Correction et assemblage de reads longs

Pierre Morisse

27 février 2017

Plan de la présentation

Idée

- Comme NaS, produire des *reads* longs synthétiques à partir d'un assemblage de *reads* courts
- Se débarrasser de l'étape d'alignement des *reads* courts entre eux
- Ne déduire des informations qu'à partir de l'alignement des *reads* courts sur les *reads* longs

Fonctionnement

Nous présentons la méthode pour le traitement d'un *read* long

Fonctionnement

Nous présentons la méthode pour le traitement d'un *read* long

Principe

Alignement des *reads* courts sur le *read* long *template*, en se fixant un seuil *lmin*, pour récupérer les *reads* :

Fonctionnement

Nous présentons la méthode pour le traitement d'un *read* long

Principe

Alignement des *reads* courts sur le *read* long *template*, en se fixant un seuil *lmin*, pour récupérer les *reads* :

- Totalement alignés, et servant de *seeds*

Fonctionnement

Nous présentons la méthode pour le traitement d'un *read* long

Principe

Alignement des *reads* courts sur le *read* long *template*, en se fixant un seuil *lmin*, pour récupérer les *reads* :

- Totalement alignés, et servant de *seeds*
- Avec un préfixe de longueur $\geqslant$ *lmin* aligné

Fonctionnement

Nous présentons la méthode pour le traitement d'un *read* long

Principe

Alignement des *reads* courts sur le *read* long *template*, en se fixant un seuil *lmin*, pour récupérer les *reads* :

- Totalement alignés, et servant de *seeds*
- Avec un préfixe de longueur $\geqslant$ *lmin* aligné
- Avec un suffixe de longueur $\geqslant$ *lmin* aligné

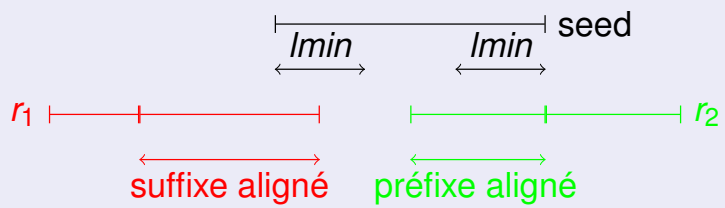
Fonctionnement

Deux étapes d'extensions :

Fonctionnement

Deux étapes d'extensions :

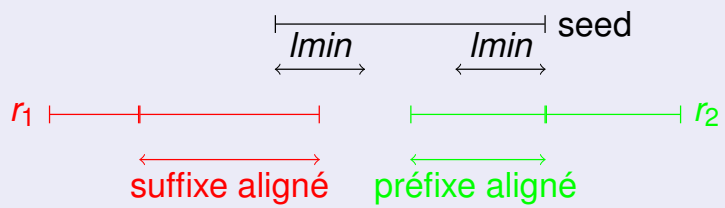
1. Recrutement de *reads* partiellement alignés, similaires aux *seeds*



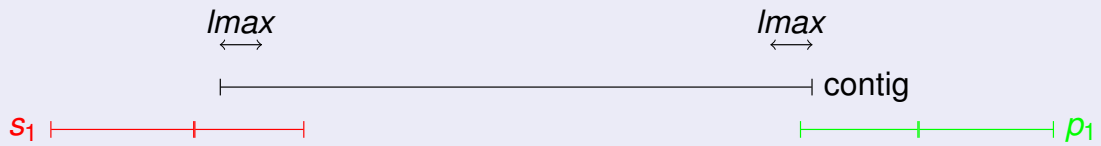
Fonctionnement

Deux étapes d'extensions :

1. Recrutement de *reads* partiellement alignés, similaires aux *seeds*



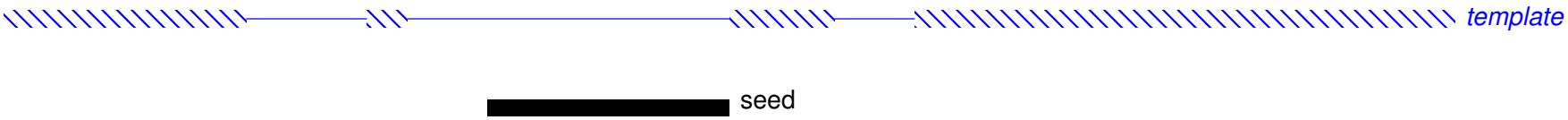
2. Recrutement de nouveaux *reads* partiellement alignés, sans relation de similarité, en se fixant un nouveau seuil *lmax*



