Correction and assembly of long reads

Pierre Morisse

16 février 2017

Plan

Idea

- Like NaS, produce synthetic long reads from an assembly of short reads
- Get rid of the alignment of short reads against each other
- Deduce information only from the mapping of short reads on long reads

Workflow

We present the method for the correction of one long read

Workflow

We present the method for the correction of one long read

Principle

Mapping of the short reads and the template long read, and definition of a *lmin* threshold, to retrieve reads that :

Workflow

We present the method for the correction of one long read

Principle

Mapping of the short reads and the template long read, and definition of a *lmin* threshold, to retrieve reads that :

- Are fully aligned, and used as seeds

Workflow

We present the method for the correction of one long read

Principle

Mapping of the short reads and the template long read, and definition of a *lmin* threshold, to retrieve reads that :

- Are fully aligned, and used as seeds
- Have an aligned prefix of length at least *lmin*

Workflow

We present the method for the correction of one long read

Principle

Mapping of the short reads and the template long read, and definition of a *lmin* threshold, to retrieve reads that :

- Are fully aligned, and used as seeds
- Have an aligned prefix of length at least *lmin*
- Have an aligned suffix of length at least *lmin*

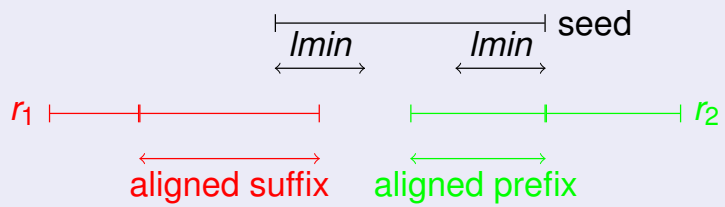
Workflow

Two extension steps :

Workflow

Two extension steps :

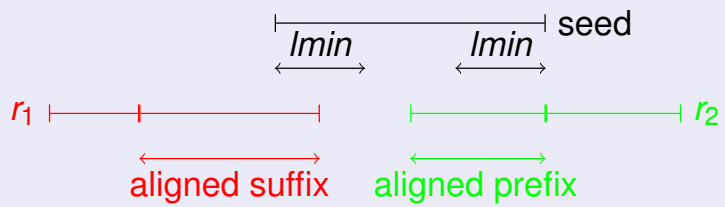
1. Recruitment of partially aligned reads, similar to the seeds



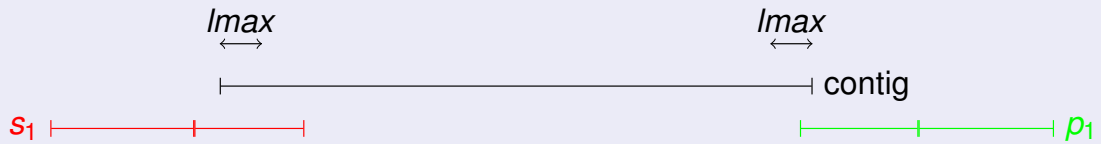
Workflow

Two extension steps :

1. Recruitment of partially aligned reads, similar to the seeds



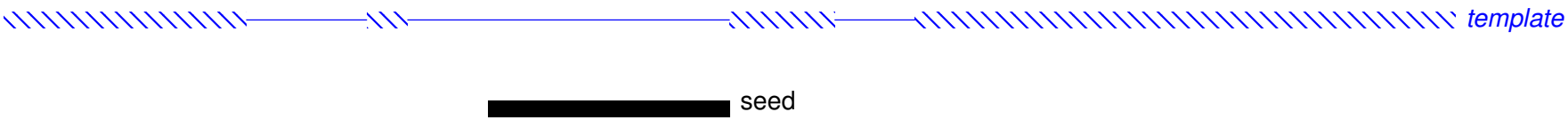
2. Recruiting of new partially aligned reads, without similarity relation, by fixing a new threshold *lmax*



