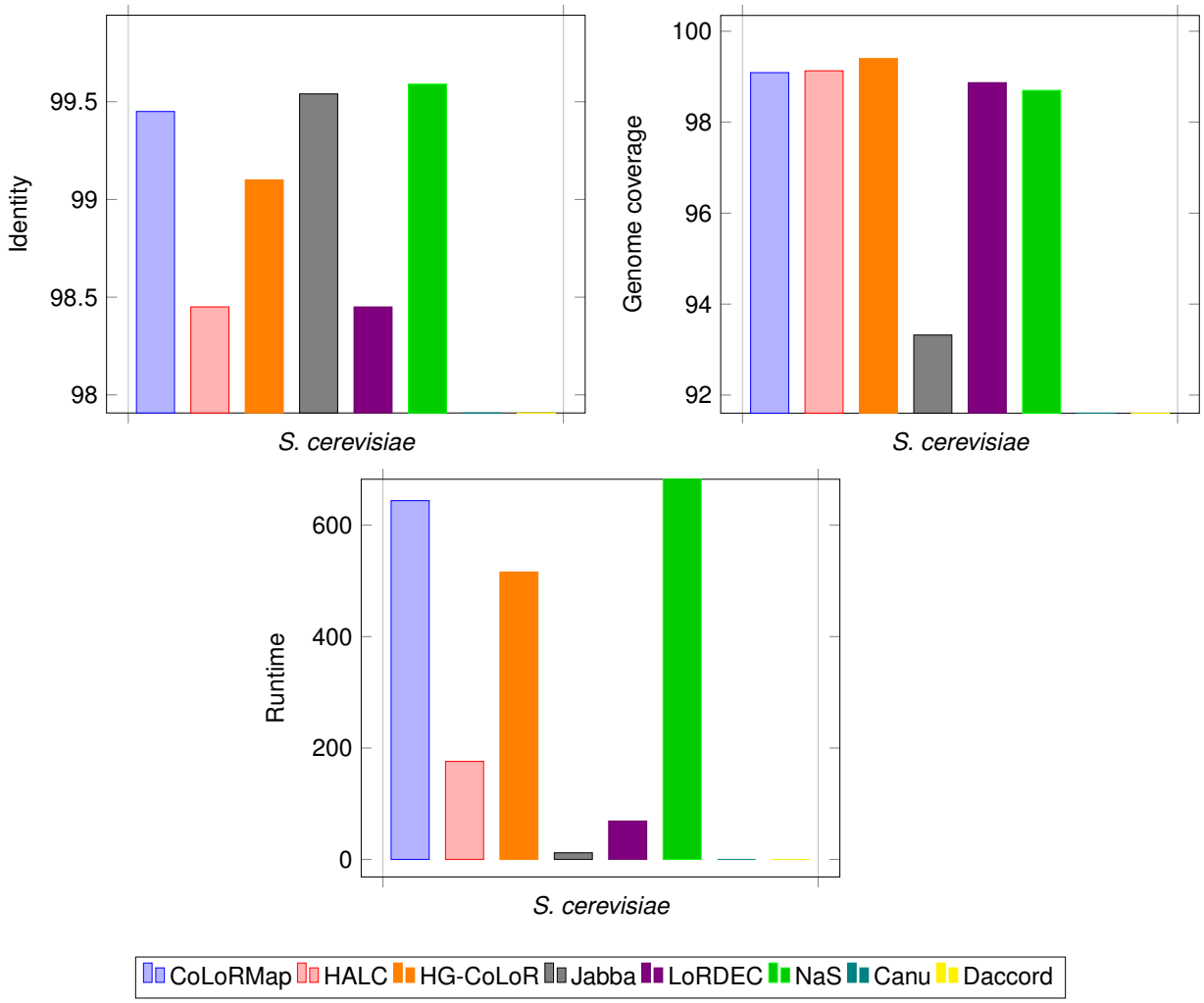
Comparison of the error correction



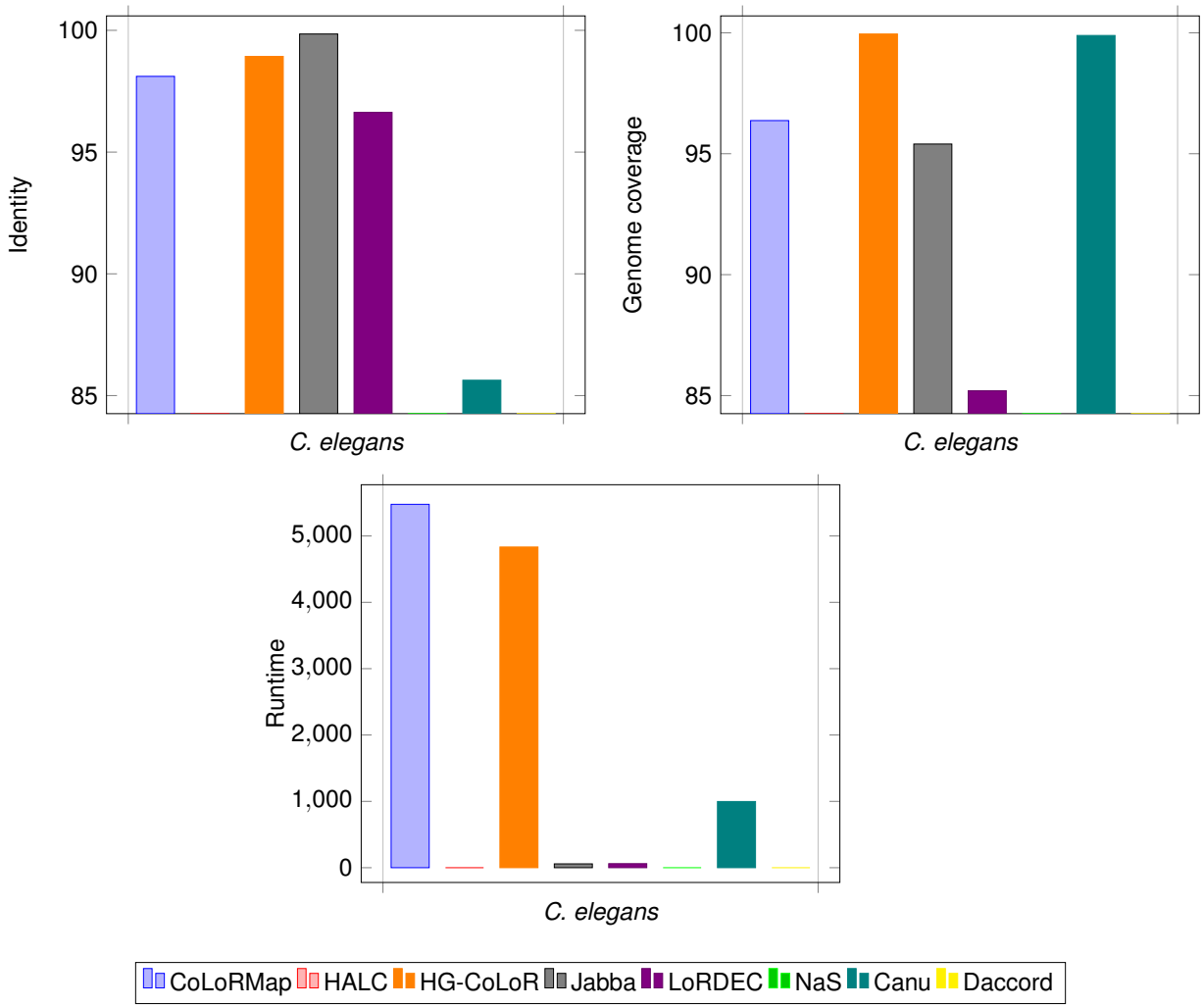
Comparison of the error correction



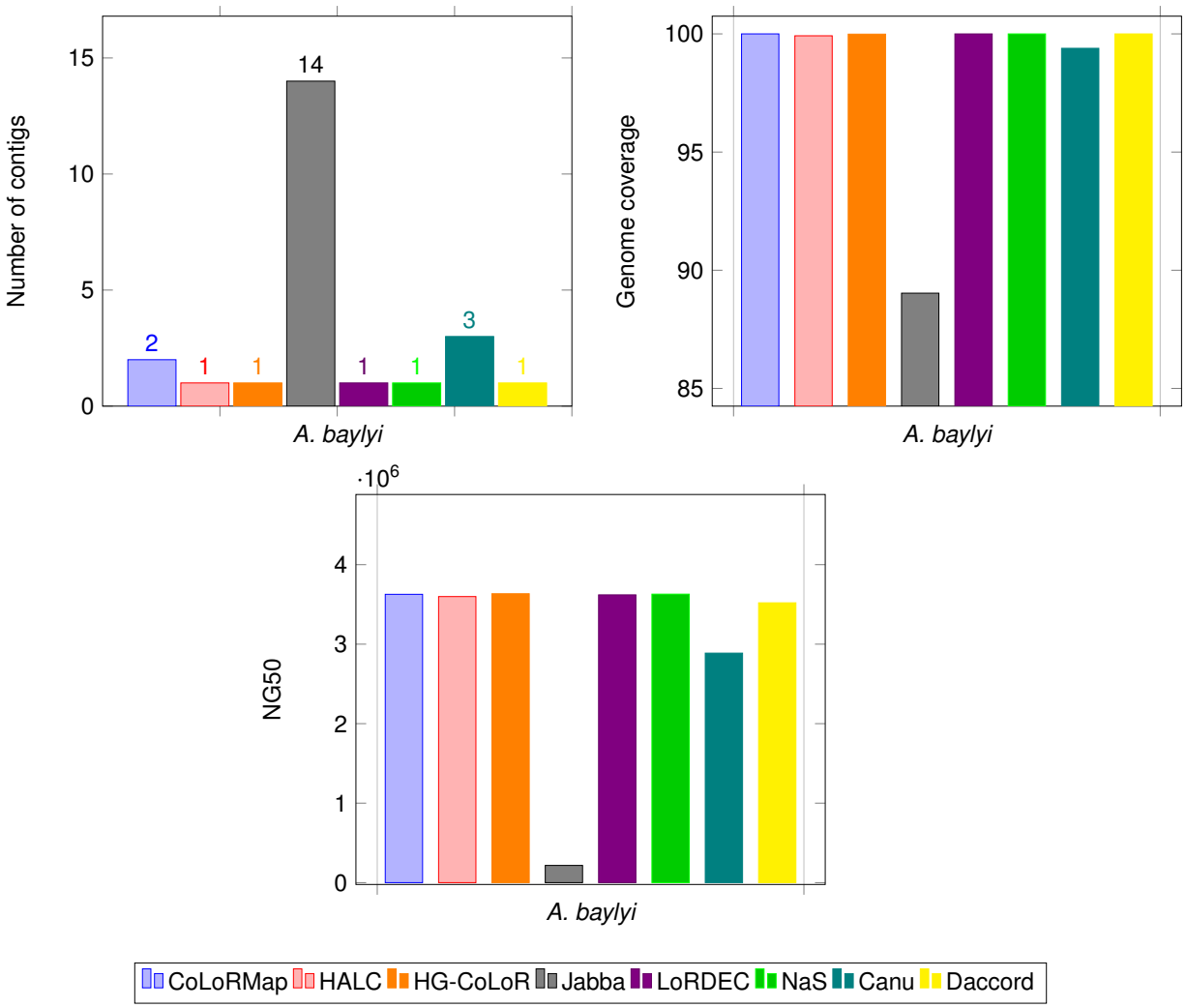
Comparison of the error correction



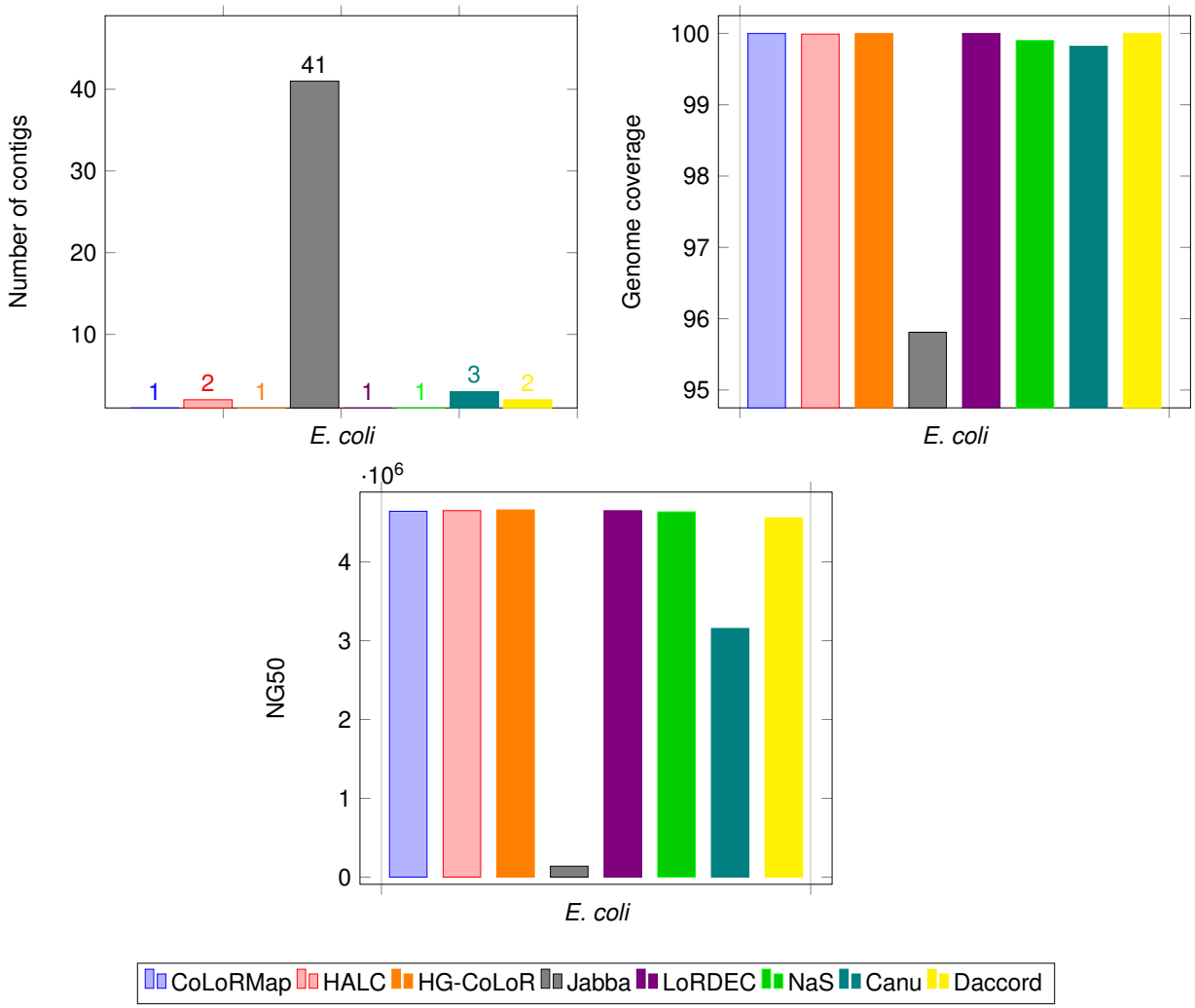
Comparison of the error correction



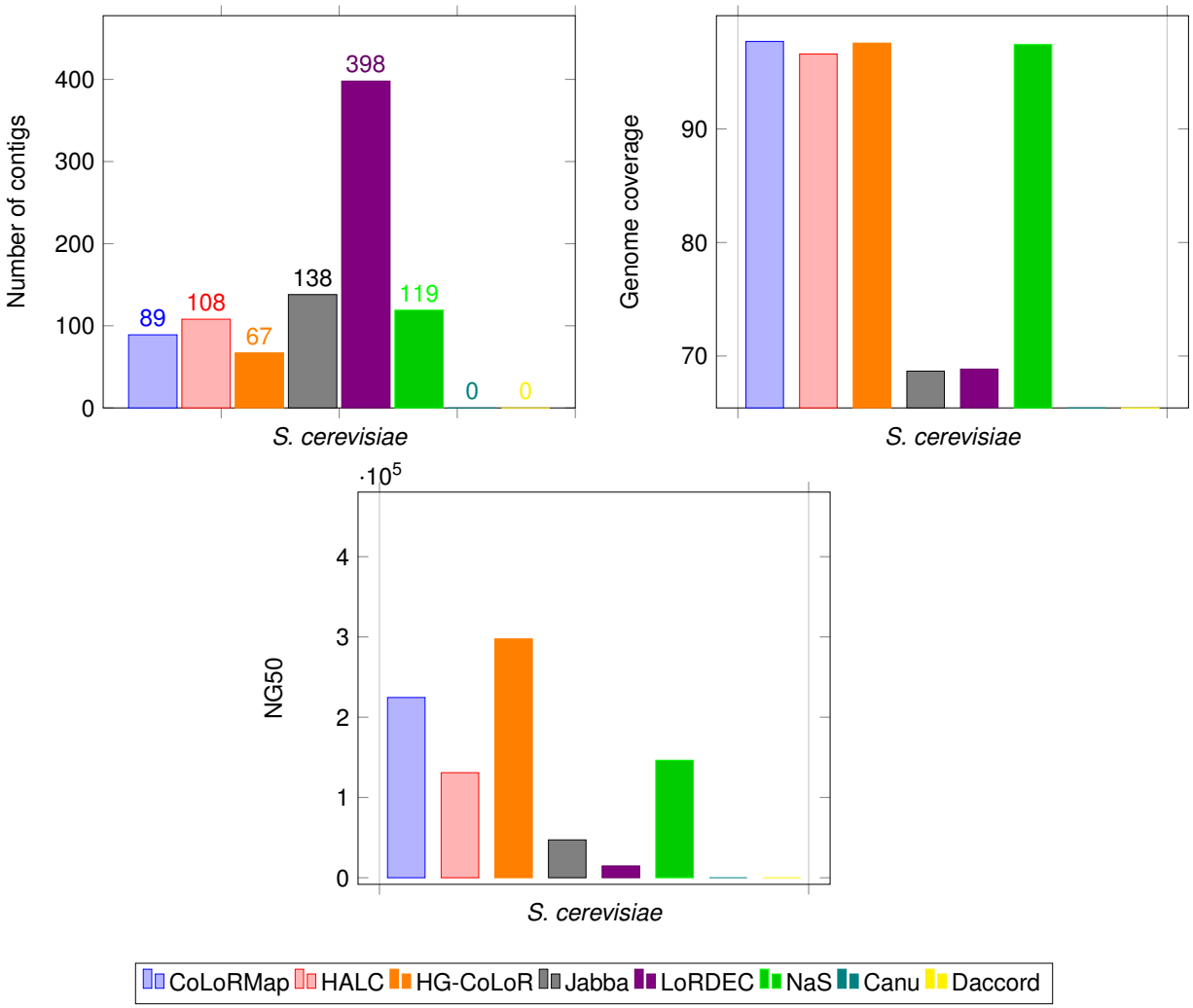
Comparison of the error correction



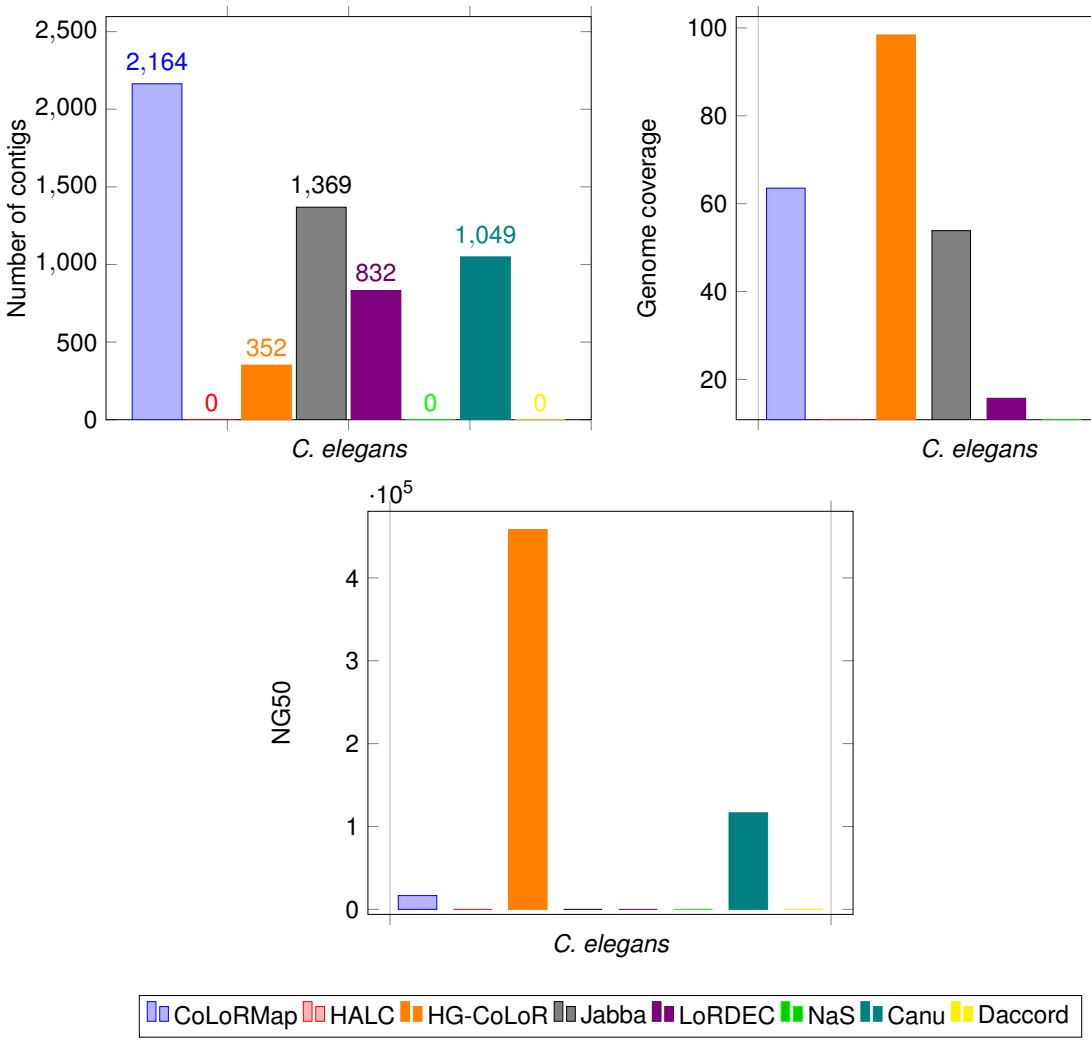
Comparison of the error correction



Comparison of the error correction



Comparison of the error correction



Conclusion

- We introduced a new long read hybrid error correction tool, HG-CoLoR, that:
  - Follows a seed-and-extend approach
  - Exploits the advantages of the variable-order de Bruijn Graph
- Results of our experiments show that HG-CoLoR:
  - Offers a good trade off between runtime and quality, when compared to state-of-the-art methods
  - Efficiently scales to eukaryotic genomes
- Is available from: `https://github.com/morispi/HG-CoLoR`

Conclusion

- We introduced a new long read hybrid error correction tool, HG-CoLoR, that:
  - Follows a seed-and-extend approach
  - Exploits the advantages of the variable-order de Bruijn Graph
- Results of our experiments show that HG-CoLoR:
  - Offers a good trade off between runtime and quality, when compared to state-of-the-art methods
  - Efficiently scales to eukaryotic genomes
- Is available from: `https://github.com/morispi/HG-CoLoR`

Conclusion

- We introduced a new long read hybrid error correction tool, HG-CoLoR, that:
  - Follows a seed-and-extend approach
  - Exploits the advantages of the variable-order de Bruijn Graph
- Results of our experiments show that HG-CoLoR:
  - Offers a good trade off between runtime and quality, when compared to state-of-the-art methods
  - Efficiently scales to eukaryotic genomes
- Is available from: `https://github.com/morispi/HG-CoLoR`

Conclusion

- We introduced a new long read hybrid error correction tool, HG-CoLoR, that:
  - Follows a seed-and-extend approach
  - Exploits the advantages of the variable-order de Bruijn Graph
- Results of our experiments show that HG-CoLoR:
  - Offers a good trade off between runtime and quality, when compared to state-of-the-art methods
  - Efficiently scales to eukaryotic genomes
- Is available from: `https://github.com/morispi/HG-CoLoR`

Conclusion

- We introduced a new long read hybrid error correction tool, HG-CoLoR, that:
  - Follows a seed-and-extend approach
  - Exploits the advantages of the variable-order de Bruijn Graph
- Results of our experiments show that HG-CoLoR:
  - Offers a good trade off between runtime and quality, when compared to state-of-the-art methods
  - Efficiently scales to eukaryotic genomes
- Is available from: `https://github.com/morispi/HG-CoLoR`

Conclusion

- We introduced a new long read hybrid error correction tool, HG-CoLoR, that:
  - Follows a seed-and-extend approach
  - Exploits the advantages of the variable-order de Bruijn Graph
- Results of our experiments show that HG-CoLoR:
  - Offers a good trade off between runtime and quality, when compared to state-of-the-art methods
  - Efficiently scales to eukaryotic genomes
- Is available from: https://github.com/morispi/HG-CoLoR

Conclusion

- We introduced a new long read hybrid error correction tool, HG-CoLoR, that:
  - Follows a seed-and-extend approach
  - Exploits the advantages of the variable-order de Bruijn Graph
- Results of our experiments show that HG-CoLoR:
  - Offers a good trade off between runtime and quality, when compared to state-of-the-art methods
  - Efficiently scales to eukaryotic genomes
- Is available from: `https://github.com/morispi/HG-CoLoR`

Future work

- Try out other aligners to discover seeds
- Focus on the implementation of PgSA

Future work

- Try out other aligners to discover seeds
- Focus on the implementation of PgSA

Thanks for your attention.

Any questions?