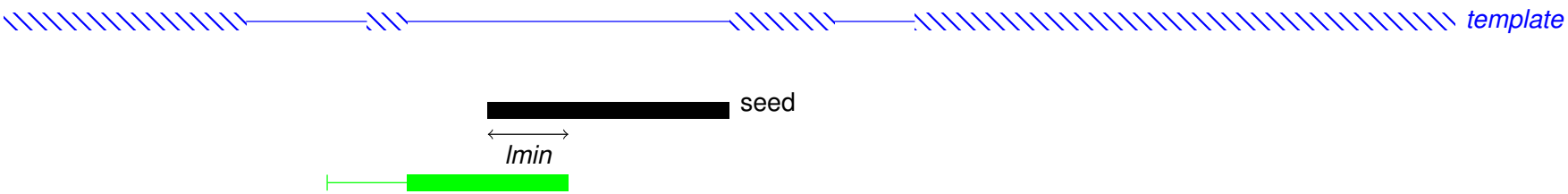
Exemple

template

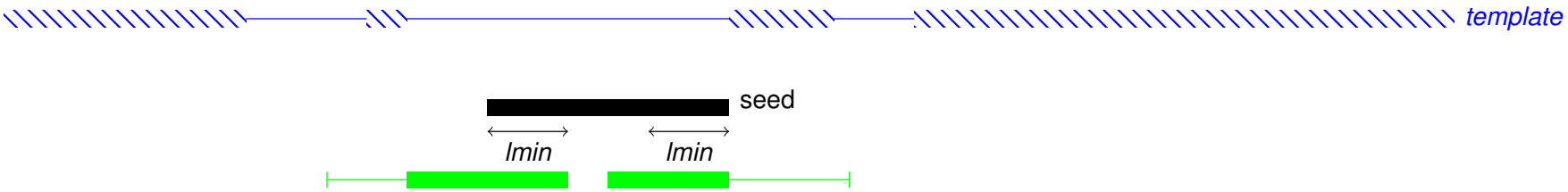
Exemple



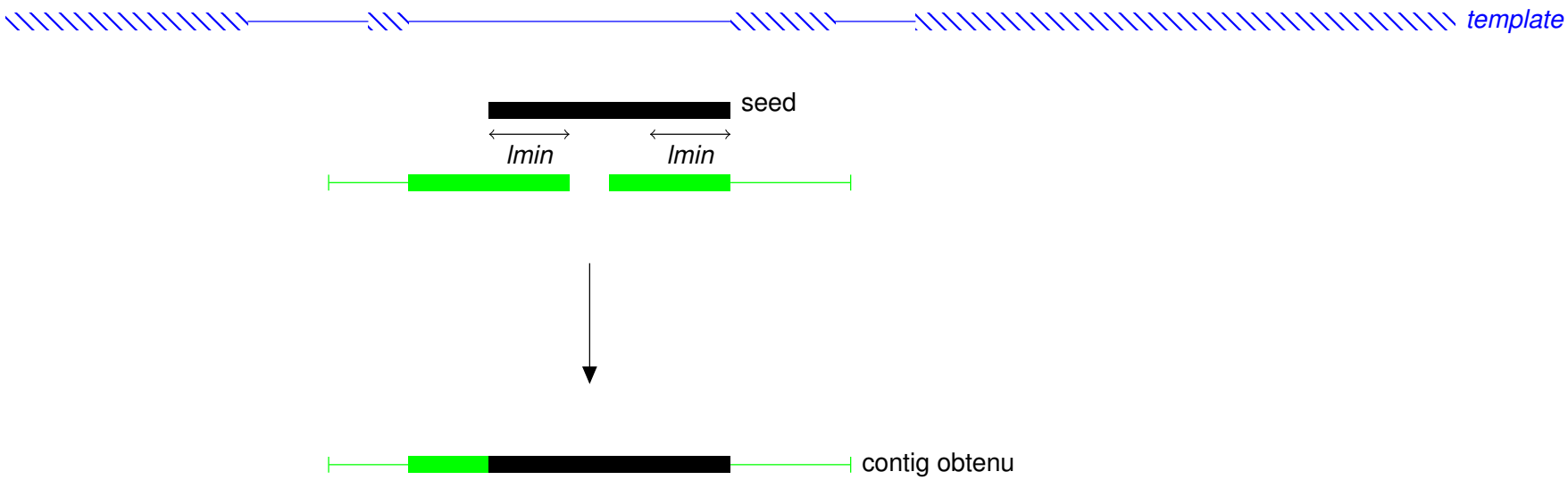
Exemple



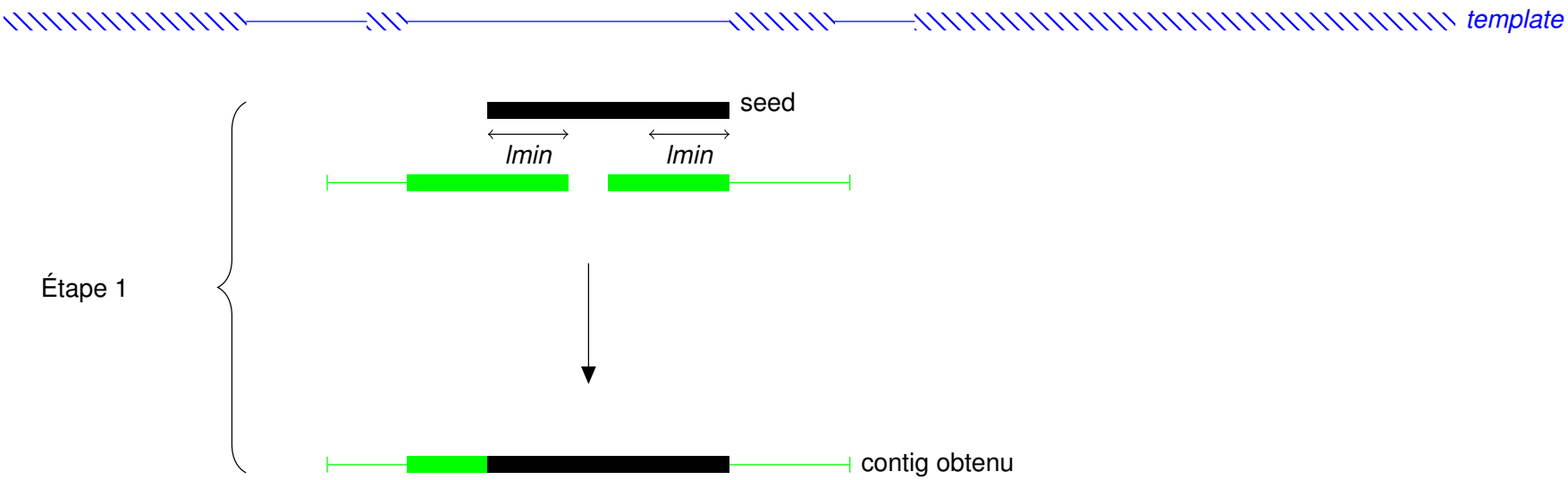
Exemple



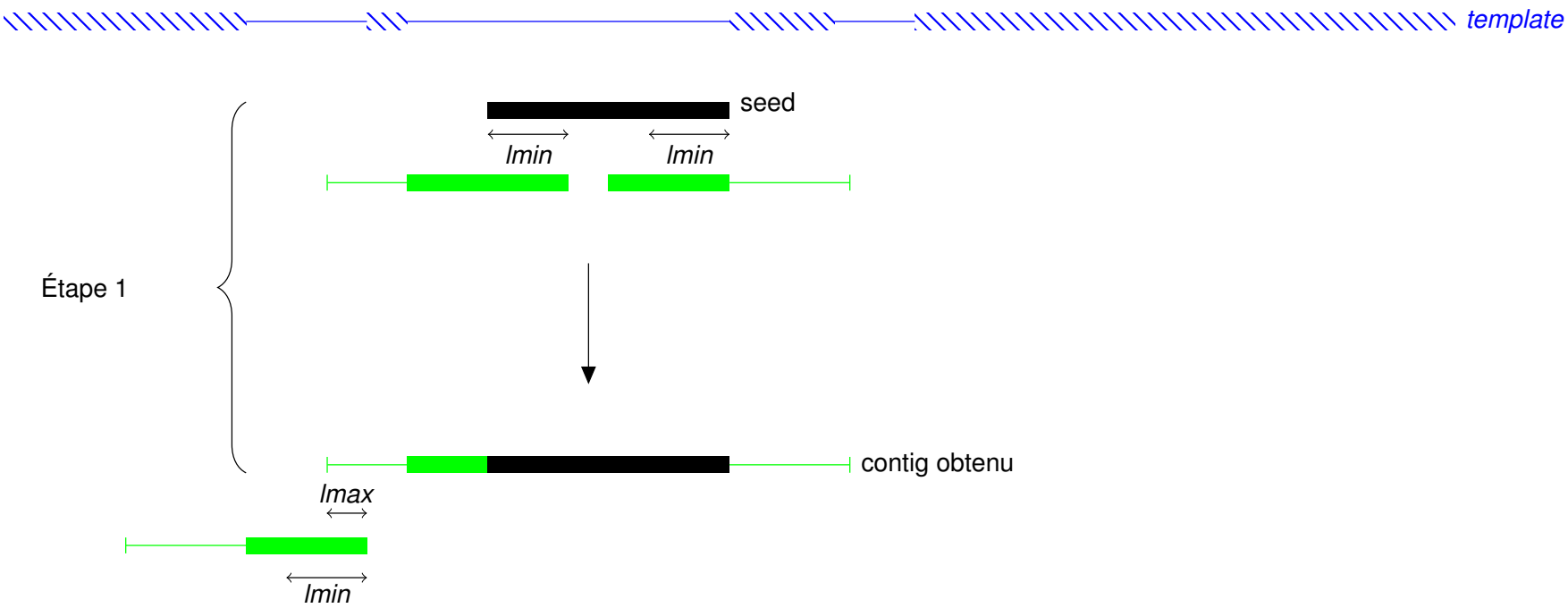
Exemple



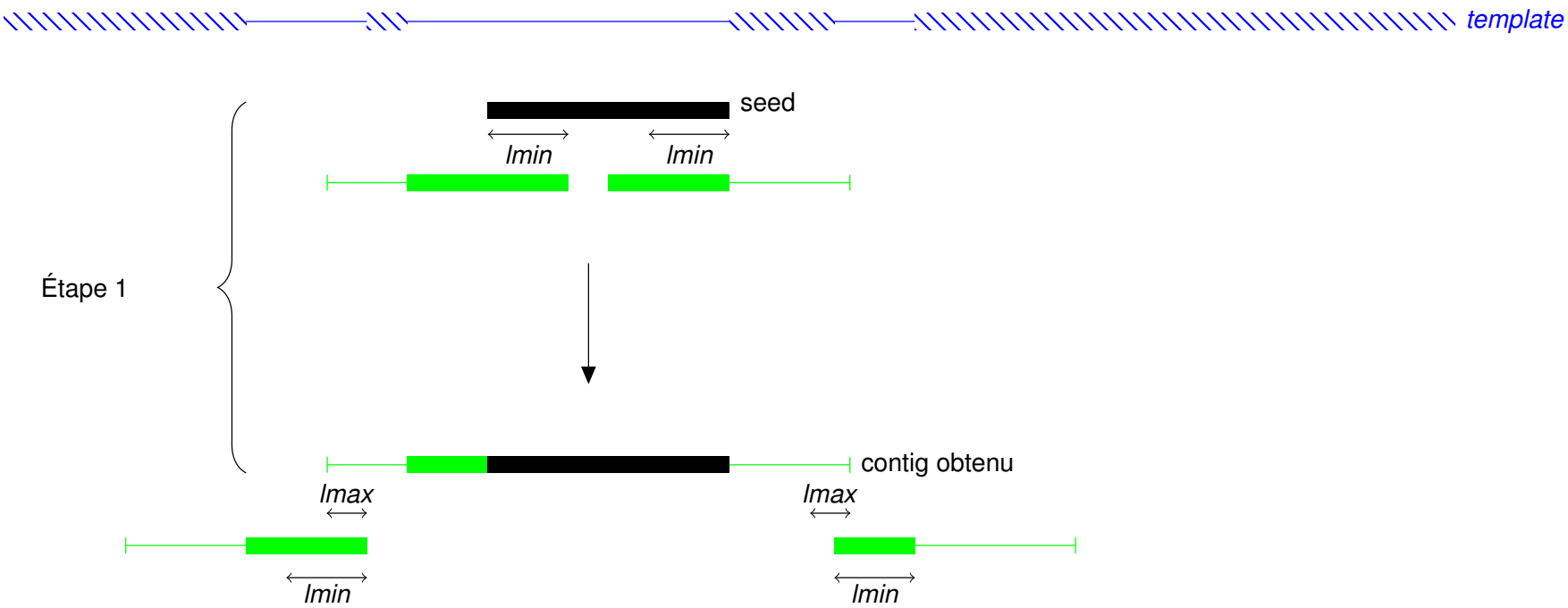
Exemple



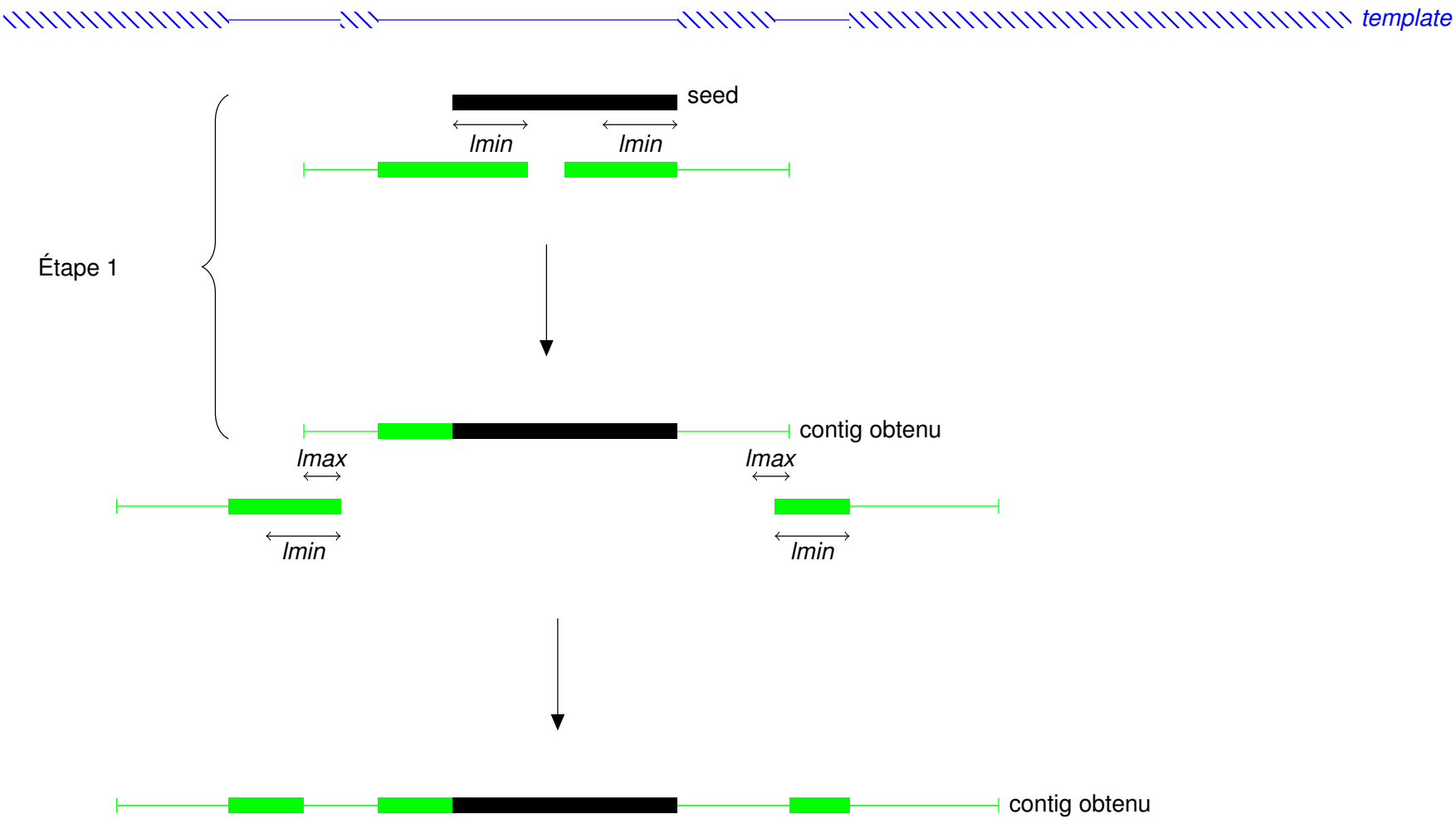
Exemple



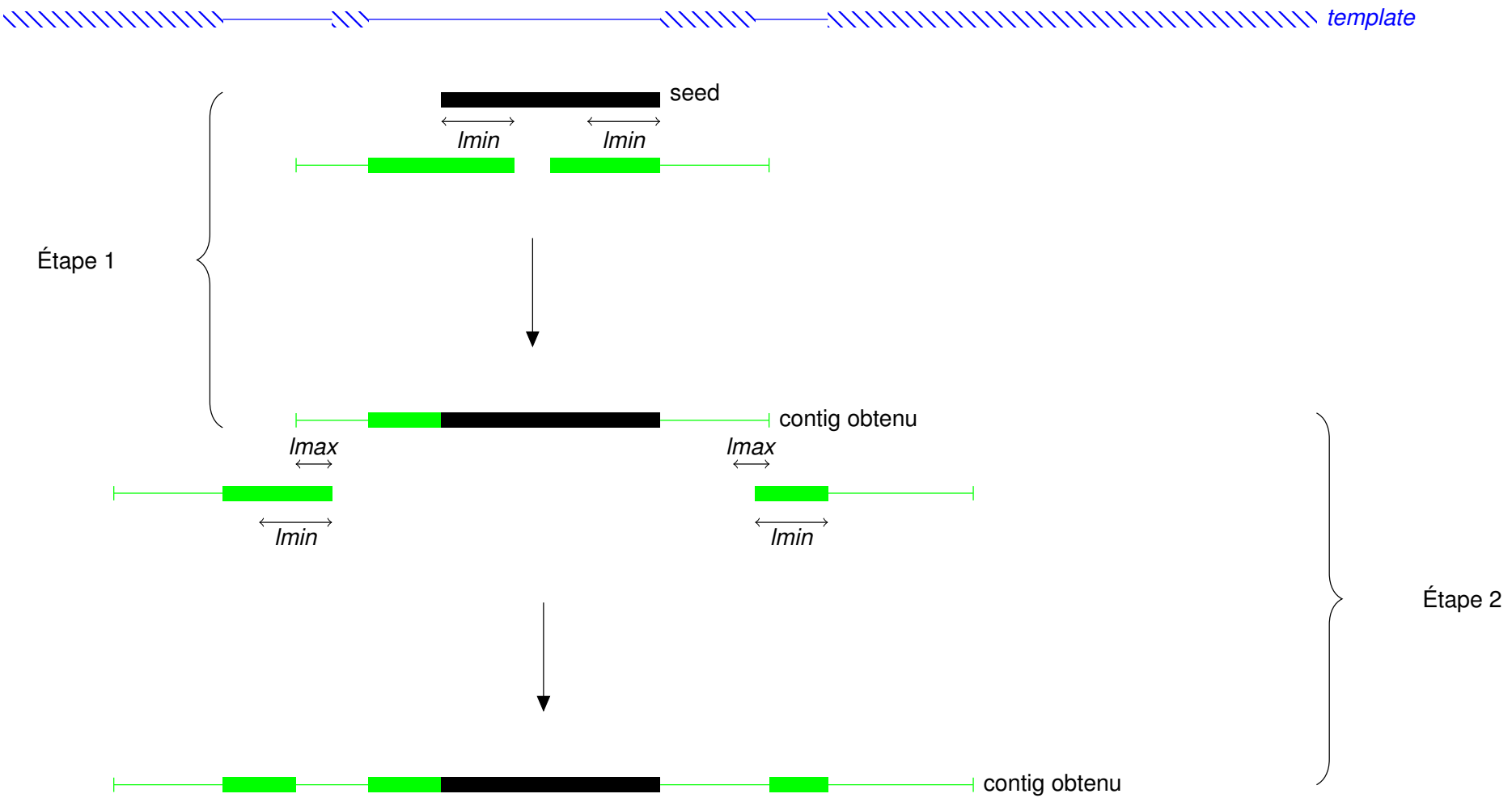
Exemple


template
seed
lmin
lmin
Étape 1
contig obtenu
lmax
lmax
lmin
lmin

Exemple


template
seed
lmin
lmin
Étape 1
contig obtenu
lmax
lmax
lmin
lmin
contig obtenu

Exemple


template
seed
lmin
lmin
Étape 1
contig obtenu
lmax
lmax
lmin
lmin
Étape 2
contig obtenu

Résultats et conclusions

Résultats après application de notre méthode sur un ensemble de *reads* longs ADP1, avec $lmin = 100$ et $lmax = 10$:

Résultats et conclusions

Résultats après application de notre méthode sur un ensemble de *reads* longs ADP1, avec $lmin = 100$ et $lmax = 10$:

Reads	Longueur moyenne	Précision moyenne	Contigs / *read*	Longueur moyenne	Précision moyenne	Template couvert	Temps
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

Résultats et conclusions

Résultats après application de notre méthode sur un ensemble de *reads* longs ADP1, avec $lmin = 100$ et $lmax = 10$:

Reads	Longueur moyenne	Précision moyenne	Contigs / *read*	Longueur moyenne	Précision moyenne	Template couvert	Temps
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

- Temps de traitement : moins de 10 secondes pour un *read* long

Résultats et conclusions

Résultats après application de notre méthode sur un ensemble de *reads* longs ADP1, avec $lmin = 100$ et $lmax = 10$:

Reads	Longueur moyenne	Précision moyenne	Contigs / *read*	Longueur moyenne	Précision moyenne	Template couvert	Temps
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

- Temps de traitement : moins de 10 secondes pour un *read* long
- Encore un taux d'erreurs de 12 %

Résultats et conclusions