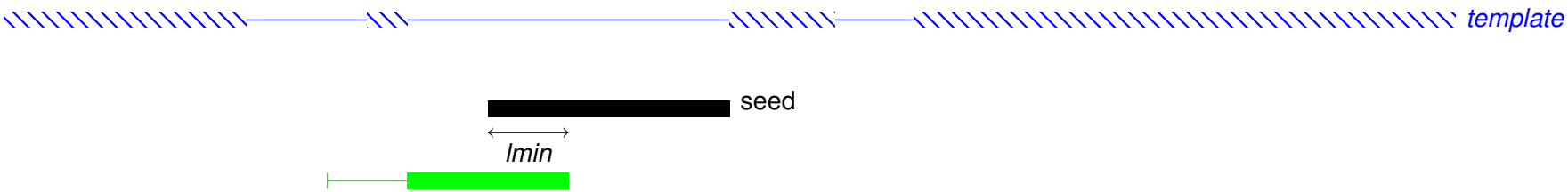
Example

template

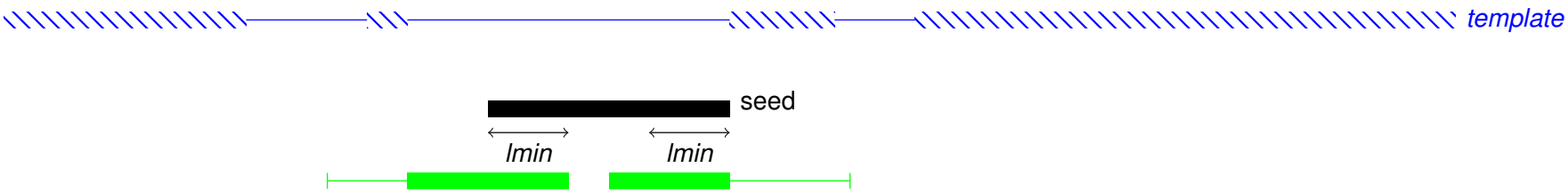
Example



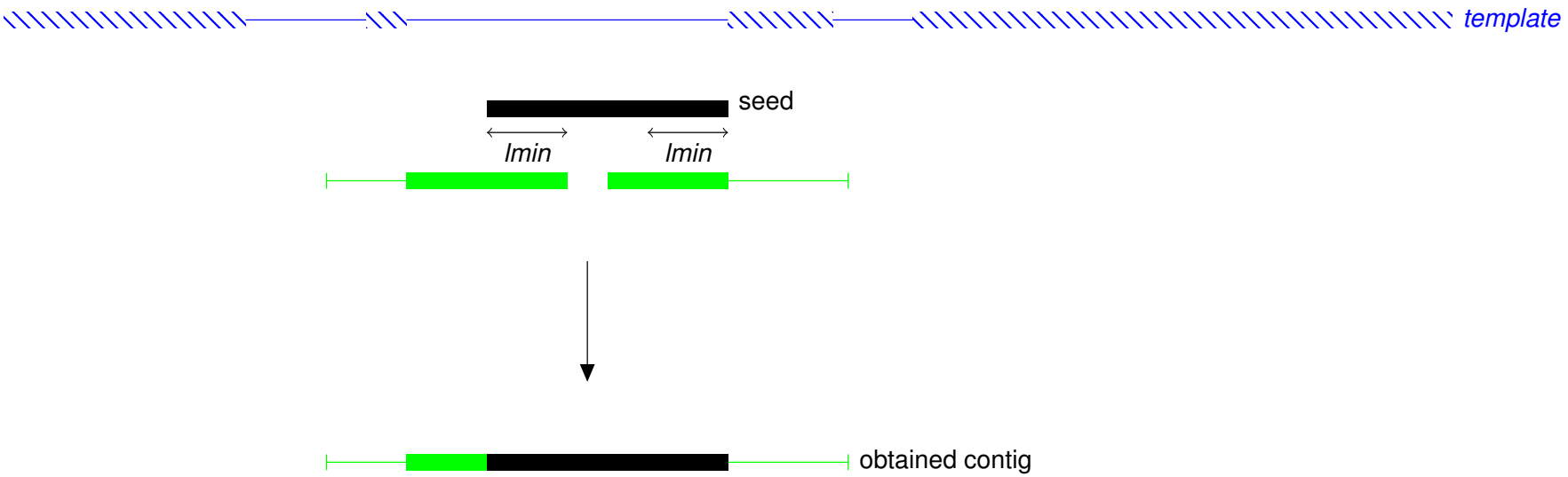
Example



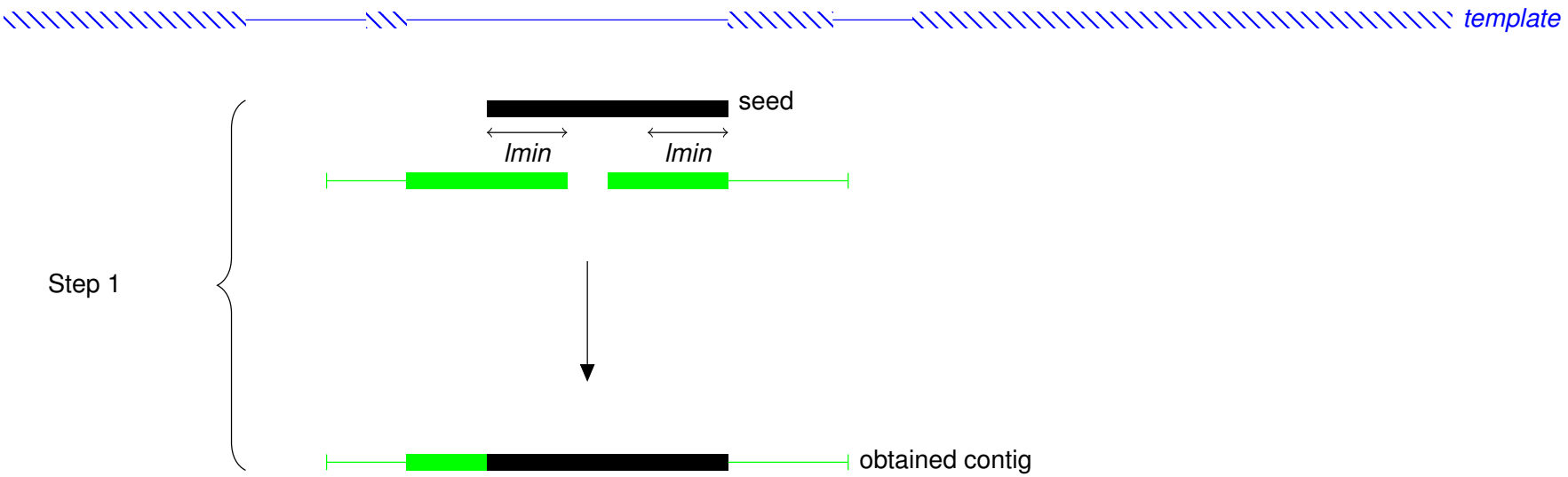
Example



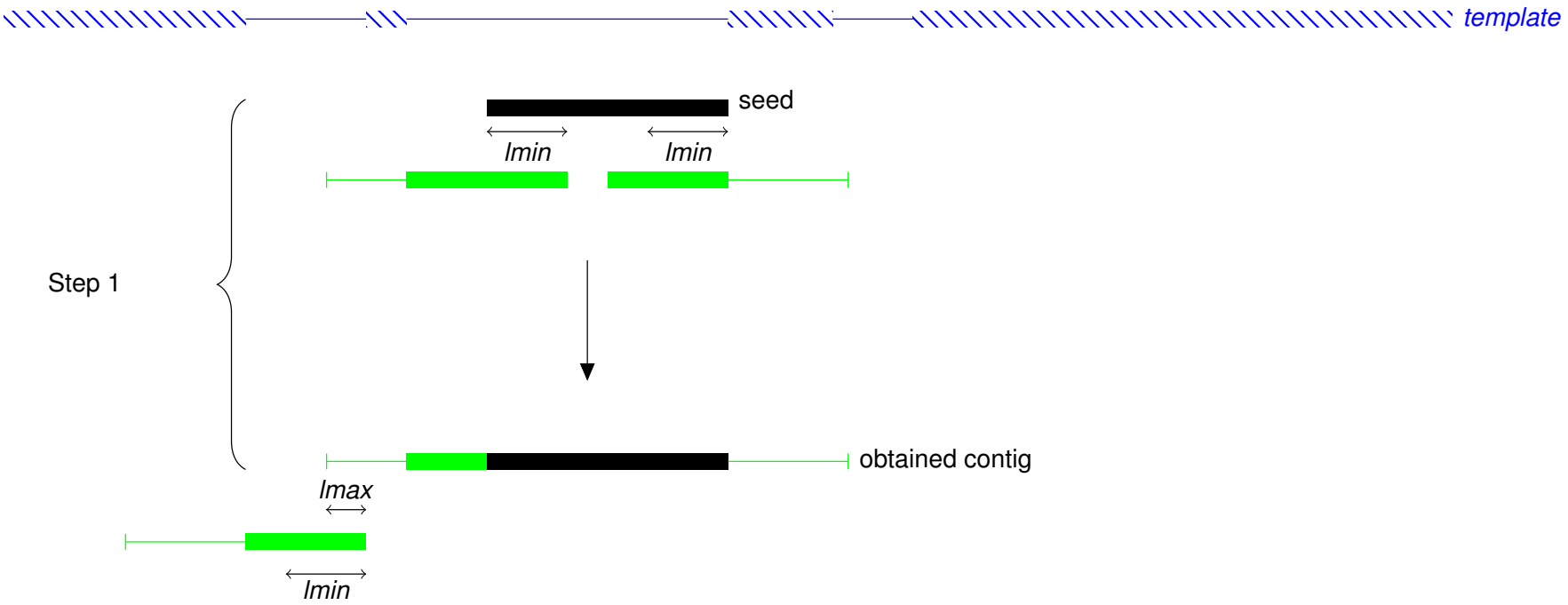
Example



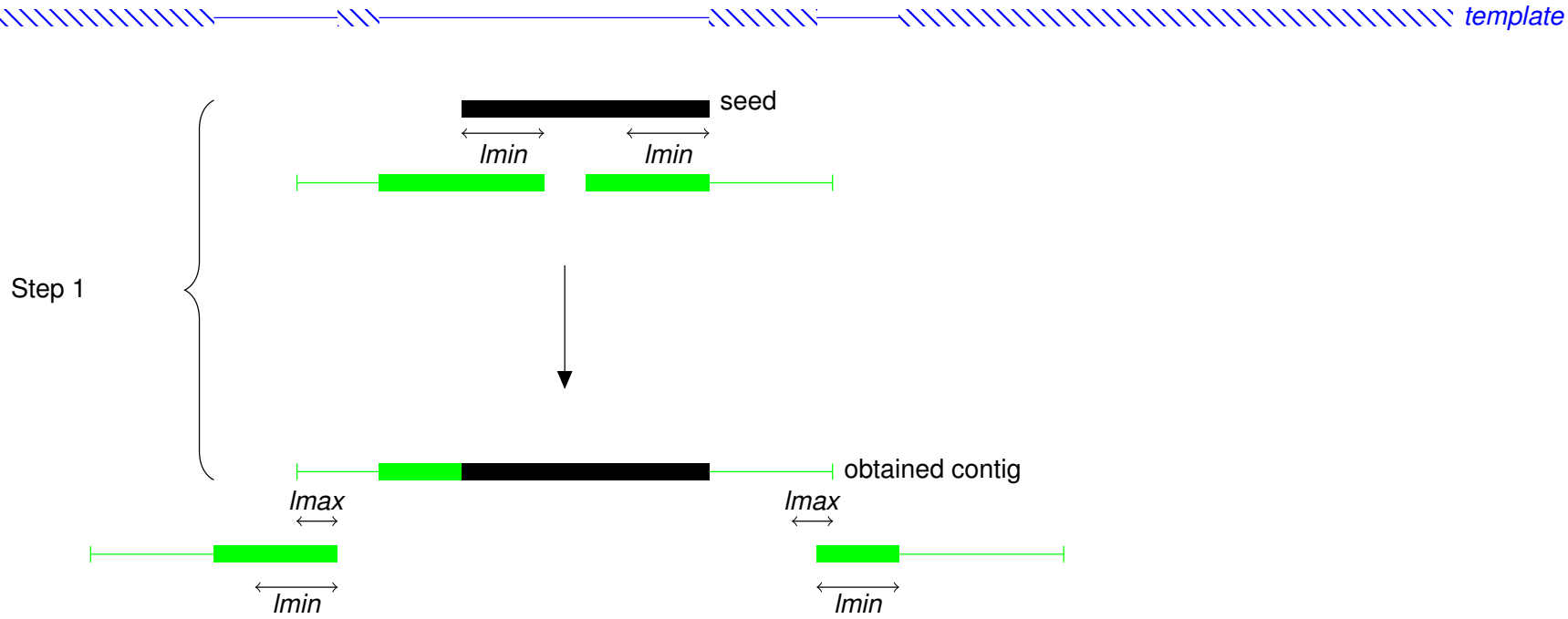
Example



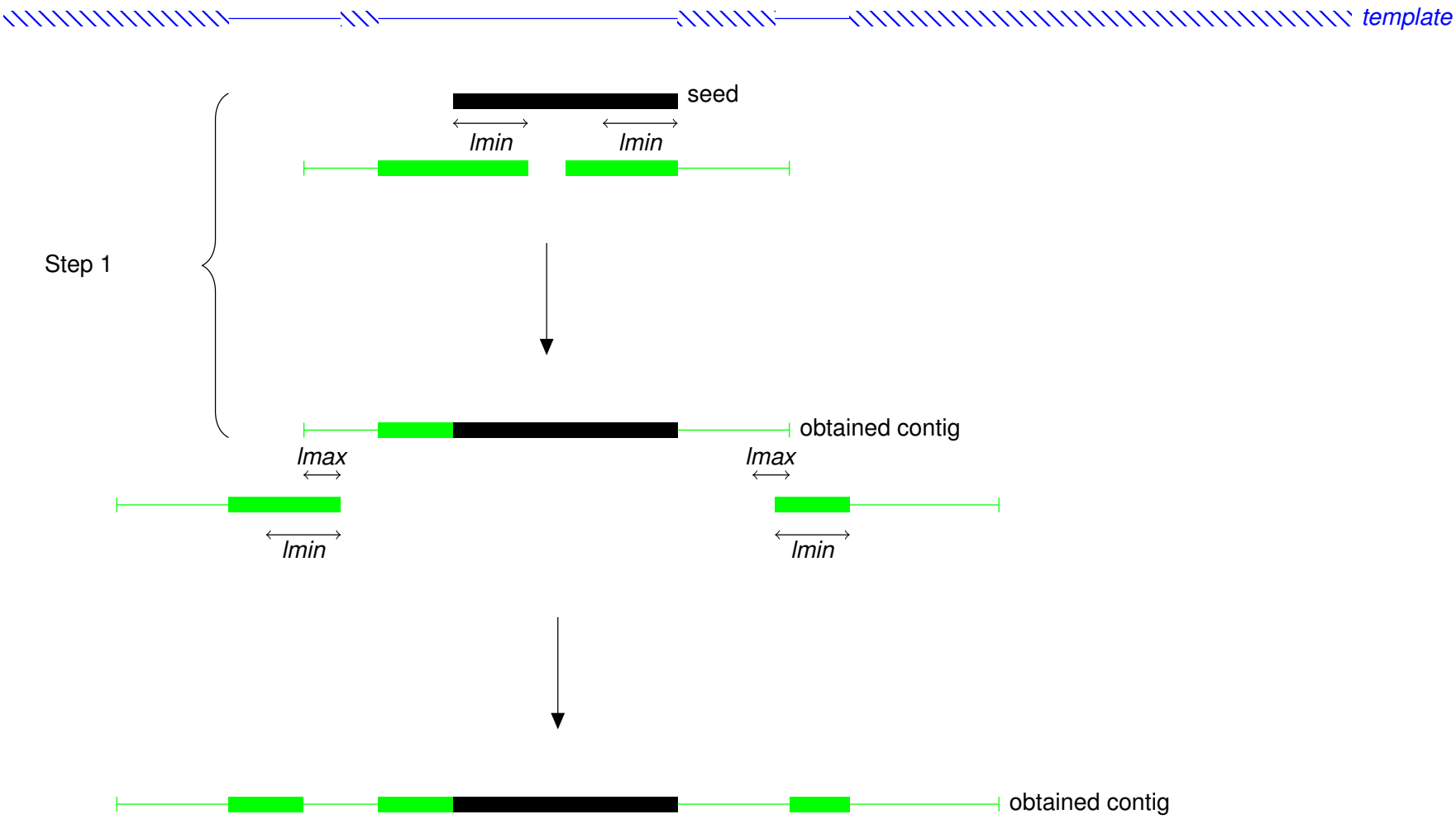
Example



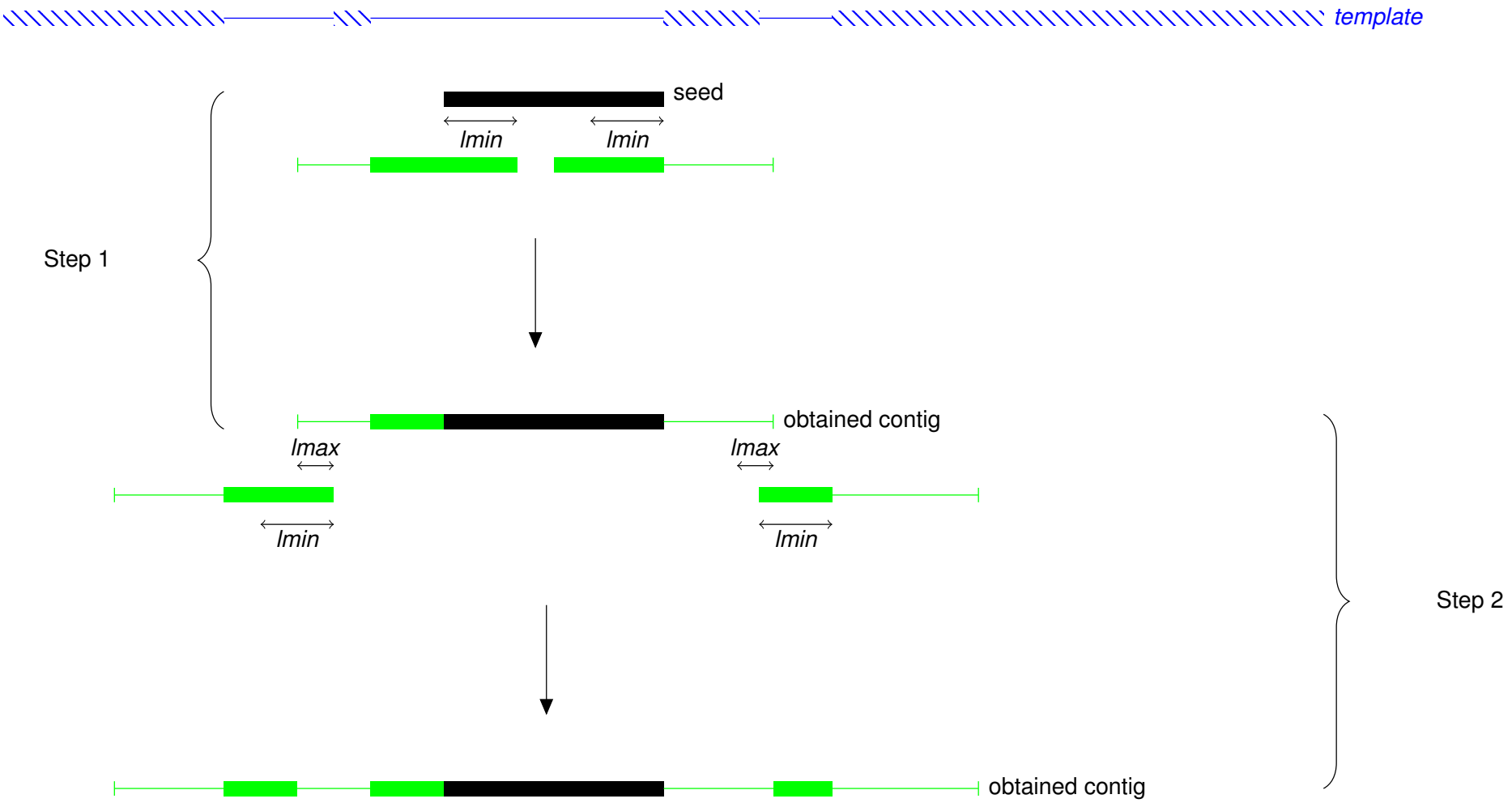
Example


template
seed
lmin
lmin
Step 1
obtained contig
lmax
lmax
lmin
lmin

Example


template
seed
lmin
lmin
Step 1
obtained contig
lmax
lmax
lmin
lmin
obtained contig

Example


template
seed
lmin
lmin
Step 1
obtained contig
lmax
lmax
lmin
lmin
Step 2
obtained contig

Results and conclusion

Results of our method on a long reads dataset from ADP1, with parameters set to $lmin = 100$ and $lmax = 10$:

Results and conclusion

Results of our method on a long reads dataset from ADP1, with parameters set to $lmin = 100$ and $lmax = 10$:

Reads	Average length	Average identity	Number of contigs / read	Average length	Average identity	Covered template	Time
1D	2 052	56,5%	2,296	645	88,636%	72,17%	19min52
2D	10 033	74,5%	2,732	2 421	88,186%	65,93%	14h06min

Results and conclusion

Results of our method on a long reads dataset from ADP1, with parameters set to $lmin = 100$ and $lmax = 10$:

Reads	Average length	Average identity	Number of contigs / read	Average length	Average identity	Covered template	Time
1D	2 052	56,5%	2,296	645	88,636%	72,17%	19min52
2D	10 033	74,5%	2,732	2 421	88,186%	65,93%	14h06min

- Runtime : Less than 10 seconds per long read on average

Results and conclusion

Results of our method on a long reads dataset from ADP1, with parameters set to $lmin = 100$ and $lmax = 10$:

Reads	Average length	Average identity	Number of contigs / read	Average length	Average identity	Covered template	Time
1D	2 052	56,5%	2,296	645	88,636%	72,17%	19min52
2D	10 033	74,5%	2,732	2 421	88,186%	65,93%	14h06min

- Runtime : Less than 10 seconds per long read on average
- Still a high error rate of 12 %

Results and conclusion

Results of our method on a long reads dataset from ADP1, with parameters set to $lmin = 100$ and $lmax = 10$:

Reads	Average length	Average identity	Number of contigs / read	Average length	Average identity	Covered template	Time
1D	2 052	56,5%	2,296	645	88,636%	72,17%	19min52
2D	10 033	74,5%	2,732	2 421	88,186%	65,93%	14h06min

- Runtime : Less than 10 seconds per long read on average
- Still a high error rate of 12 %
- Production of multiple contigs for each long read, and weak coverage

Results and conclusion

Results of our method on a long reads dataset from ADP1, with parameters set to $lmin = 100$ and $lmax = 10$:

Reads	Average length	Average identity	Number of contigs / read	Average length	Average identity	Covered template	Time
1D	2 052	56,5%	2,296	645	88,636%	72,17%	19min52
2D	10 033	74,5%	2,732	2 421	88,186%	65,93%	14h06min

- Runtime : Less than 10 seconds per long read on average
- Still a high error rate of 12 %
- Production of multiple contigs for each long read, and weak coverage
- Tuning the parameters didn't allow better results

Idea

- Obtain contigs from a short reads assembly

Idea

- Obtain contigs from a short reads assembly
- Align long reads on these contigs with BLASR to find local alignments

Idea

- Obtain contigs from a short reads assembly
- Align long reads on these contigs with BLASR to find local alignments
- Link an order contigs with the help of long reads aligned on multiple contigs

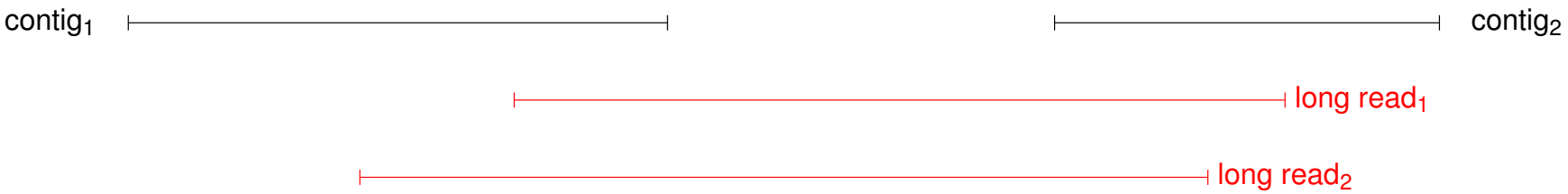
Idea

- Obtain contigs from a short reads assembly
- Align long reads on these contigs with BLASR to find local alignments
- Link an order contigs with the help of long reads aligned on multiple contigs
- Fill the gaps between two linked contigs with a consensus of the bases from the linking long reads