Résultats après application de notre méthode sur un ensemble de *reads* longs ADP1, avec $lmin = 100$ et $lmax = 10$:

Reads	Longueur moyenne	Précision moyenne	Contigs / *read*	Longueur moyenne	Précision moyenne	Template couvert	Temps
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

- Temps de traitement : moins de 10 secondes pour un *read* long
- Encore un taux d'erreurs de 12 %
- Obtention de plusieurs contigs pour chaque *read* long, et faible taux de couverture de ces derniers

Résultats et conclusions

Résultats après application de notre méthode sur un ensemble de *reads* longs ADP1, avec $lmin = 100$ et $lmax = 10$:

Reads	Longueur moyenne	Précision moyenne	Contigs / *read*	Longueur moyenne	Précision moyenne	Template couvert	Temps
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

- Temps de traitement : moins de 10 secondes pour un *read* long
- Encore un taux d'erreurs de 12 %
- Obtention de plusieurs contigs pour chaque *read* long, et faible taux de couverture de ces derniers
- Modifier les paramètres ne permet pas d'obtenir de meilleurs résultats

Idée

- Obtenir des contigs à partir d'un assemblage de *reads* courts

Idée

- Obtenir des contigs à partir d'un assemblage de *reads* courts
- Aligner les *reads* longs sur ces contigs avec BLASR pour trouver des alignements locaux

Idée

- Obtenir des contigs à partir d'un assemblage de *reads* courts
- Aligner les *reads* longs sur ces contigs avec BLASR pour trouver des alignements locaux
- Relier et ordonner les contigs grâce aux *reads* longs s'alignant sur plusieurs contigs

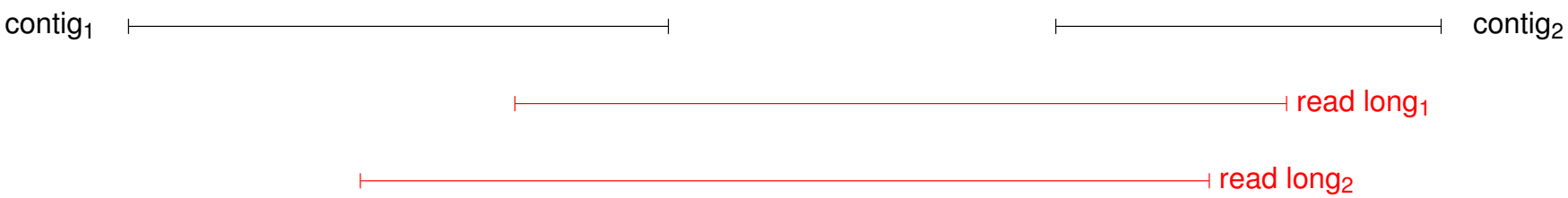
Idée

- Obtenir des contigs à partir d'un assemblage de *reads* courts
- Aligner les *reads* longs sur ces contigs avec BLASR pour trouver des alignements locaux
- Relier et ordonner les contigs grâce aux *reads* longs s'alignant sur plusieurs contigs
- Combler les gaps entre deux contigs liés par un consensus des bases des *reads* longs