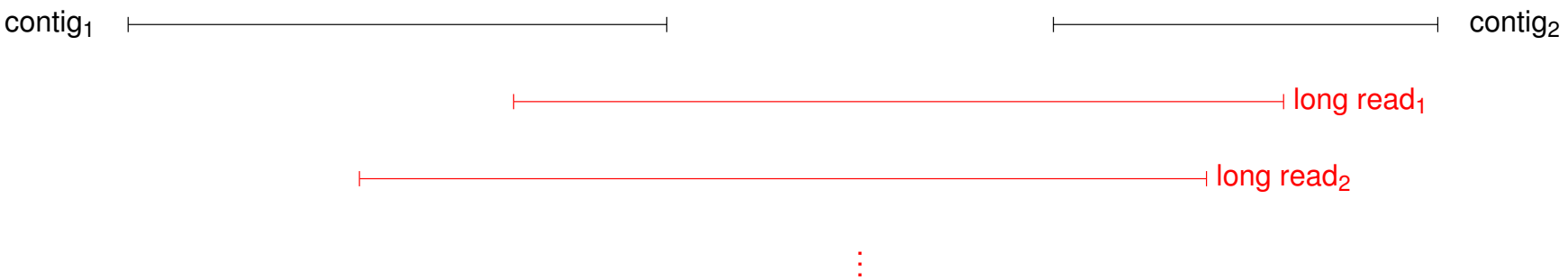
Example

Example

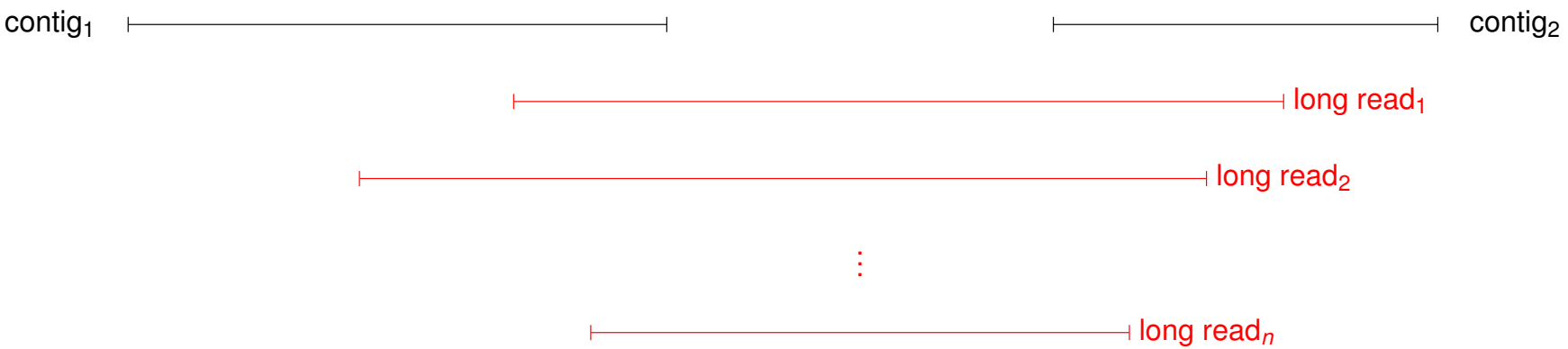
Example



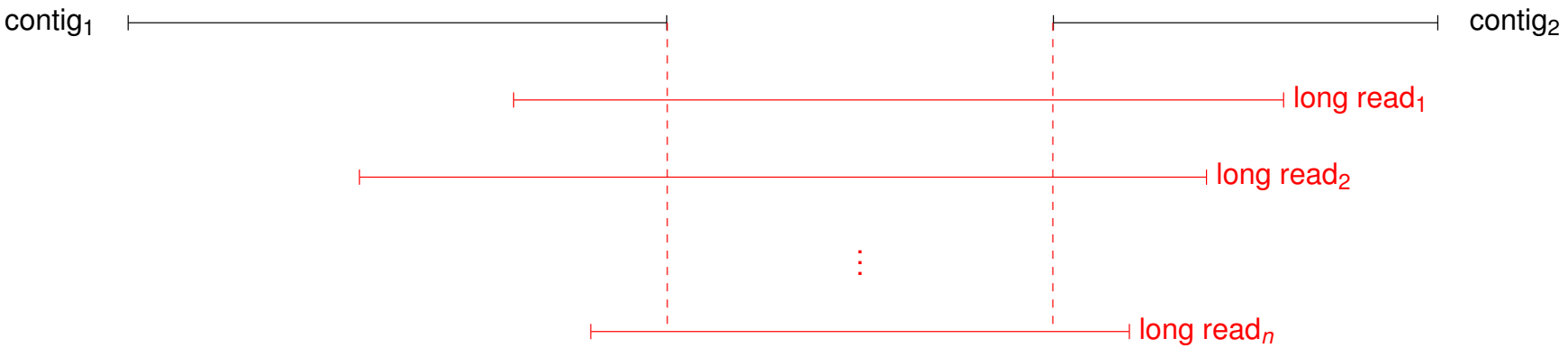
Example



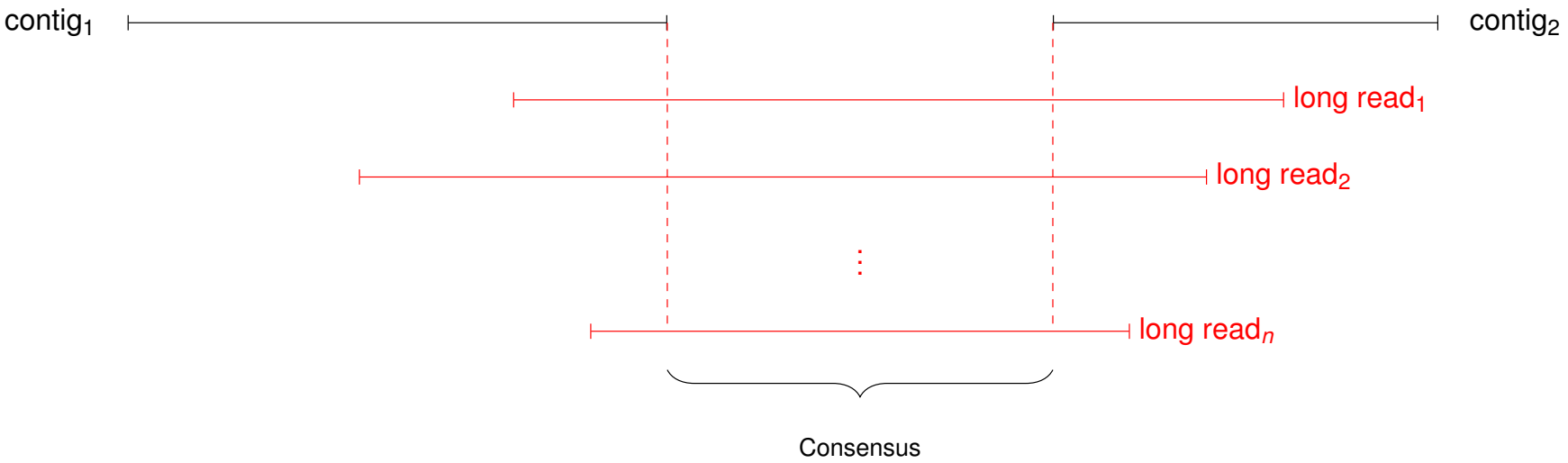
Example



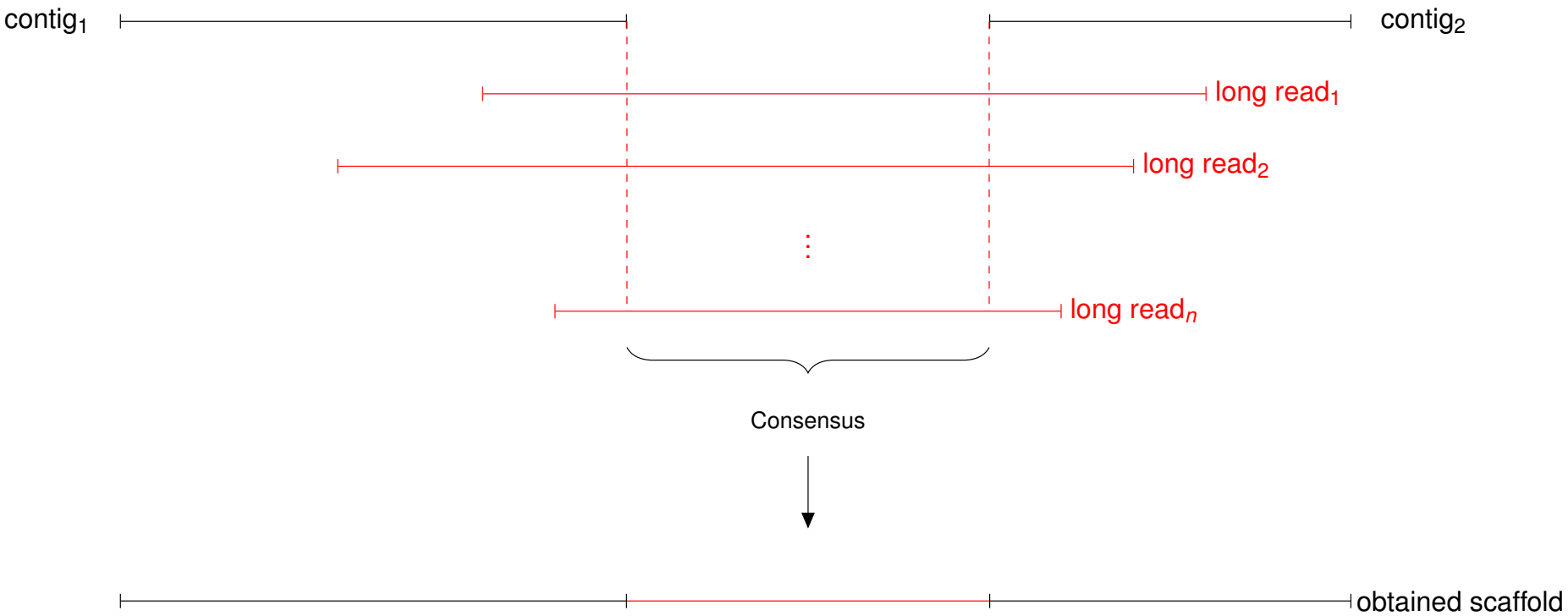
Example



Example


contig$_1$
contig$_2$
long read$_1$
long read$_2$
long read$_n$
Consensus

Example


contig$_1$
contig$_2$
long read$_1$
long read$_2$
long read$_n$
Consensus
obtained scaffold

Observations and conclusion

- Already existing method : SSPACE-LongRead already allows scaffolding with long reads, but fills the gaps between the contigs with Ns instead of building a consensus of the long reads bases

Observations and conclusion

- Already existing method : SSPACE-LongRead already allows scaffolding with long reads, but fills the gaps between the contigs with Ns instead of building a consensus of the long reads bases
- PBJelly also allows scaffolding with long reads, and fills the gaps with bases from the long reads

Observations and conclusion

- Already existing method : SSPACE-LongRead already allows scaffolding with long reads, but fills the gaps between the contigs with Ns instead of building a consensus of the long reads bases
- PBJelly also allows scaffolding with long reads, and fills the gaps with bases from the long reads
- Useless to develop this idea any more

Idea

- Get k-mers appearing in the long reads, but not in the short reads contigs

Idea

- Get k-mers appearing in the long reads, but not in the short reads contigs
- Assemble the obtained k-mers $\Rightarrow$ Obtain new contigs

Idea

- Get k-mers appearing in the long reads, but not in the short reads contigs

- Assemble the obtained k-mers $\Rightarrow$ Obtain new contigs
- Use these new contigs to cover regions from the reference genome that were previously uncovered by the short reads contigs

Idea

- Get k-mers appearing in the long reads, but not in the short reads contigs
- Assemble the obtained k-mers $\Rightarrow$ Obtain new contigs
- Use these new contigs to cover regions from the reference genome that were previously uncovered by the short reads contigs
- Try to allow a less fragmented assembly by assembling together short reads and long reads contigs

Tests and results

With a set of reads from ADP1 :

k-mer size	Number of k-mers	Number of k-mer in the short reads contigs
64	375 999 371	380 967
32	335 836 269	2 942 135
16	322 493 916	6 197 488
8	65 376	65 309

Tests and results

With a set of reads from ADP1 :

k-mer size	Number of *k*-mers	Number of *k*-mer in the short reads contigs
64	375 999 371	380 967
32	335 836 269	2 942 135
16	322 493 916	6 197 488
8	65 376	65 309

Only 8-mers seem to be present enough in the short reads contigs, but too short to bring any interesting information

Tests and results

With a set of reads from ADP1 :

k-mer size	Number of k-mers	Number of k-mer in the short reads contigs
64	375 999 371	380 967
32	335 836 269	2 942 135
16	322 493 916	6 197 488
8	65 376	65 309

Only 8-mers seem to be present enough in the short reads contigs, but too short to bring any interesting information

$\Rightarrow$ More interesting to search for spaced k-mers ?

spaced *k*-mers

Definition

Here, we define a spaced *k*-mer as a *k*-mer in which we allow a gap of a certain length

spaced *k*-mers

Definition

Here, we define a spaced *k*-mer as a *k*-mer in which we allow a gap of a certain length

Example

For instance, the 8-mer GATCTTAC, if we allow a gap of length 2, becomes the following spaced "8"-mer : GATC**TTAC, where * denote joker positions (match or mismatch allowed)

spaced *k*-mers

Definition

Here, we define a spaced *k*-mer as a *k*-mer in which we allow a gap of a certain length

Example

For instance, the 8-mer GATCTTAC, if we allow a gap of length 2, becomes the following spaced "8"-mer : GATC**TTAC, where * denote joker positions (match or mismatch allowed)

We use this definition of spaced *k*-mers rather than the classical one in order to take into account indel. error instead of substitutions errors, as they are more common in long reads

Tests and results

Results on the previous dataset, allowing a gap of length at most 10 :

k-mer size	Number of k-mers	Number of k-mer in the short reads contigs
64	375 999 371	425 155
32	335 836 269	3 859 742
16	322 493 916	16 036 610

Tests and results

Results on the previous dataset, allowing a gap of length at most 10 :

k-mer size	Number of k-mers	Number of k-mer in the short reads contigs
64	375 999 371	425 155
32	335 836 269	3 859 742
16	322 493 916	16 036 610

$\Rightarrow$ Comparison is still not interesting, even with spaced k-mers

Observations

- Trimming the ends of the longs reads doesn't allow any better results $\Rightarrow$ Unlike short reads, errors aren't mostly located at the ends of the reads, but occur everywhere

Observations

- Trimming the ends of the longs reads doesn't allow any better results $\Rightarrow$ Unlike short reads, errors aren't mostly located at the ends of the reads, but occur everywhere

- Comparing long reads k-mer directly with short reads k-mers doesn't allow any better results

Observations

- Trimming the ends of the longs reads doesn't allow any better results $\Rightarrow$ Unlike short reads, errors aren't mostly located at the ends of the reads, but occur everywhere
- Comparing long reads k-mer directly with short reads k-mers doesn't allow any better results
- Not interesting approach

Observations

- Trimming the ends of the longs reads doesn't allow any better results $\Rightarrow$ Unlike short reads, errors aren't mostly located at the ends of the reads, but occur everywhere
- Comparing long reads k-mer directly with short reads k-mers doesn't allow any better results
- Not interesting approach
- Allow more gaps in the spaced k-mers, and not just one in the middle, could potentially yield better resultats $\Rightarrow$ Need to find an algorithm that builds a spaced suffix arrays

Principle

5 étapes :

Principle

5 étapes :

1. Short reads correction (with Quorum)

Principle

5 étapes :

1. Short reads correction (with Quorum)
2. Mapping of the short reads on the long reads, in order to find seeds (with BLASR)

Principle

5 étapes :

1. Short reads correction (with Quorum)
2. Mapping of the short reads on the long reads, in order to find seeds (with BLASR)
3. Merging of the seeds that overlap over more than a certain length

Principle

5 étapes :

1. Short reads correction (with Quorum)
2. Mapping of the short reads on the long reads, in order to find seeds (with BLASR)
3. Merging of the seeds that overlap over more than a certain length
4. Linking of the seeds, by extending them with perfect overlaps with k-mers from the corrected short reads

Principle

5 étapes :

1. Short reads correction (with Quorum)
2. Mapping of the short reads on the long reads, in order to find seeds (with BLASR)
3. Merging of the seeds that overlap over more than a certain length
4. Linking of the seeds, by extending them with perfect overlaps with k-mers from the corrected short reads
5. Extension of the obtained synthetic long read, on the left (resp. right) of the leftmost (resp. rightmost) seed

Used tool : PgSA

PgSA (Pseudogenome Suffix Array) allows the indexing of a set of reads, in order to answer the 7 following queries, for a given string f :

Used tool : PgSA

PgSA (Pseudogenome Suffix Array) allows the indexing of a set of reads, in order to answer the 7 following queries, for a given string f :

1. In which reads does f occur ?

Used tool : PgSA

PgSA (Pseudogenome Suffix Array) allows the indexing of a set of reads, in order to answer the 7 following queries, for a given string f :

1. In which reads does f occur ?
2. In how many reads does f occur ?

Used tool : PgSA

PgSA (Pseudogenome Suffix Array) allows the indexing of a set of reads, in order to answer the 7 following queries, for a given string f :

1. In which reads does f occur ?
2. In how many reads does f occur ?
3. What are the occurrence positions of f ?

Used tool : PgSA

PgSA (Pseudogenome Suffix Array) allows the indexing of a set of reads, in order to answer the 7 following queries, for a given string f :

1. In which reads does f occur ?
2. In how many reads does f occur ?
3. What are the occurrence positions of f ?
4. What is the number of occurrences of f ?

Used tool : PgSA

PgSA (Pseudogenome Suffix Array) allows the indexing of a set of reads, in order to answer the 7 following queries, for a given string f :

1. In which reads does f occur ?
2. In how many reads does f occur ?
3. What are the occurrence positions of f ?
4. What is the number of occurrences of f ?
5. In which reads does f occur only once ?

Used tool : PgSA

PgSA (Pseudogenome Suffix Array) allows the indexing of a set of reads, in order to answer the 7 following queries, for a given string f :

1. In which reads does f occur ?
2. In how many reads does f occur ?
3. What are the occurrence 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 ?

Used tool : PgSA

PgSA (Pseudogenome Suffix Array) allows the indexing of a set of reads, in order to answer the 7 following queries, for a given string f :

1. In which reads does f occur ?
2. In how many reads does f occur ?
3. What are the occurrence 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 occurrence positions of f in the reads where it occurs only once ?

Used tool : PgSA

PgSA (Pseudogenome Suffix Array) allows the indexing of a set of reads, in order to answer the 7 following queries, for a given string f :

1. In which reads does f occur ?
2. In how many reads does f occur ?
3. What are the occurrence 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 occurrence positions of f in the reads where it occurs only once ?

Among these requests, the third one will allow us to find perfect overlaps between k-mers

Used tool : PgSA

Other data structures (Gk-Arrays, Compressed Gk-Arrays) are able to handle these requests, but the length k of the string f has to bet set at compilation time, whereas PgSA allows the handling of the requests for arbitrary values of k

Used tool : PgSA

Other data structures (Gk-Arrays, Compressed Gk-Arrays) are able to handle these requests, but the length k of the string f has to bet set at compilation time, whereas PgSA allows the handling of the requests for arbitrary values of k

$\Rightarrow$ Allow to look for overlaps of length $k-2$ if no overlap of length $k-1$ was found, without any need to recompute the whole index

Step 4

The *k*-mers from the set of correct short reads (and their reverse-complements) are indexed with PgSA, and third request is looped over to find perfect overlap between *k*-mers, allowing to link together the seeds

Step 4

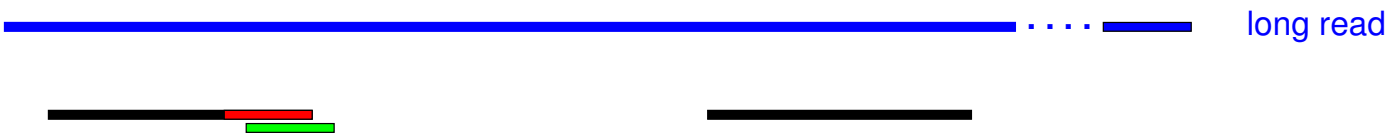
The *k*-mers from the set of correct short reads (and their reverse-complements) are indexed with PgSA, and third request is looped over to find perfect overlap between *k*-mers, allowing to link together the seeds

Step 4

The *k*-mers from the set of correct short reads (and their reverse-complements) are indexed with PgSA, and third request is looped over to find perfect overlap between *k*-mers, allowing to link together the seeds

Step 4

The *k*-mers from the set of correct short reads (and their reverse-complements) are indexed with PgSA, and third request is looped over to find perfect overlap between *k*-mers, allowing to link together the seeds



Step 4

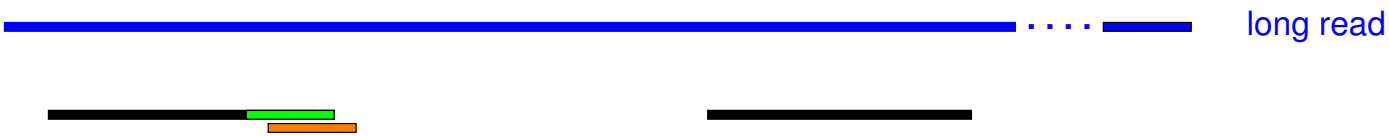
The *k*-mers from the set of correct short reads (and their reverse-complements) are indexed with PgSA, and third request is looped over to find perfect overlap between *k*-mers, allowing to link together the seeds

Step 4

The *k*-mers from the set of correct short reads (and their reverse-complements) are indexed with PgSA, and third request is looped over to find perfect overlap between *k*-mers, allowing to link together the seeds

Step 4

The *k*-mers from the set of correct short reads (and their reverse-complements) are indexed with PgSA, and third request is looped over to find perfect overlap between *k*-mers, allowing to link together the seeds



Step 4

The *k*-mers from the set of correct short reads (and their reverse-complements) are indexed with PgSA, and third request is looped over to find perfect overlap between *k*-mers, allowing to link together the seeds

Step 4

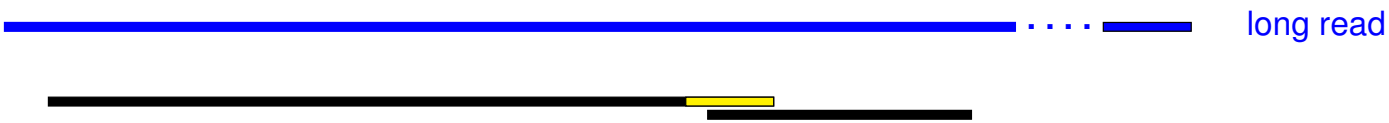
The *k*-mers from the set of correct short reads (and their reverse-complements) are indexed with PgSA, and third request is looped over to find perfect overlap between *k*-mers, allowing to link together the seeds

Step 4

The *k*-mers from the set of correct short reads (and their reverse-complements) are indexed with PgSA, and third request is looped over to find perfect overlap between *k*-mers, allowing to link together the seeds

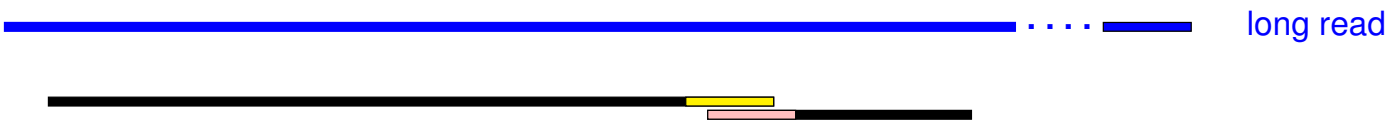
Step 4

The *k*-mers from the set of correct short reads (and their reverse-complements) are indexed with PgSA, and third request is looped over to find perfect overlap between *k*-mers, allowing to link together the seeds



Step 4

The *k*-mers from the set of correct short reads (and their reverse-complements) are indexed with PgSA, and third request is looped over to find perfect overlap between *k*-mers, allowing to link together the seeds



Step 4

The *k*-mers from the set of correct short reads (and their reverse-complements) are indexed with PgSA, and third request is looped over to find perfect overlap between *k*-mers, allowing to link together the seeds

Observations

- It possible that a given k-mer perfectly overlaps multiple other k-mers $\Rightarrow$ Exploration of every possible extension with backtracking

Observations

- It possible that a given k-mer perfectly overlaps multiple other k-mers $\Rightarrow$ Exploration of every possible extension with backtracking

- Some seeds might be impossible to link together $\Rightarrow$ Production of a synthetic long read fragmented in multiple parts

Observations

- It possible that a given k-mer perfectly overlaps multiple other k-mers $\Rightarrow$ Exploration of every possible extension with backtracking

- Some seeds might be impossible to link together $\Rightarrow$ Production of a synthetic long read fragmented in multiple parts

Observations

- It possible that a given k-mer perfectly overlaps multiple other k-mers $\Rightarrow$ Exploration of every possible extension with backtracking

- Some seeds might be impossible to link together $\Rightarrow$ Production of a synthetic long read fragmented in multiple parts

Observations

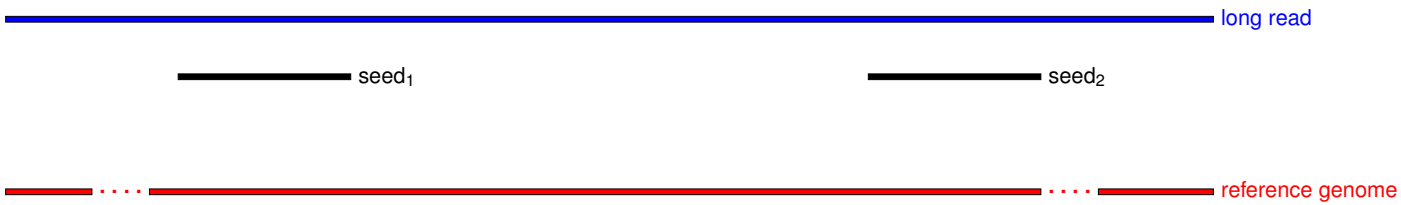
- It possible that a given k-mer perfectly overlaps multiple other k-mers $\Rightarrow$ Exploration of every possible extension with backtracking