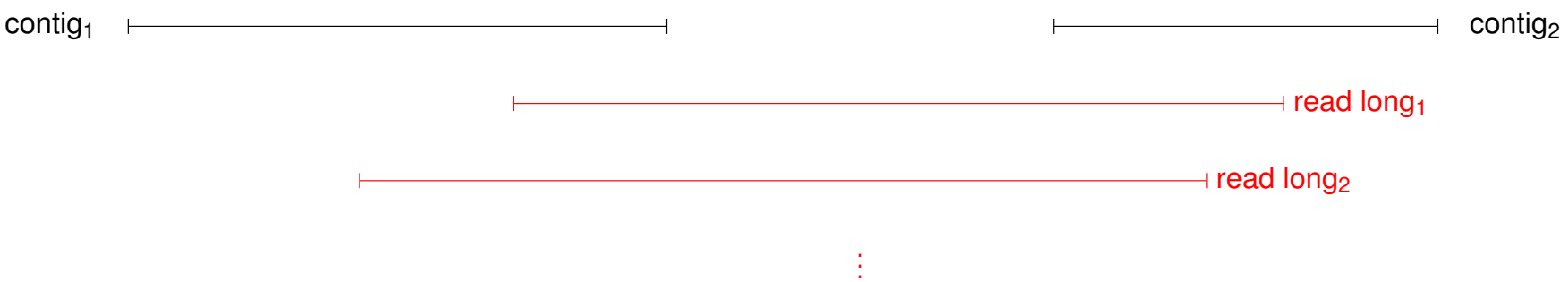
Exemple

Exemple

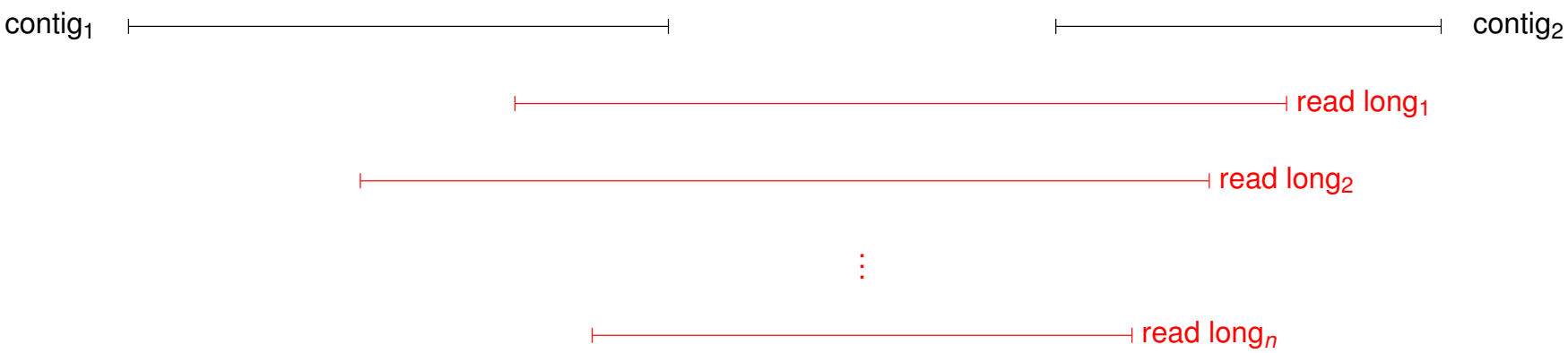
Exemple



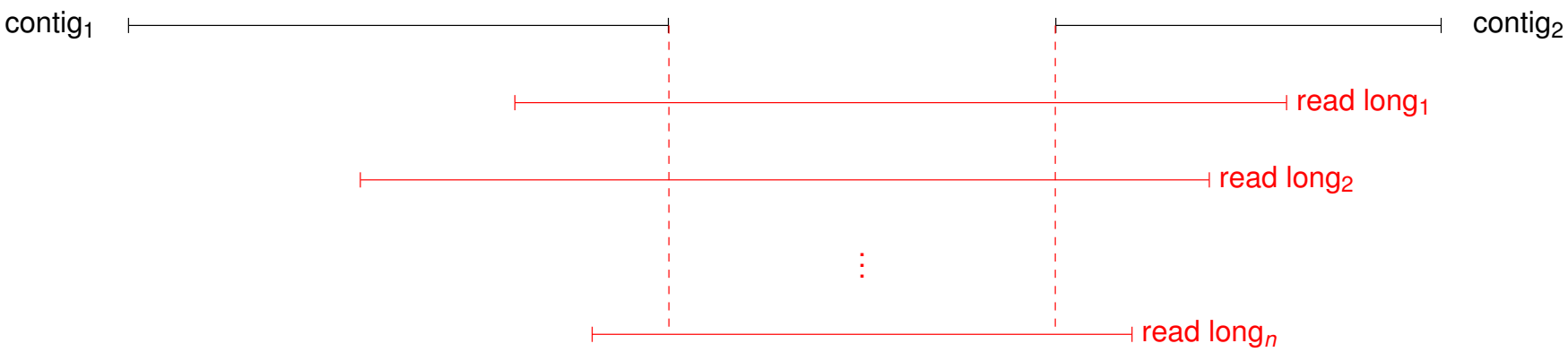
Exemple



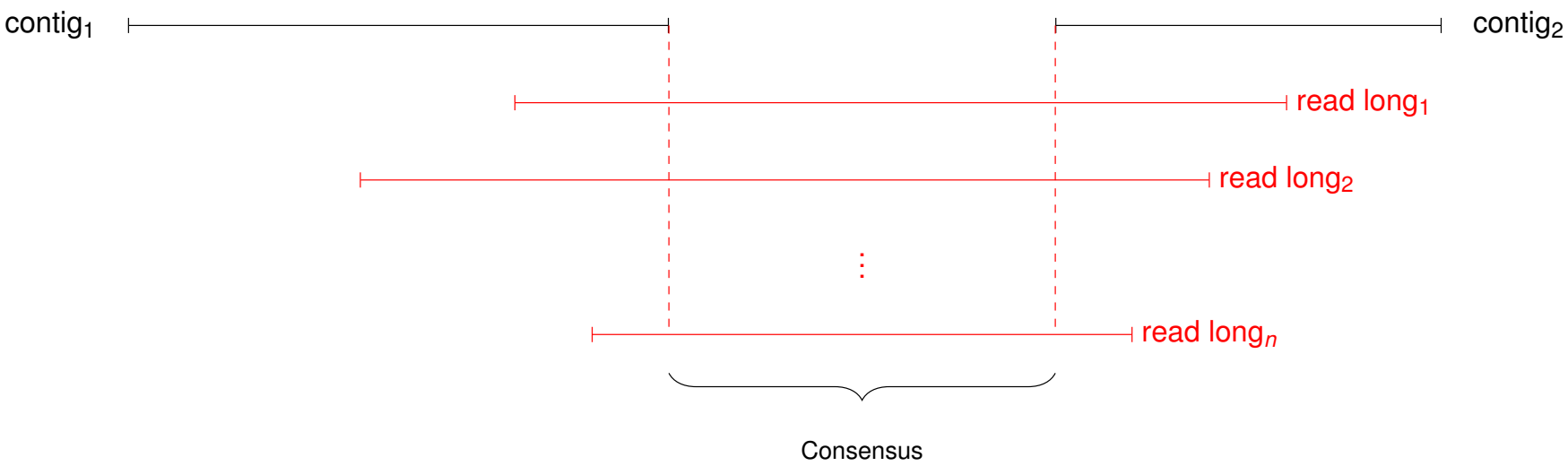
Exemple



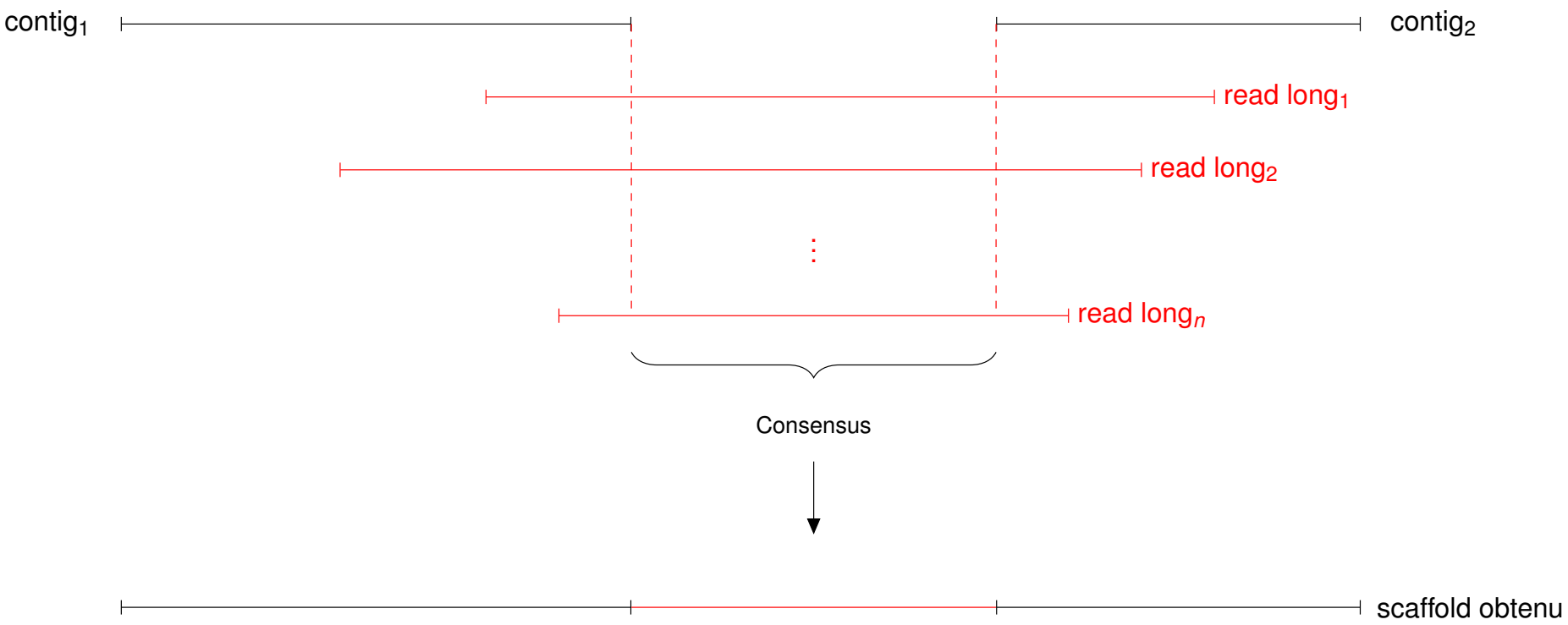
Exemple



Exemple


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

Exemple


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

Remarques et conclusion

- Méthode déjà existante : SSPACE-LongRead permet déjà de scaffolder des contigs à l'aide de *reads* longs, mais comble les gaps avec des N plutôt qu'avec un consensus des bases des *reads* longs

Remarques et conclusion

- Méthode déjà existante : SSPACE-LongRead permet déjà de scaffolder des contigs à l'aide de *reads* longs, mais comble les gaps avec des N plutôt qu'avec un consensus des bases des *reads* longs
- PBJelly permet également de scaffolder les contigs, mais quant à lui, comble également les gaps avec les bases des *reads* longs

Remarques et conclusion

- Méthode déjà existante : SSPACE-LongRead permet déjà de scaffolder des contigs à l'aide de *reads* longs, mais comble les gaps avec des N plutôt qu'avec un consensus des bases des *reads* longs
- PBJelly permet également de scaffolder les contigs, mais quant à lui, comble également les gaps avec les bases des *reads* longs
- Inutile de développer davantage cette idée

Idée

- Récupérer les *k*-mers apparaissant dans les *reads* longs mais pas dans les contigs de *reads* courts

Idée

- Récupérer les *k*-mers apparaissant dans les *reads* longs mais pas dans les contigs de *reads* courts
- Assembler les *k*-mers récupérés $\Rightarrow$ Obtention de nouveau contigs

Idée

- Récupérer les *k*-mers apparaissant dans les *reads* longs mais pas dans les contigs de *reads* courts
- Assembler les *k*-mers récupérés $\Rightarrow$ Obtention de nouveau contigs
- Utiliser ces nouveaux contigs pour couvrir les zones du génome de référence non couvertes par les contigs de *reads* courts

Idée

- Récupérer les *k*-mers apparaissant dans les *reads* longs mais pas dans les contigs de *reads* courts
- Assembler les *k*-mers récupérés $\Rightarrow$ Obtention de nouveau contigs
- Utiliser ces nouveaux contigs pour couvrir les zones du génome de référence non couvertes par les contigs de *reads* courts
- Permettre un assemblage moins fragmenté en assemblant les contigs de *reads* courts et les contigs de *reads* longs

Tests et résultats

Avec un ensemble de *reads* de ADP1 :

Taille d'un *k*-mer	Nombre de *k*-mers	Nombre de *k*-mers dans les 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 et résultats

Avec un ensemble de *reads* de ADP1 :

Taille d'un *k*-mer	Nombre de *k*-mers	Nombre de *k*-mers dans les 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

Peu concluant, hormis pour les 8-mers, mais ceux-ci n'apportent pas d'information pertinente

Tests et résultats

Avec un ensemble de *reads* de ADP1 :

Taille d'un *k*-mer	Nombre de *k*-mers	Nombre de *k*-mers dans les 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

Peu concluant, hormis pour les 8-mers, mais ceux-ci n'apportent pas d'information pertinente

$\Rightarrow$ Plus intéressant rechercher des *k*-mers espacés ?

k-mers espacés

Définition

Ici, on appelle k-mer espacé un k-mer dans lequel on autorise un gap d'une certaine longueur au centre.

k-mers espacés

Définition

Ici, on appelle *k*-mer espacé un *k*-mer dans lequel on autorise un gap d'une certaine longueur au centre.

Exemple

Par exemple, le 8-mer GATCTTAC, si on autorise un gap de longueur 2, deviendra le "8"-mer-2-espacé suivant : GATC**TTAC, où les * désignent des positions jokers (match ou mismatch autorisé)

k-mers espacés

Définition

Ici, on appelle *k*-mer espacé un *k*-mer dans lequel on autorise un gap d'une certaine longueur au centre.

Exemple

Par exemple, le 8-mer GATCTTAC, si on autorise un gap de longueur 2, deviendra le "8"-mer-2-espacé suivant : GATC**TTAC, où les * désignent des positions jokers (match ou mismatch autorisé)

On utilise cette définition des *k*-mers espacés plutôt que la définition classique afin de pouvoir prendre en compte les erreurs d'indels plutôt que les erreurs de substitution

Tests et résultats

Résultats, sur le jeu de données précédent, en autorisant un gap de longueur comprise entre 0 et 10 :

Taille d'un *k*-mer	Nombre de *k*-mers	Nombre de *k*-mers dans les contigs
64	375 999 371	425 155
32	335 836 269	3 859 742
16	322 493 916	16 036 610

Tests et résultats

Résultats, sur le jeu de données précédent, en autorisant un gap de longueur comprise entre 0 et 10 :

Taille d'un k-mer	Nombre de k-mers	Nombre de k-mers dans les contigs
64	375 999 371	425 155
32	335 836 269	3 859 742
16	322 493 916	16 036 610

$\Rightarrow$ La comparaison n'est toujours pas concluante, même avec des k-mers espacés

Remarques

- Trimmer les extrémités des *reads* longs ne permet pas d'obtenir de meilleurs résultats $\Rightarrow$ Contrairement aux *reads* courts, les erreurs ne se trouvent pas majoritairement aux extrémités, mais partout dans les *reads* longs

Remarques

- Trimmer les extrémités des *reads* longs ne permet pas d'obtenir de meilleurs résultats $\Rightarrow$ Contrairement aux *reads* courts, les erreurs ne se trouvent pas majoritairement aux extrémités, mais partout dans les *reads* longs

- Comparer les *k*-mers des *reads* longs directement avec les *k*-mers des *reads* courts plutôt qu'avec les contigs ne permet pas non plus d'obtenir de meilleurs résultats

Remarques

- Trimmer les extrémités des *reads* longs ne permet pas d'obtenir de meilleurs résultats $\Rightarrow$ Contrairement aux *reads* courts, les erreurs ne se trouvent pas majoritairement aux extrémités, mais partout dans les *reads* longs

- Comparer les *k*-mers des *reads* longs directement avec les *k*-mers des *reads* courts plutôt qu'avec les contigs ne permet pas non plus d'obtenir de meilleurs résultats

- Approche non concluante

Remarques

- Trimmer les extrémités des *reads* longs ne permet pas d'obtenir de meilleurs résultats $\Rightarrow$ Contrairement aux *reads* courts, les erreurs ne se trouvent pas majoritairement aux extrémités, mais partout dans les *reads* longs

- Comparer les *k*-mers des *reads* longs directement avec les *k*-mers des *reads* courts plutôt qu'avec les contigs ne permet pas non plus d'obtenir de meilleurs résultats

- Approche non concluante

- Autoriser plus de gaps dans les *k*-mers, et non un seul au milieu, pourrait potentiellement amener de meilleurs résultats => Besoin d'un algorithme pour construire une table des suffixes espacée

Principe

5 étapes :

Principe

5 étapes :

1. Correction des *reads* courts (avec Quorum)

Principe

5 étapes :