- Some seeds might be impossible to link together $\Rightarrow$ Production of a synthetic long read fragmented in multiple parts

Observations

- It possible that a given k-mer perfectly overlaps multiple other k-mers $\Rightarrow$ Exploration of every possible extension with backtracking

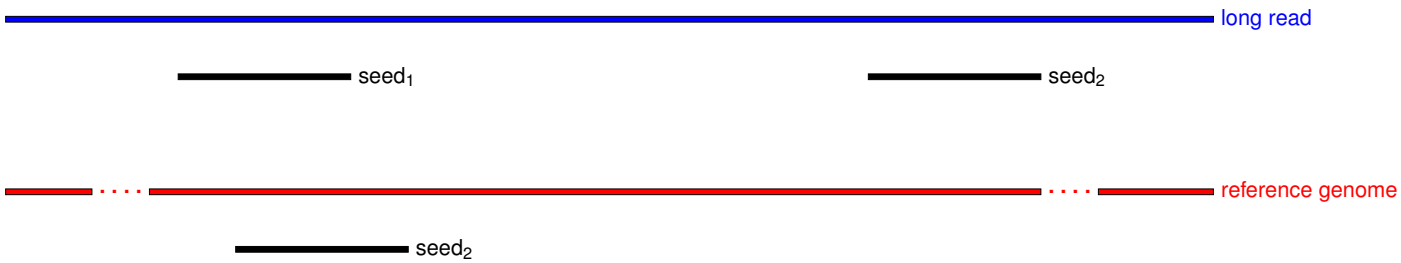
- Some seeds might be impossible to link together $\Rightarrow$ Production of a synthetic long read fragmented in multiple parts



Observations

- It possible that a given k-mer perfectly overlaps multiple other k-mers $\Rightarrow$ Exploration of every possible extension with backtracking

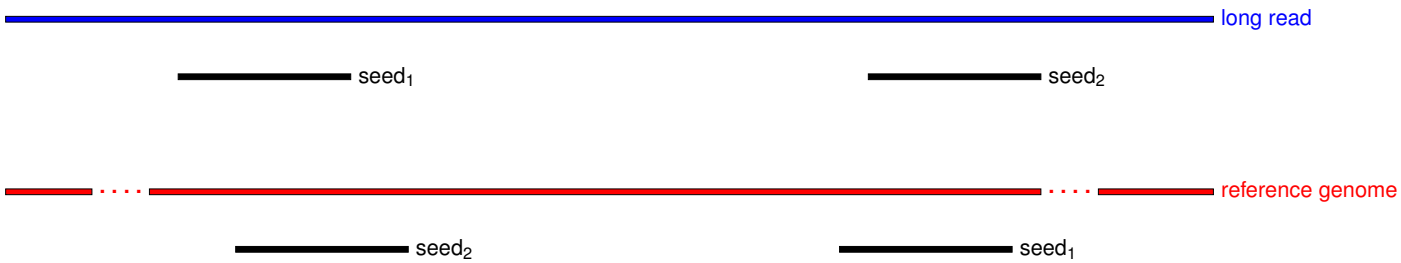
- Some seeds might be impossible to link together $\Rightarrow$ Production of a synthetic long read fragmented in multiple parts



Observations

- It possible that a given k-mer perfectly overlaps multiple other k-mers $\Rightarrow$ Exploration of every possible extension with backtracking

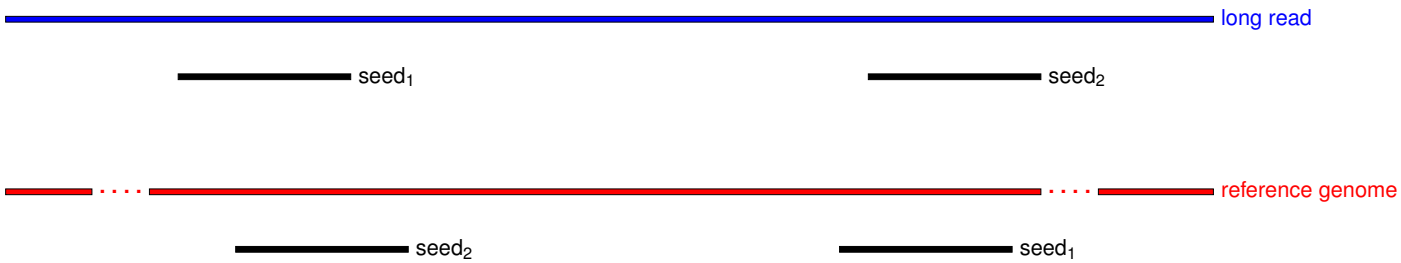
- Some seeds might be impossible to link together $\Rightarrow$ Production of a synthetic long read fragmented in multiple parts



Observations

- It possible that a given k-mer perfectly overlaps multiple other k-mers $\Rightarrow$ Exploration of every possible extension with backtracking

- Some seeds might be impossible to link together $\Rightarrow$ Production of a synthetic long read fragmented in multiple parts



- When only one seed mapped on a given long read, we just extend this seed on both directions

Step 5

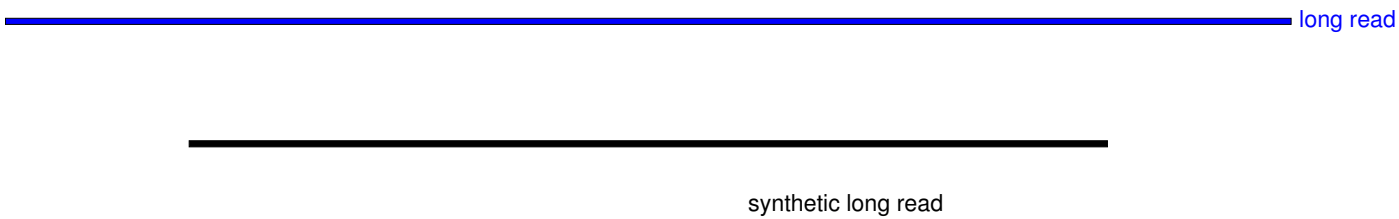
The leftmost seed doesn't always map right at the beginning of the long read, and the rightmost doesn't always reach the end

Step 5

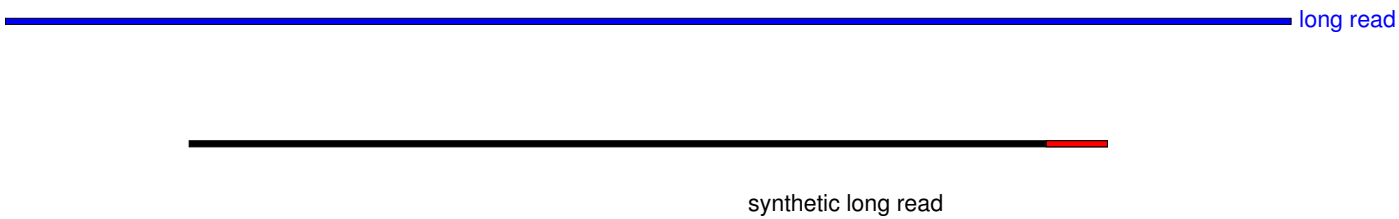
The leftmost seed doesn't always map right at the beginning of the long read, and the rightmost doesn't always reach the end

$\Rightarrow$ Once all seeds have been linked, and the synthetic long read produced, we extend, with the help of perfectly overlapping k-mers, its both ends, until the borders of the initial long reads are reached, or an ambiguity is met (multiple k-mers perfectly overlapping the currently considered one)