1. Correction des *reads* courts (avec Quorum)
2. Alignement des *reads* courts sur le *read* long, afin de trouver des *seeds* (avec BLASR)

Principe

5 étapes :

1. Correction des *reads* courts (avec Quorum)
2. Alignement des *reads* courts sur le *read* long, afin de trouver des *seeds* (avec BLASR)
3. Fusion des *seeds* se chevauchant sur une longueur assez importante

Principe

5 étapes :

1. Correction des *reads* courts (avec Quorum)
2. Alignement des *reads* courts sur le *read* long, afin de trouver des *seeds* (avec BLASR)
3. Fusion des *seeds* se chevauchant sur une longueur assez importante
4. Relier les *seeds* en les étendant à l'aide de chevauchements parfaits avec les *k*-mers des *reads* courts

Principe

5 étapes :

1. Correction des *reads* courts (avec Quorum)
2. Alignement des *reads* courts sur le *read* long, afin de trouver des *seeds* (avec BLASR)
3. Fusion des *seeds* se chevauchant sur une longueur assez importante
4. Relier les *seeds* en les étendant à l'aide de chevauchements parfaits avec les *k*-mers des *reads* courts
5. Extension du *read* long sythétique obtenu, à gauche (resp. à droite) du *seed* le plus à gauche (resp. à droite)

Outil utilisé : PgSA

PgSA (Pseudogenome Suffix Array) permet d'indexer un ensemble de *reads*, et de répondre aux 7 requêtes suivantes, pour une chaîne f donnée :

Outil utilisé : PgSA

PgSA (Pseudogenome Suffix Array) permet d'indexer un ensemble de *reads*, et de répondre aux 7 requêtes suivantes, pour une chaîne f donnée :

1. Dans quels *reads* f apparaît ?

Outil utilisé : PgSA

PgSA (Pseudogenome Suffix Array) permet d'indexer un ensemble de *reads*, et de répondre aux 7 requêtes suivantes, pour une chaîne f donnée :

1. Dans quels *reads* f apparaît ?
2. Dans combien de *reads* f apparaît ?

Outil utilisé : PgSA

PgSA (Pseudogenome Suffix Array) permet d'indexer un ensemble de *reads*, et de répondre aux 7 requêtes suivantes, pour une chaîne f donnée :

1. Dans quels *reads* f apparaît ?
2. Dans combien de *reads* f apparaît ?
3. Quelles sont les occurrences de f ?

Outil utilisé : PgSA

PgSA (Pseudogenome Suffix Array) permet d'indexer un ensemble de *reads*, et de répondre aux 7 requêtes suivantes, pour une chaîne f donnée :

1. Dans quels *reads* f apparaît ?
2. Dans combien de *reads* f apparaît ?
3. Quelles sont les occurrences de f ?
4. Quel est le nombre d'occurrences de f ?

Outil utilisé : PgSA

PgSA (Pseudogenome Suffix Array) permet d'indexer un ensemble de *reads*, et de répondre aux 7 requêtes suivantes, pour une chaîne *f* donnée :

1. Dans quels *reads f* apparaît ?
2. Dans combien de *reads f* apparaît ?
3. Quelles sont les occurrences de *f* ?
4. Quel est le nombre d'occurrences de *f* ?
5. Dans quels *reads f* n'apparaît qu'une fois ?

Outil utilisé : PgSA

PgSA (Pseudogenome Suffix Array) permet d'indexer un ensemble de *reads*, et de répondre aux 7 requêtes suivantes, pour une chaîne f donnée :

1. Dans quels *reads* f apparaît ?
2. Dans combien de *reads* f apparaît ?
3. Quelles sont les occurrences de f ?
4. Quel est le nombre d'occurrences de f ?
5. Dans quels *reads* f n'apparaît qu'une fois ?
6. Dans combien de *reads* f n'apparaît qu'une fois ?

Outil utilisé : PgSA

PgSA (Pseudogenome Suffix Array) permet d'indexer un ensemble de *reads*, et de répondre aux 7 requêtes suivantes, pour une chaîne f donnée :

1. Dans quels *reads* f apparaît ?
2. Dans combien de *reads* f apparaît ?
3. Quelles sont les occurrences de f ?
4. Quel est le nombre d'occurrences de f ?
5. Dans quels *reads* f n'apparaît qu'une fois ?
6. Dans combien de *reads* f n'apparaît qu'une fois ?
7. Quelles sont les occurrences de f dans les *reads* où f n'apparaît qu'une fois ?

Outil utilisé : PgSA

PgSA (Pseudogenome Suffix Array) permet d'indexer un ensemble de *reads*, et de répondre aux 7 requêtes suivantes, pour une chaîne f donnée :

1. Dans quels *reads* f apparaît ?
2. Dans combien de *reads* f apparaît ?
3. Quelles sont les occurrences de f ?
4. Quel est le nombre d'occurrences de f ?
5. Dans quels *reads* f n'apparaît qu'une fois ?
6. Dans combien de *reads* f n'apparaît qu'une fois ?
7. Quelles sont les occurrences de f dans les *reads* où f n'apparaît qu'une fois ?

Parmi ces requêtes, la 3ème va nous permettre de trouver des chevauchements parfaits entre les k-mers

Outil utilisé : PgSA

D'autres structures (Gk-Arrays, Compressed Gk-Arrays) permettent le traitement des ces requêtes, mais la longueur k de f doit être fixée à la compilation, alors que PgSA permet de traiter les requêtes pour des valeurs de k variables.

Outil utilisé : PgSA

D'autres structures (Gk-Arrays, Compressed Gk-Arrays) permettent le traitement des ces requêtes, mais la longueur k de f doit être fixée à la compilation, alors que PgSA permet de traiter les requêtes pour des valeurs de k variables.

=> Permet de chercher des chevauchements de longueur $k-2$ si aucun chevauchement de longueur $k-1$ n'a été trouvé, sans avoir besoin de recalculer l'index

Étape 4

Indexation de l'ensemble des *k*-mers des *reads* courts (et de leurs reverse-complement) avec PgSA, et boucle sur la requête 3 afin de trouver des chevauchements parfaits de *k*-mers permettant de relier les *seeds* entre eux

Étape 4

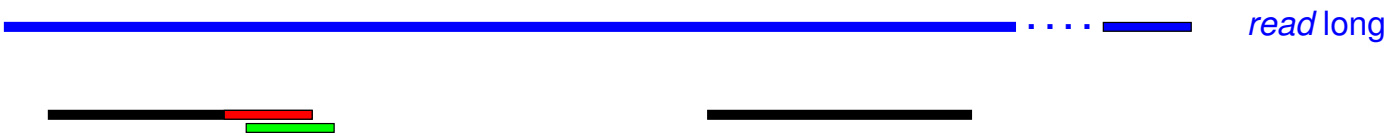
Indexation de l'ensemble des k-mers des *reads* courts (et de leurs reverse-complement) avec PgSA, et boucle sur la requête 3 afin de trouver des chevauchements parfaits de k-mers permettant de relier les *seeds* entre eux

Étape 4

Indexation de l'ensemble des k-mers des *reads* courts (et de leurs reverse-complement) avec PgSA, et boucle sur la requête 3 afin de trouver des chevauchements parfaits de k-mers permettant de relier les *seeds* entre eux

Étape 4

Indexation de l'ensemble des k-mers des *reads* courts (et de leurs reverse-complement) avec PgSA, et boucle sur la requête 3 afin de trouver des chevauchements parfaits de k-mers permettant de relier les *seeds* entre eux



Étape 4

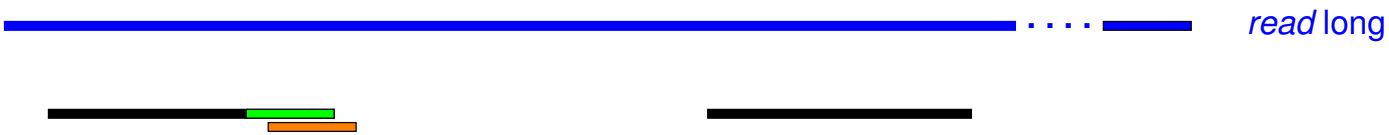
Indexation de l'ensemble des *k*-mers des *reads* courts (et de leurs reverse-complement) avec PgSA, et boucle sur la requête 3 afin de trouver des chevauchements parfaits de *k*-mers permettant de relier les *seeds* entre eux

Étape 4

Indexation de l'ensemble des *k*-mers des *reads* courts (et de leurs reverse-complement) avec PgSA, et boucle sur la requête 3 afin de trouver des chevauchements parfaits de *k*-mers permettant de relier les *seeds* entre eux

Étape 4

Indexation de l'ensemble des k-mers des *reads* courts (et de leurs reverse-complement) avec PgSA, et boucle sur la requête 3 afin de trouver des chevauchements parfaits de k-mers permettant de relier les *seeds* entre eux



Étape 4

Indexation de l'ensemble des *k*-mers des *reads* courts (et de leurs reverse-complement) avec PgSA, et boucle sur la requête 3 afin de trouver des chevauchements parfaits de *k*-mers permettant de relier les *seeds* entre eux

Étape 4

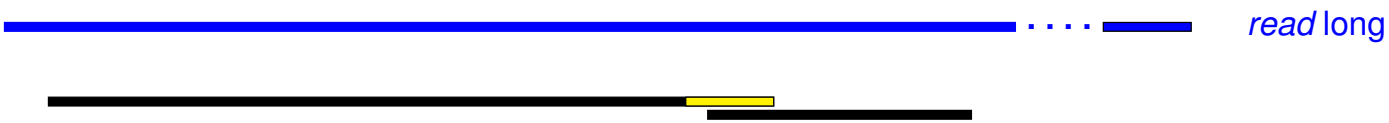
Indexation de l'ensemble des k-mers des *reads* courts (et de leurs reverse-complement) avec PgSA, et boucle sur la requête 3 afin de trouver des chevauchements parfaits de k-mers permettant de relier les *seeds* entre eux

Étape 4

Indexation de l'ensemble des k-mers des *reads* courts (et de leurs reverse-complement) avec PgSA, et boucle sur la requête 3 afin de trouver des chevauchements parfaits de k-mers permettant de relier les *seeds* entre eux

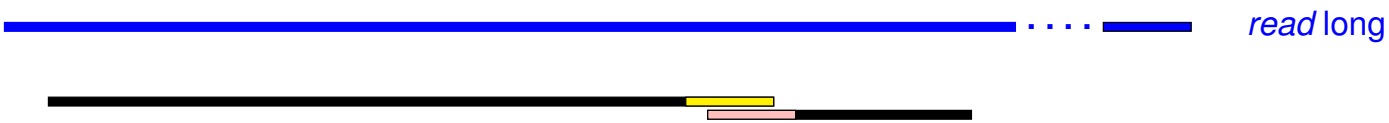
Étape 4

Indexation de l'ensemble des *k*-mers des *reads* courts (et de leurs reverse-complement) avec PgSA, et boucle sur la requête 3 afin de trouver des chevauchements parfaits de *k*-mers permettant de relier les *seeds* entre eux



Étape 4

Indexation de l'ensemble des k-mers des *reads* courts (et de leurs reverse-complement) avec PgSA, et boucle sur la requête 3 afin de trouver des chevauchements parfaits de k-mers permettant de relier les *seeds* entre eux



Étape 4

Indexation de l'ensemble des k-mers des *reads* courts (et de leurs reverse-complement) avec PgSA, et boucle sur la requête 3 afin de trouver des chevauchements parfaits de k-mers permettant de relier les *seeds* entre eux

Remarques

- Il est possible qu'un k-mer chevauche parfaitement plusieurs k-mers $\Rightarrow$ Exploration de toutes les extensions possibles avec du backtracking

Remarques

- Il est possible qu'un k-mer chevauche parfaitement plusieurs k-mers $\Rightarrow$ Exploration de toutes les extensions possibles avec du backtracking