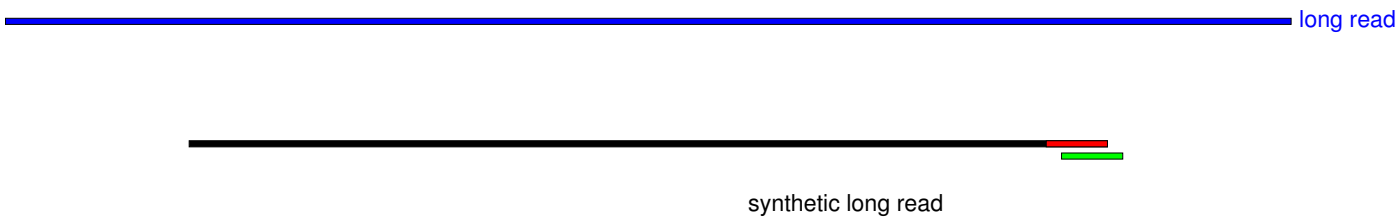
Step 5, first case : No ambiguity



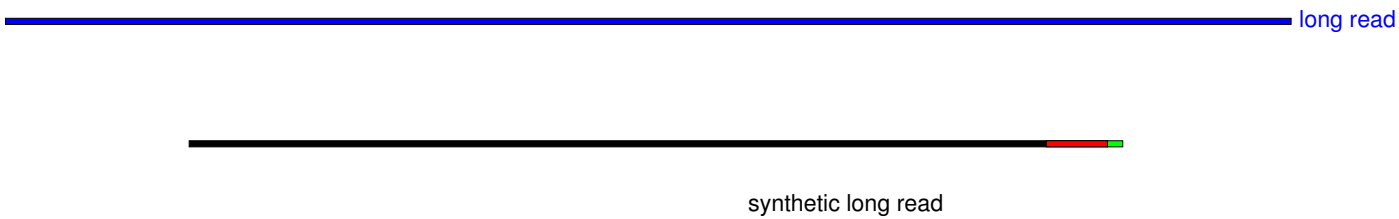
Step 5, first case : No ambiguity



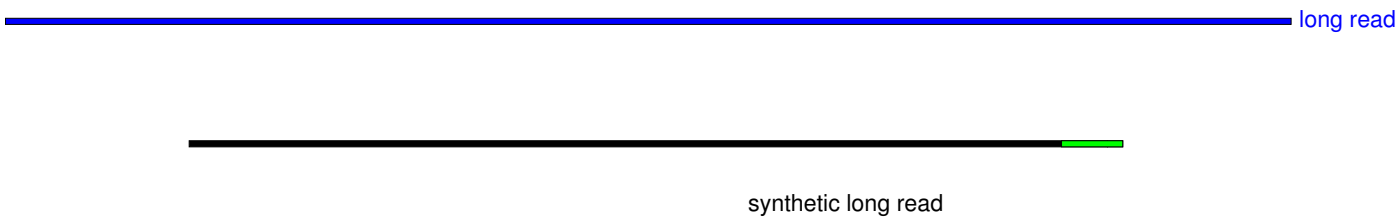
Step 5, first case : No ambiguity



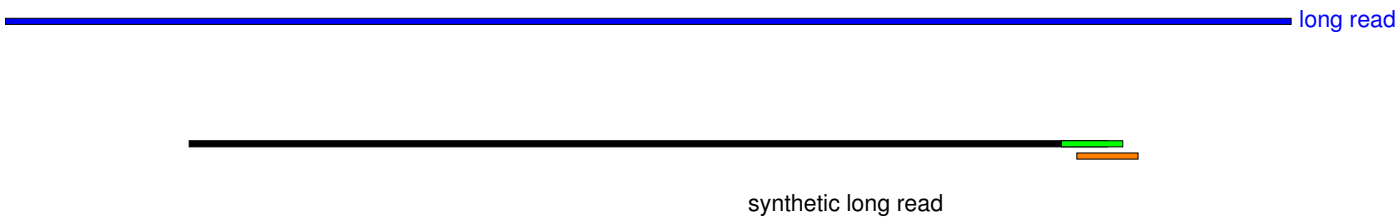
Step 5, first case : No ambiguity



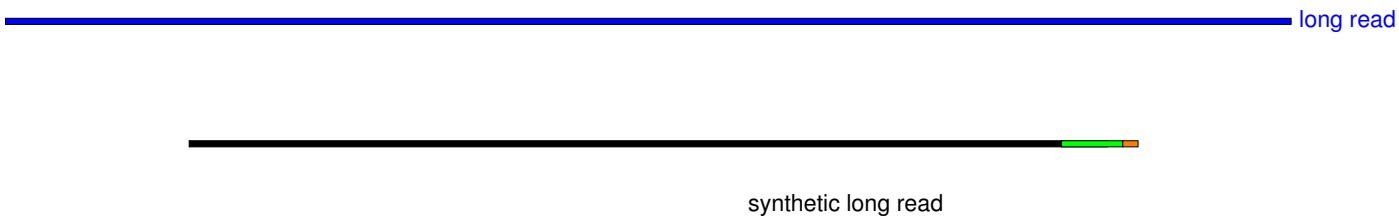
Step 5, first case : No ambiguity



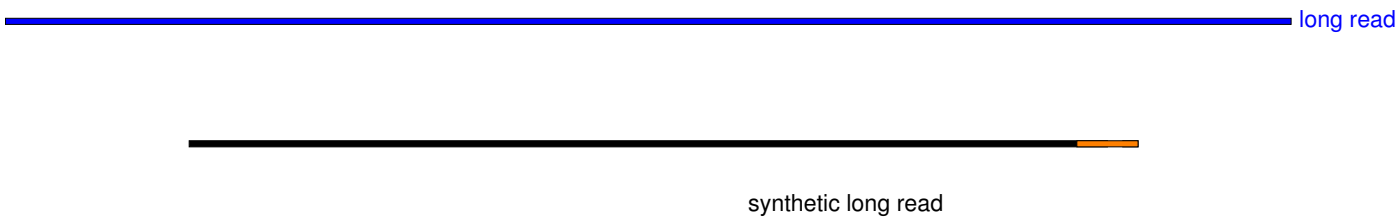
Step 5, first case : No ambiguity



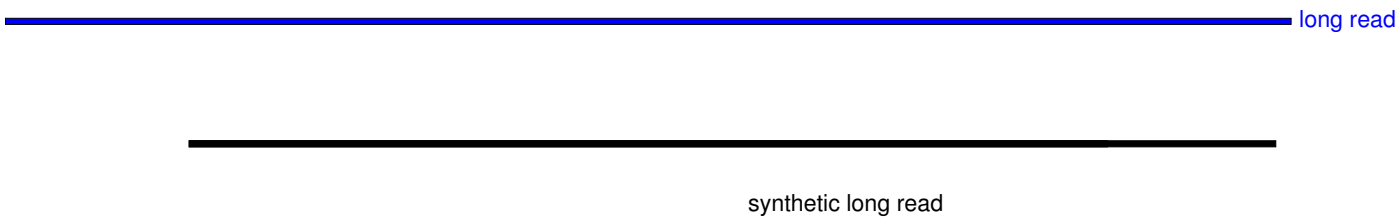
Step 5, first case : No ambiguity



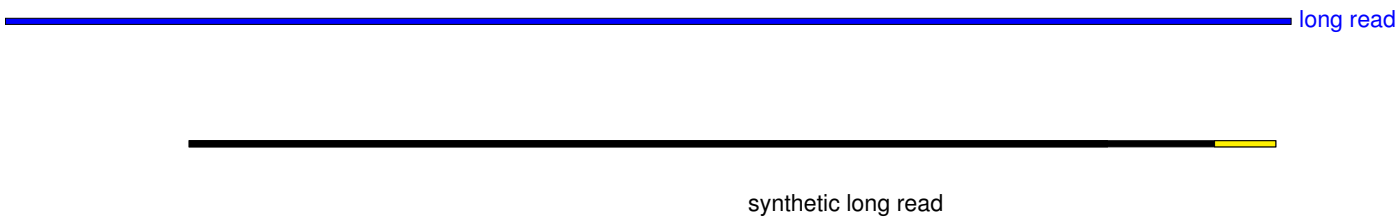
Step 5, first case : No ambiguity



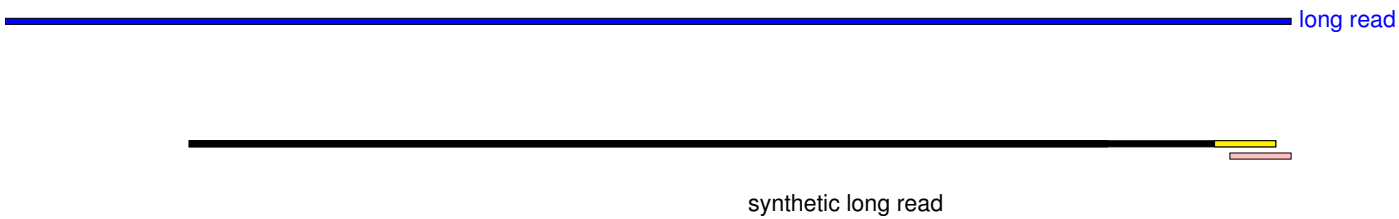
Step 5, first case : No ambiguity



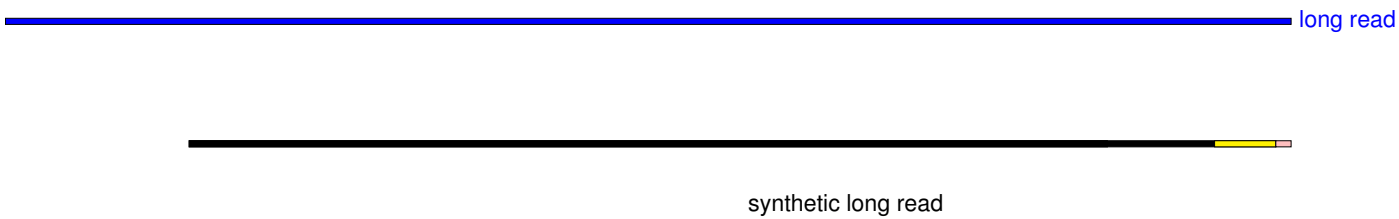
Step 5, first case : No ambiguity



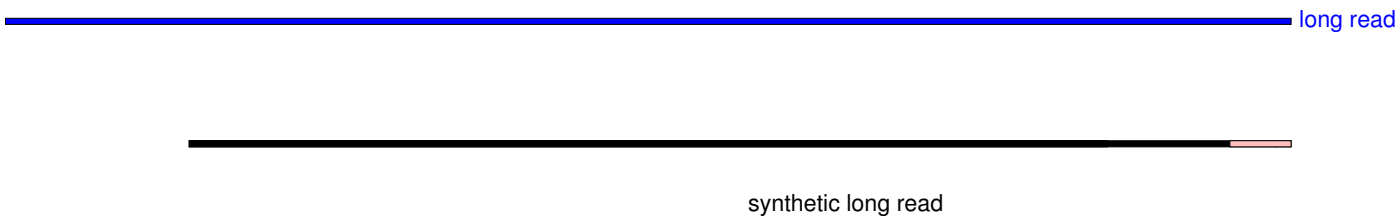
Step 5, first case : No ambiguity



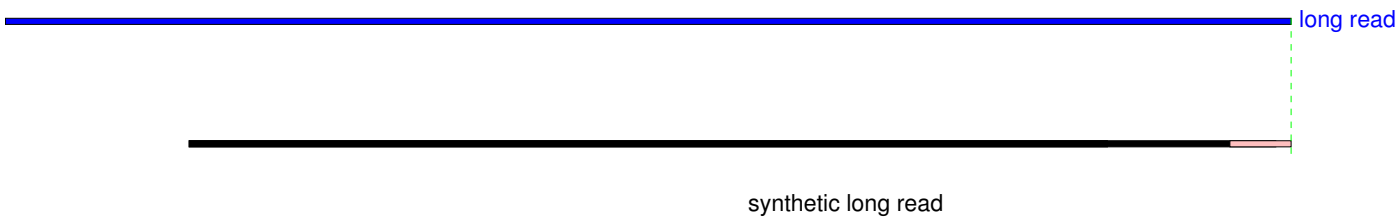
Step 5, first case : No ambiguity



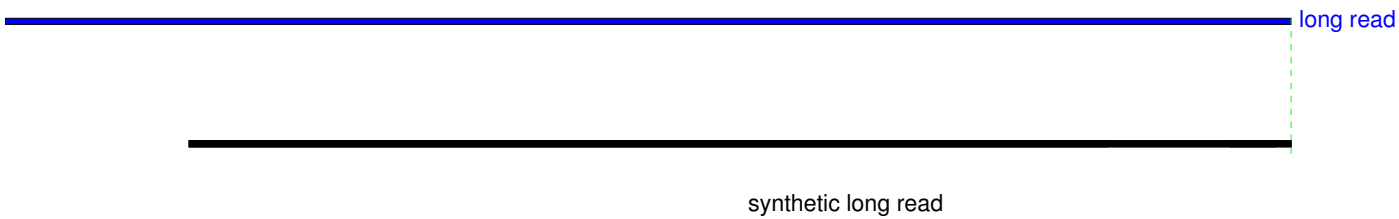
Step 5, first case : No ambiguity



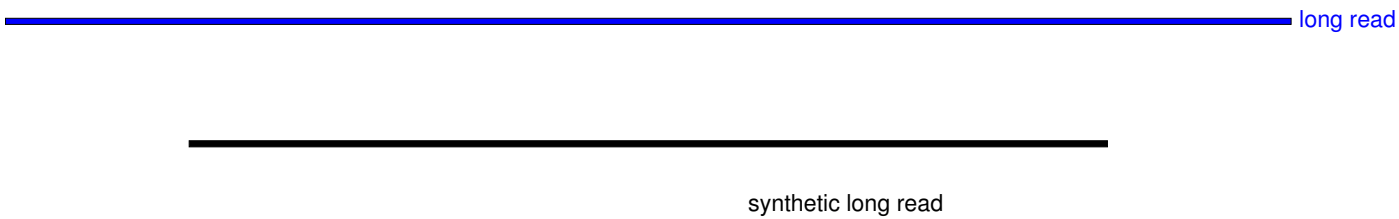
Step 5, first case : No ambiguity



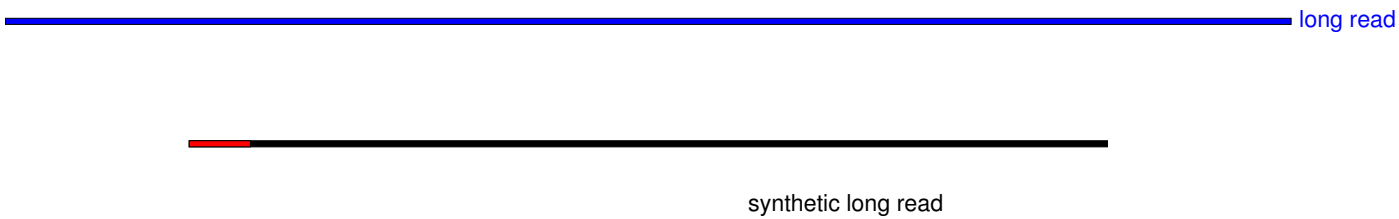
Step 5, first case : No ambiguity



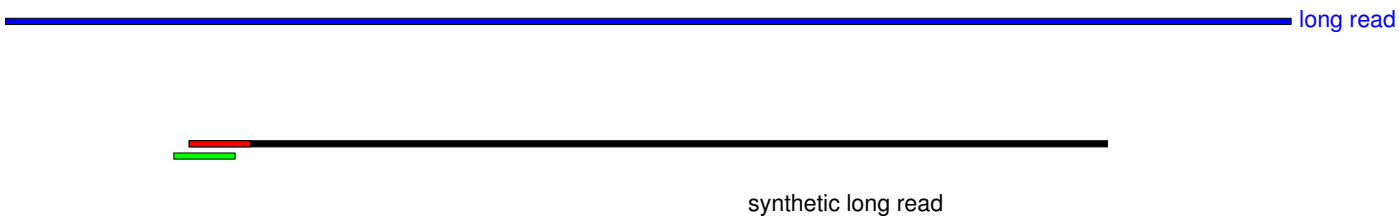
Step 5, second case : Ambiguity



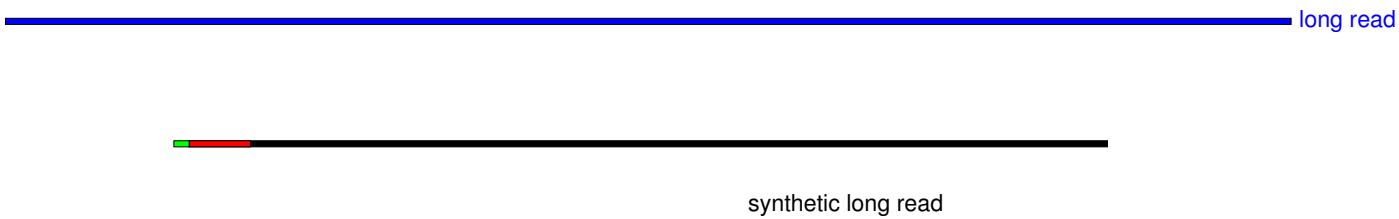
Step 5, second case : Ambiguity



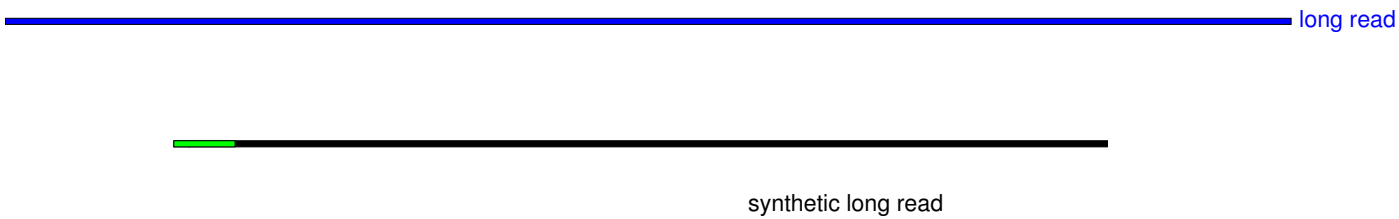
Step 5, second case : Ambiguity



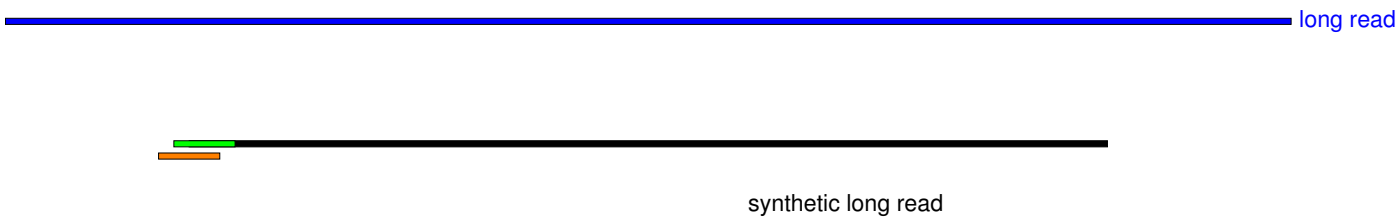
Step 5, second case : Ambiguity



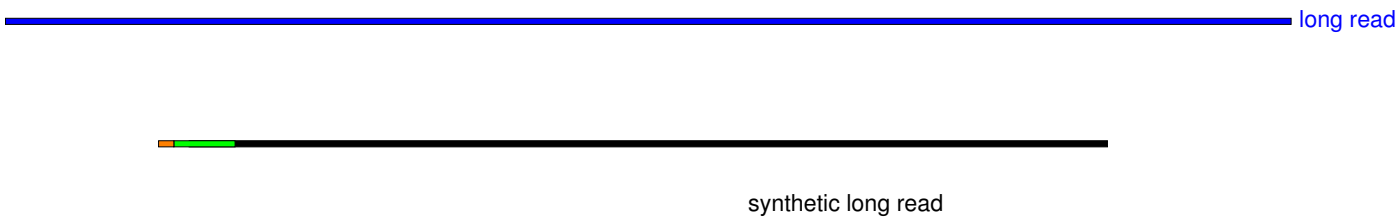
Step 5, second case : Ambiguity



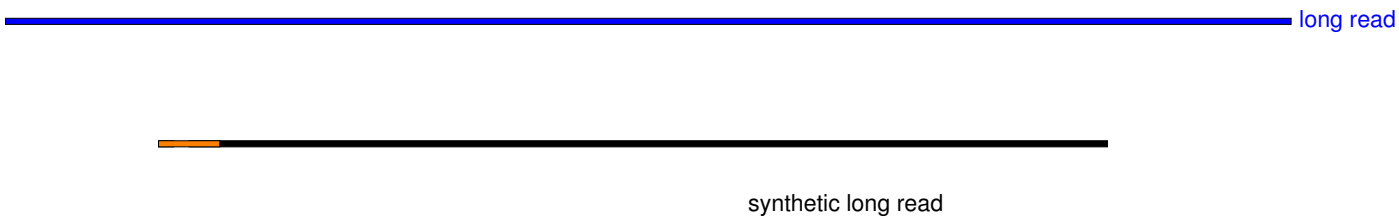
Step 5, second case : Ambiguity



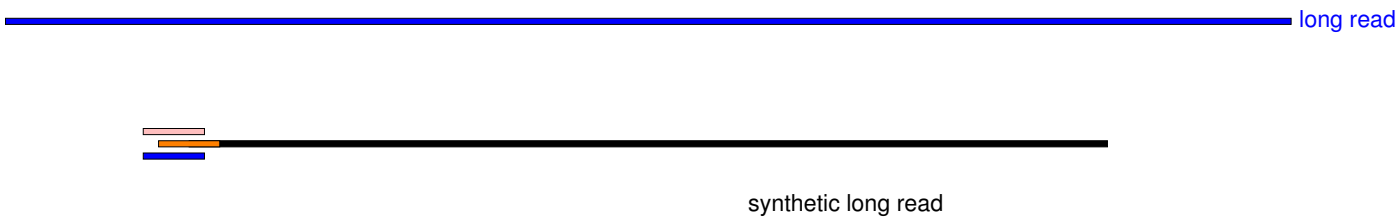
Step 5, second case : Ambiguity



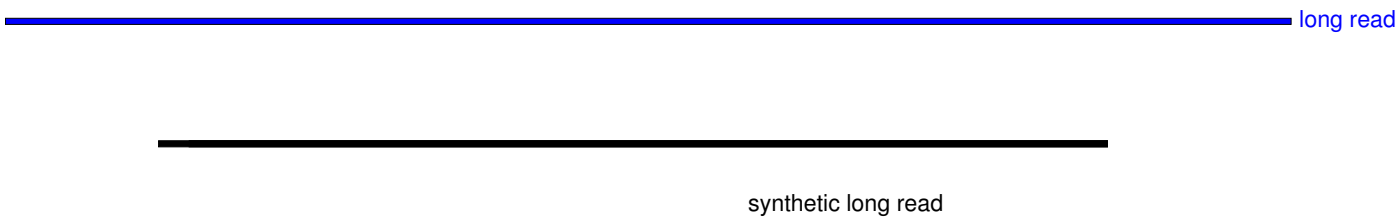
Step 5, second case : Ambiguity



Step 5, second case : Ambiguity



Step 5, second case : Ambiguity