- Certains *seeds* peuvent être impossibles à relier $\Rightarrow$ Production d'un *read* long synthétique fragmenté en plusieurs parties

Remarques

- Il est possible qu'un k-mer chevauche parfaitement plusieurs k-mers $\Rightarrow$ Exploration de toutes les extensions possibles avec du backtracking

- Certains *seeds* peuvent être impossibles à relier $\Rightarrow$ Production d'un *read* long synthétique fragmenté en plusieurs parties

Remarques

- Il est possible qu'un k-mer chevauche parfaitement plusieurs k-mers $\Rightarrow$ Exploration de toutes les extensions possibles avec du backtracking

- Certains *seeds* peuvent être impossibles à relier $\Rightarrow$ Production d'un *read* long synthétique fragmenté en plusieurs parties

Remarques

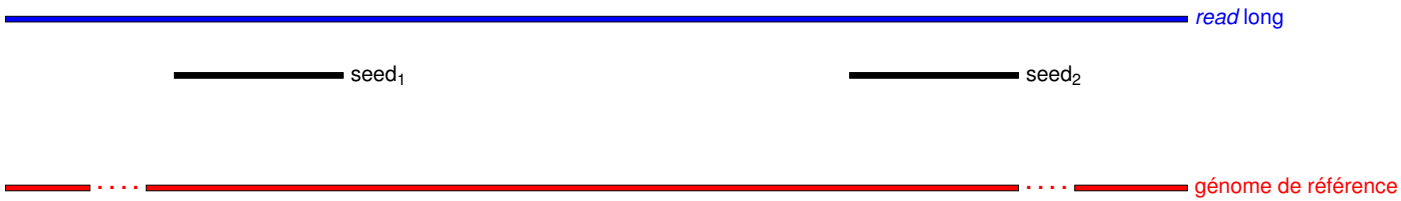
- Il est possible qu'un k-mer chevauche parfaitement plusieurs k-mers $\Rightarrow$ Exploration de toutes les extensions possibles avec du backtracking

- Certains *seeds* peuvent être impossibles à relier $\Rightarrow$ Production d'un *read* long synthétique fragmenté en plusieurs parties

Remarques

- Il est possible qu'un k-mer chevauche parfaitement plusieurs k-mers $\Rightarrow$ Exploration de toutes les extensions possibles avec du backtracking

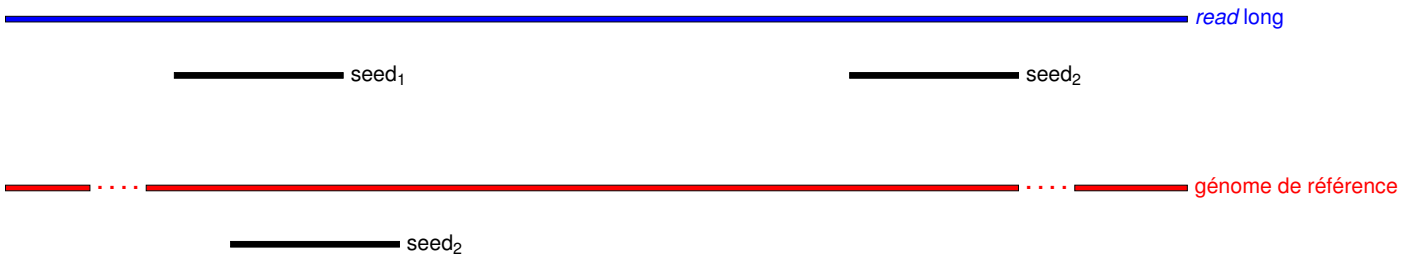
- Certains *seeds* peuvent être impossibles à relier $\Rightarrow$ Production d'un *read* long synthétique fragmenté en plusieurs parties



Remarques

- Il est possible qu'un k-mer chevauche parfaitement plusieurs k-mers $\Rightarrow$ Exploration de toutes les extensions possibles avec du backtracking

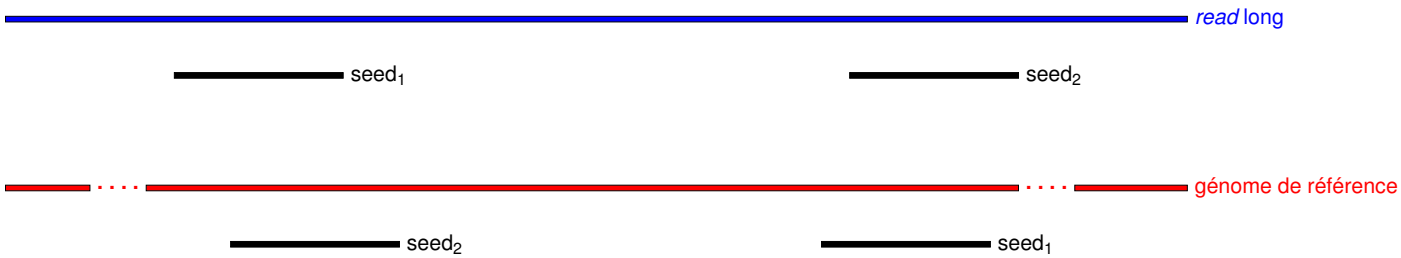
- Certains *seeds* peuvent être impossibles à relier $\Rightarrow$ Production d'un *read* long synthétique fragmenté en plusieurs parties



Remarques

- Il est possible qu'un k-mer chevauche parfaitement plusieurs k-mers $\Rightarrow$ Exploration de toutes les extensions possibles avec du backtracking

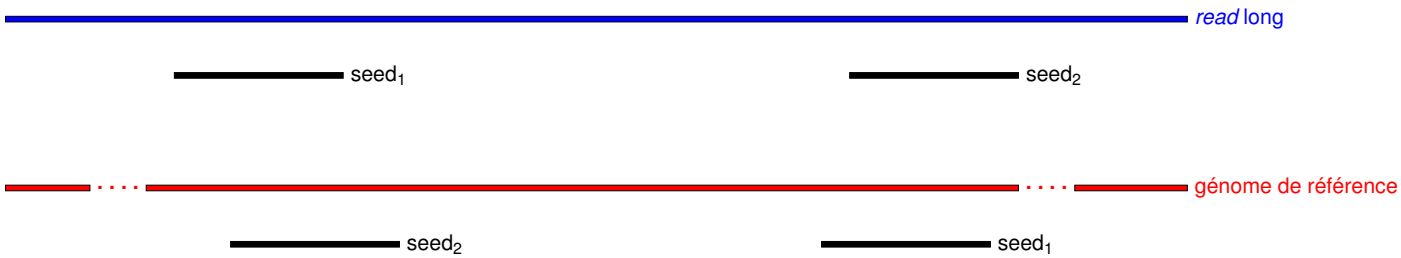
- Certains *seeds* peuvent être impossibles à relier $\Rightarrow$ Production d'un *read* long synthétique fragmenté en plusieurs parties



Remarques

- Il est possible qu'un k-mer chevauche parfaitement plusieurs k-mers $\Rightarrow$ Exploration de toutes les extensions possibles avec du backtracking

- Certains *seeds* peuvent être impossibles à relier $\Rightarrow$ Production d'un *read* long synthétique fragmenté en plusieurs parties



- Lorsqu'un *read* long ne possède qu'un *seed*, on se contente alors d'étendre celui-ci à gauche et à droite

Étape 5

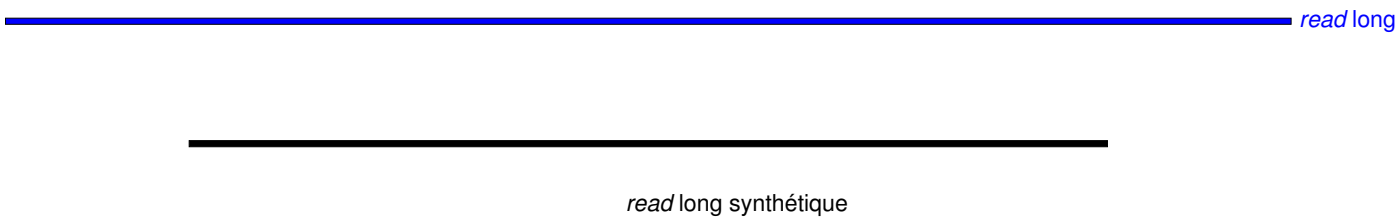
Le *seed* le plus à gauche ne s'aligne pas toujours en position 0 sur le *read* long, et de même, le *seed* le plus à droite n'atteint pas toujours l'extrémité droite du *read* long.

Étape 5

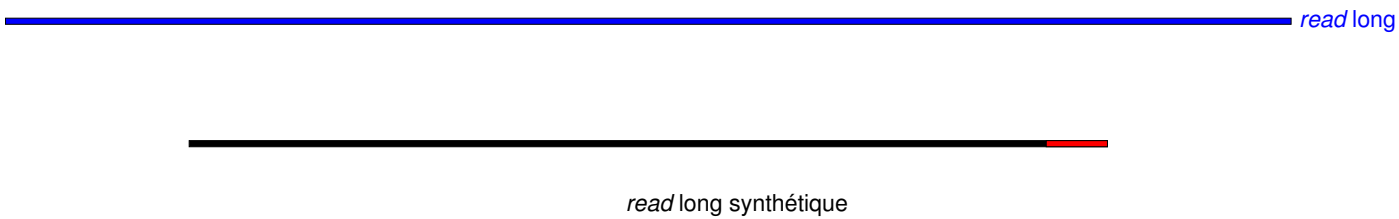
Le *seed* le plus à gauche ne s'aligne pas toujours en position 0 sur le *read* long, et de même, le *seed* le plus à droite n'atteint pas toujours l'extrémité droite du *read* long.

$\Rightarrow$ Une fois tous les *seeds* reliés et le *read* long synthétique produit, on étend, à l'aide de chevauchements parfaits de *k*-mers, son extrémité gauche et son extrémité droite, jusqu'à atteindre les extrémités du *read* long initial, ou une ambiguïté (extension possible à l'aide de plus d'un k-mer)

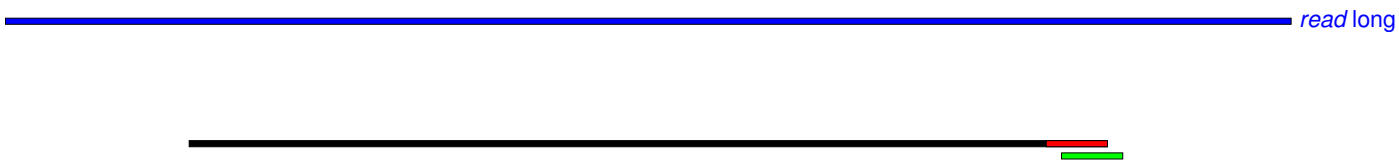
Étape 5, Cas 1 : Pas d'ambiguïté



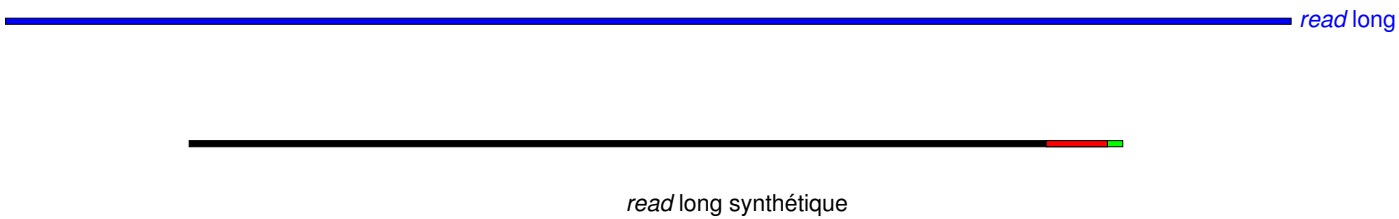
Étape 5, Cas 1 : Pas d'ambiguïté



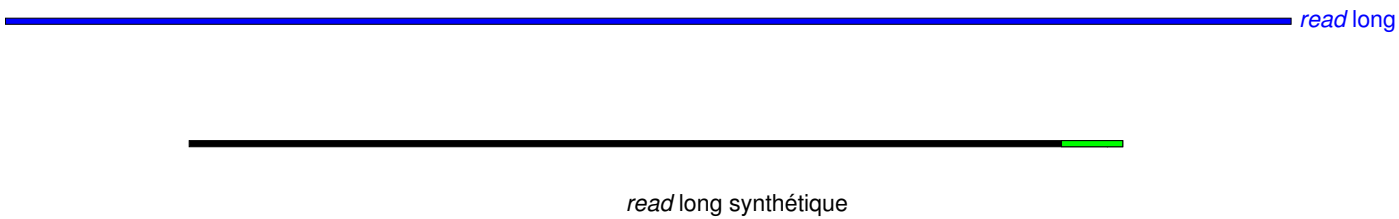
Étape 5, Cas 1 : Pas d'ambiguïté



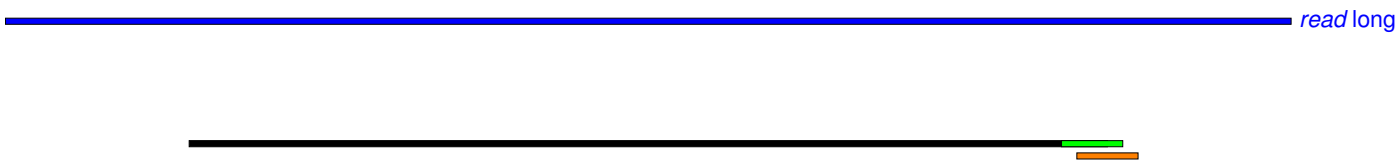
Étape 5, Cas 1 : Pas d'ambiguïté



Étape 5, Cas 1 : Pas d'ambiguïté

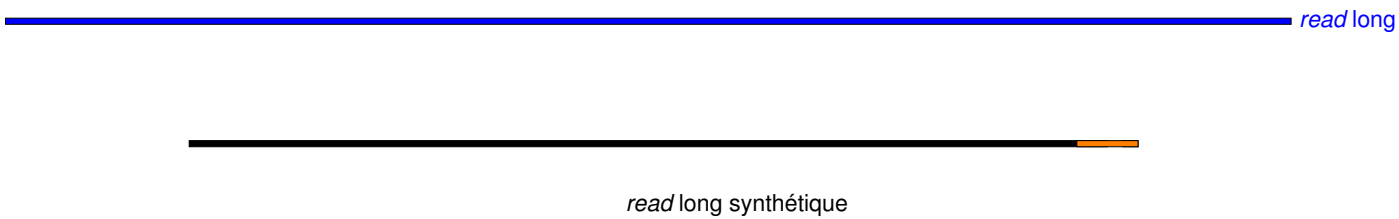


Étape 5, Cas 1 : Pas d'ambiguïté

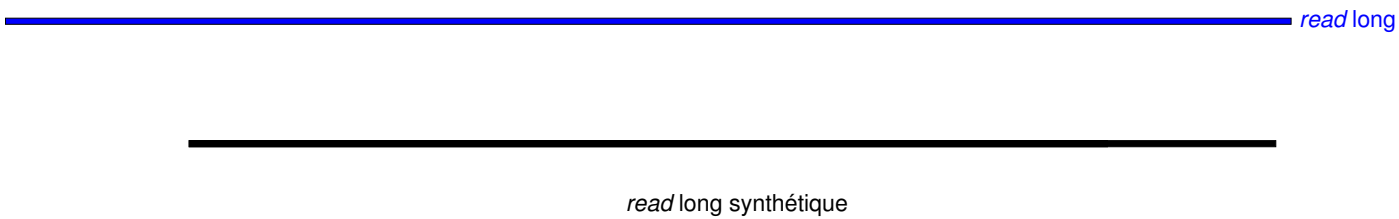


Étape 5, Cas 1 : Pas d'ambiguïté

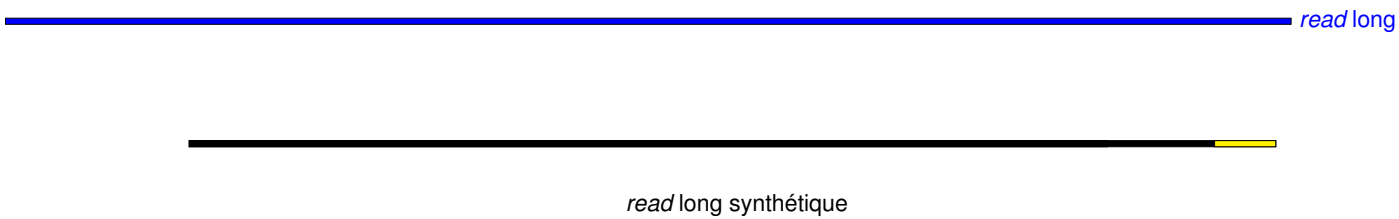
Étape 5, Cas 1 : Pas d'ambiguïté



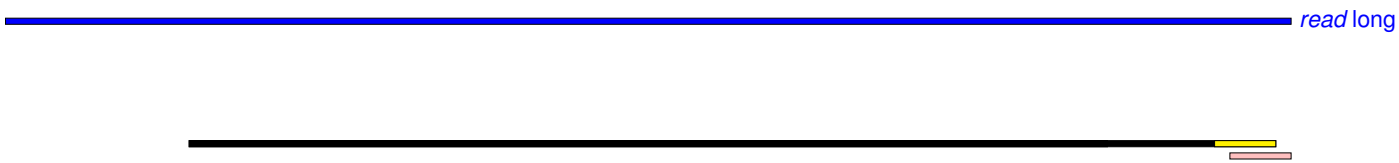
Étape 5, Cas 1 : Pas d'ambiguïté



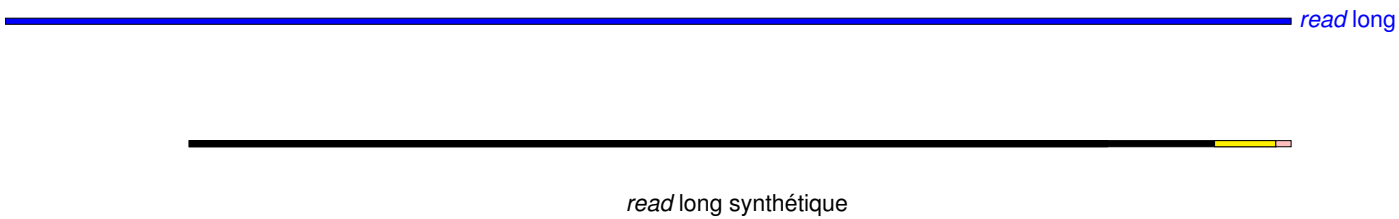
Étape 5, Cas 1 : Pas d'ambiguïté



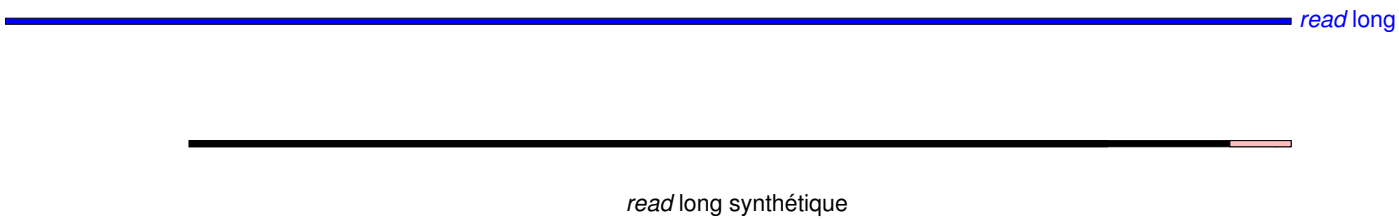
Étape 5, Cas 1 : Pas d'ambiguïté



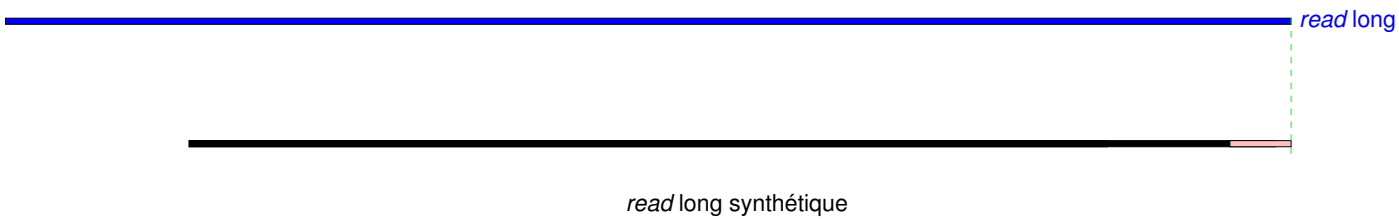
Étape 5, Cas 1 : Pas d'ambiguïté



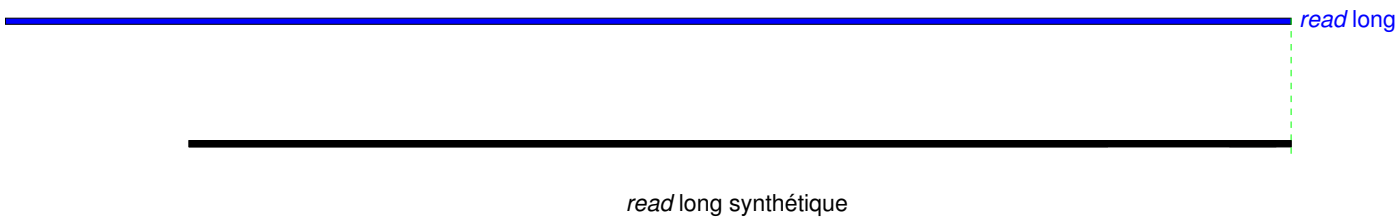
Étape 5, Cas 1 : Pas d'ambiguïté



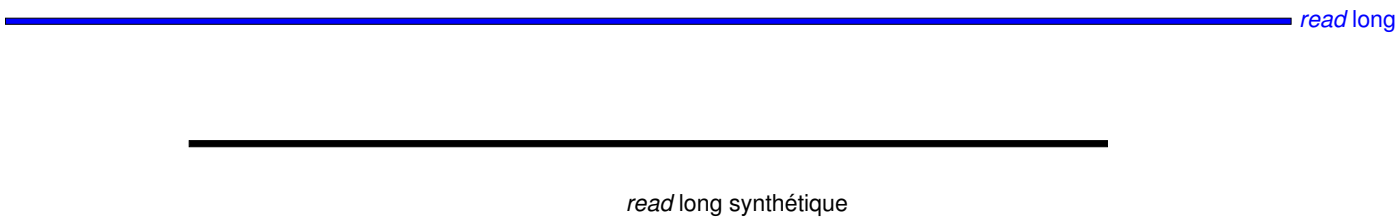
Étape 5, Cas 1 : Pas d'ambiguïté



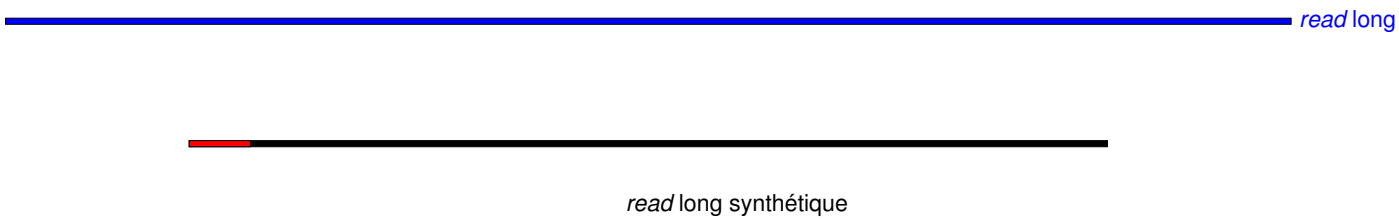
Étape 5, Cas 1 : Pas d'ambiguïté



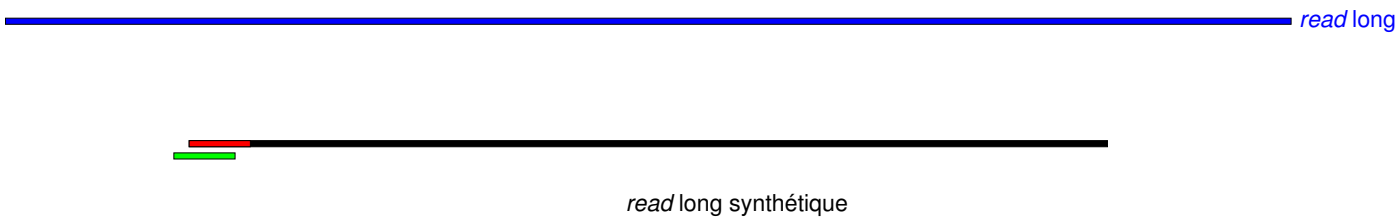
Étape 5, Cas 2 : Présence d'une ambiguïté