Results and comparison with NaS : Mapping

On the 6 ADP1 long reads datasets available from the Genoscope website :

	Number of reads	Average length	Cumulative size	Average identity	s **Runtime**
Raw LRs	70 314	2 530	177 869 033	3,84 %	N.A.
NaS (fast)	8 219	4 514	37 099 564	99,92 %	TODO
NaS (sensitive)	12 053	6 338	76 388 104	99,89 %	TODO
Nous	7 425 (249 fragmented)	10 250	78 739 767	99,57 %	TODO

TABLE – On the 1D long reads

Results and comparison with NaS : Mapping

On the 6 ADP1 long reads datasets available from the Genoscope website :

	Number of reads	Average length	Cumulative size	Average identity	s Runtime
Raw LRs	70 314	2 530	177 869 033	3,84 %	N.A.
NaS (fast)	8 219	4 514	37 099 564	99,92 %	TODO
NaS (sensitive)	12 053	6 338	76 388 104	99,89 %	TODO
Nous	7 425 (249 fragmented)	10 250	78 739 767	99,57 %	TODO

TABLE – On the 1D long reads

	Number of reads	Average length	Cumulative size	Average identity	s Runtime
Raw LRs	18 697	10 884	203 496 742	37,07 %	N.A.
NaS (fast)	15 844	11 084	175 607 625	99,78 %	TODO
NaS (sensitive)	16 439	11 871	195 138 674	99,79 %	TODO
Nous	15 575 (984 fragmented)	10 562	178 222 404	99,54 %	TODO

TABLE – On the 2D long reads

Results and comparison with NaS : Assembly

With the two 1D and 2D datasets previously corrected :

Tool	Number of reads	Number of contigs	Genome coverage	Identity
NaS (fast)	24,063	1	100 %	99.98 %
NaS (sensitive)	28,492	1	100 %	99.99 %
Our method	TODO	TODO	TODO	TODO

TABLE – Assembly results

To do

- Try to produce less fragmented synthetic long reads $\Rightarrow$ Tuning of BLASR parameters

To do

- Try to produce less fragmented synthetic long reads $\Rightarrow$ Tuning of BLASR parameters
- Allow the correction to run in parallel to reduce runtime

Méthodes existantes

- LoRDEC (2014, Hybrid, short reads DBG)
- Jabba (2016, Hybrid, short reads DBG)
- LoRMA (2016, Only LRs, long reads DBG)