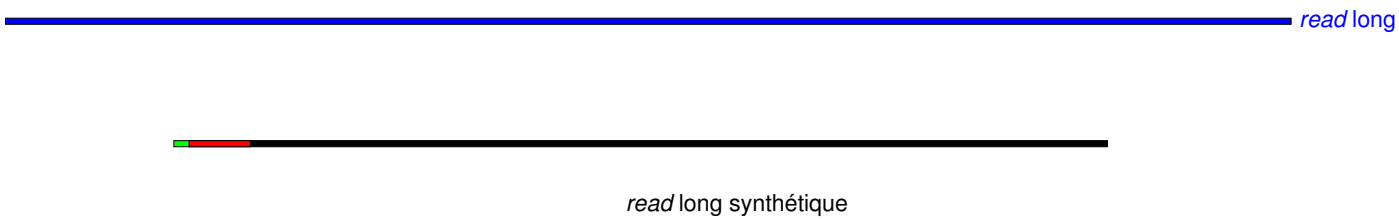
Étape 5, Cas 2 : Présence d'une ambiguïté



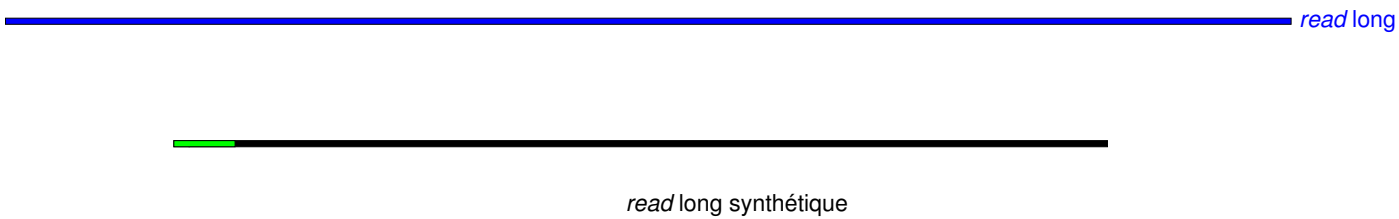
Étape 5, Cas 2 : Présence d'une ambiguïté



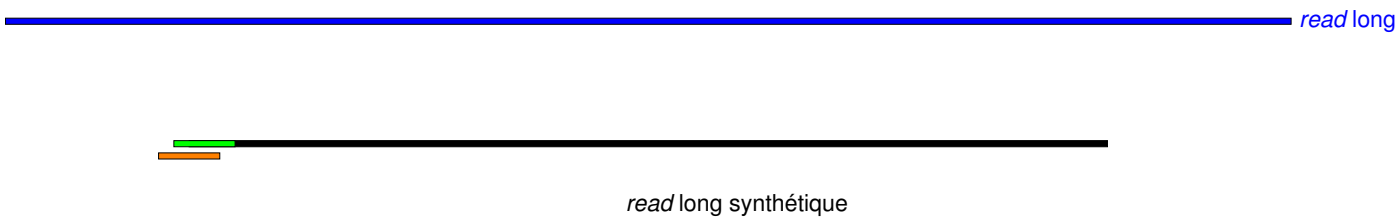
Étape 5, Cas 2 : Présence d'une ambiguïté



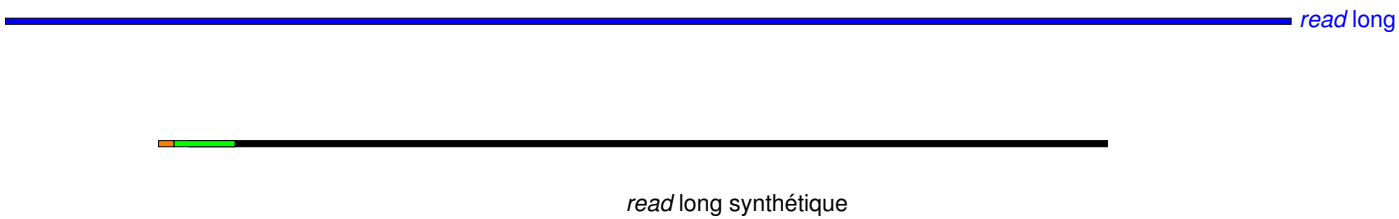
Étape 5, Cas 2 : Présence d'une ambiguïté



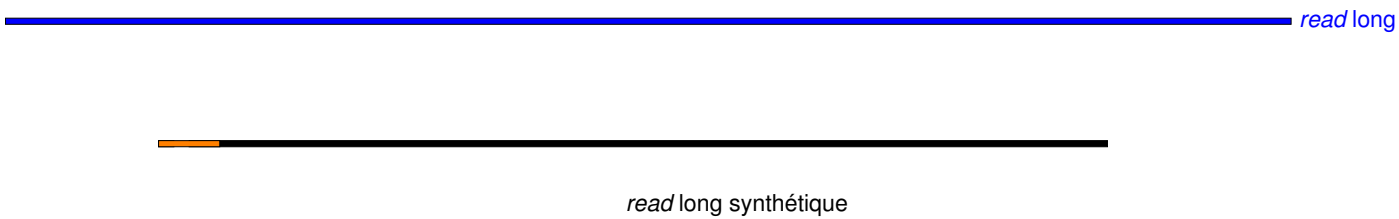
Étape 5, Cas 2 : Présence d'une ambiguïté



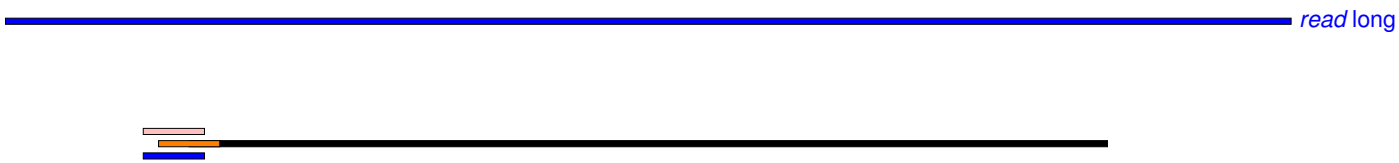
Étape 5, Cas 2 : Présence d'une ambiguïté



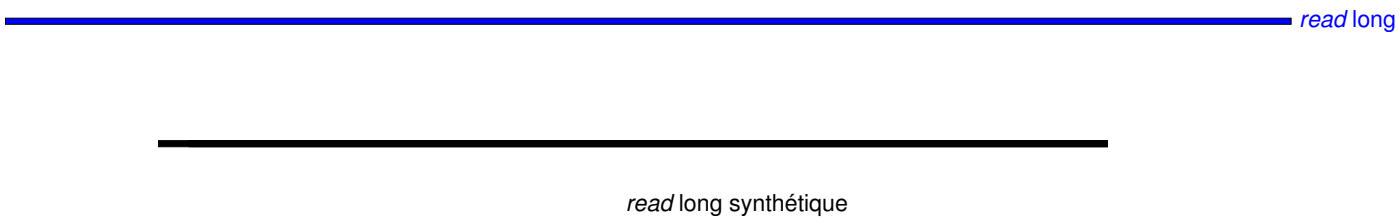
Étape 5, Cas 2 : Présence d'une ambiguïté



Étape 5, Cas 2 : Présence d'une ambiguïté



Étape 5, Cas 2 : Présence d'une ambiguïté



Résultats et comparaison avec NaS : Alignement

Sur les 6 ensembles de *reads* longs ADP1 disponibles sur le site du Génoscope :

	Nombre de reads	Longueur moyenne	Taille totale	Identité moyenne
***reads* longs bruts**	70 314	2 530	177 869 033	3,84 %
NaS (fast)	8 219	4 514	37 099 564	99,92 %
NaS (sensitive)	12 053	6 338	76 388 104	99,89 %
Nous	7 425 (dont 249 fragmentés)	10 250	78 739 767	99,57 %

TABLE – Sur l'ensemble des *reads* 1D

Résultats et comparaison avec NaS : Alignement

Sur les 6 ensembles de *reads* longs ADP1 disponibles sur le site du Génoscope :

	Nombre de reads	Longueur moyenne	Taille totale	Identité moyenne
***reads* longs bruts**	70 314	2 530	177 869 033	3,84 %
NaS (fast)	8 219	4 514	37 099 564	99,92 %
NaS (sensitive)	12 053	6 338	76 388 104	99,89 %
Nous	7 425 (dont 249 fragmentés)	10 250	78 739 767	99,57 %

TABLE – Sur l'ensemble des *reads* 1D

	Nombre de reads	Longueur moyenne	Taille totale	Identité moyenne
***reads* longs brutss**	18 697	10 884	203 496 742	37,07 %
NaS (fast)	15 844	11 084	175 607 625	99,78 %
NaS (sensitive)	16 439	11 871	195 138 674	99,79 %
Nous	15 575 (dont 984 fragmentés)	10 562	178 222 404	99,54 %

TABLE – Sur l'ensemble des *reads* 2D

Résultats et comparaison avec NaS : Assemblage

Avec les 2 ensembles de *reads* longs 1D et 2D préalablement décrits et corrigés :

Outil	Nombre de reads	Nombre de contigs	Couverture du génome	Identité
NaS (fast)	24 063	1	100 %	99,98 %
NaS (sensitive)	28 492	1	100 %	99,99 %
Notre méthode	23 000 (1 233 fragmentés)	1	99,95 %	99,97 %

TABLE – Résultats d'assemblage

Résultats et comparaison avec NaS : Temps et espace

Résultats et comparaison avec NaS : Temps et espace

NaS : 72h avec 8 processus, et production de 100Go de fichiers temporaires

Résultats et comparaison avec NaS : Temps et espace

NaS : 72h avec 8 processus, et production de 100Go de fichiers temporaires

Nous : 35h avec 1 processus, et aucune production de fichiers temporaires, hormis le fichier d'alignement des *reads* courts sur les *reads* longs

Perspectives

- Paralléliser la correction pour accélérer le temps de traitement

Perspectives

- Paralléliser la correction pour accélérer le temps de traitement
- Tester la méthode sur les autres jeux de données disponibles sur le site du Génoscope

Perspectives

- Paralléliser la correction pour accélérer le temps de traitement
- Tester la méthode sur les autres jeux de données disponibles sur le site du Génoscope
- Ajuster les paramètres afin d'obtenir de meilleurs résultats

Méthodes existantes

- LoRDEC (2014, Hybride, DBG de *reads* courts)
- Jabba (2016, Hybride, DBG de *reads* courts)
- LoRMA (2016, Uniquement *reads* longs, DBG de *reads* longs)