1 Algo correction de reads longs MinION

1.1 Résultats sans filtration des contigs de mauvaise qualité

Ensemble	Dimension	Reads	Contigs / *read*	avLen	avId	maxLen	avCov	NaS	Temps (CPU)	Temps (8 cores)	Temps (BLAT)
1	1D	583	1,65	578	88,28	3 704 (90%)	28,58	20			
	2D	411	2,29	1 775	86,96	10 808 (92%)	69,51	87			
2	1D	525	3	670	86,60	2 501 (97%)	27,78	14			
	2D	329	3,40	2 258	86,35	30 177 (90%)	71,97	54			
3	1D	358	2,53	659	86,46	5 513 (93%)	21,57	2			
	2D	1 664	2,96	2 442	86,85	25 68 (92%)	72,75	377			
4	1D	543	3,33	704	86,56	8 138 (91%)	27,76	14			
	2D	949	2,71	3 374	87,08	31 973 (92%)	80,67	325			
5	1D	1 884	3,37	697	86,77	7 539 (94%)	26,73	44			
	2D	2 879	2,26	4 754	88,06	41 396 (91%)	89,43	1 241			

TABLE 1 – Résultats avant filtration (paramètres par défaut, Etape 1 ET Etape 2).

Ensemble	Dimension	Reads	Contigs / *read*	avLen	avId	maxLen	avCov	NaS	Temps (CPU)	Temps (8 cores)	Temps (BLAT)
1	1D	583	1.65	573.51	88.43	3704 (90 %)	28.44	19	1min38	1min09	3min40
	2D	411	2.34	1737.09	87.27	10808 (92 %)	69.14	84	9min45	2min58	7min50
2	1D	525	3.02	650.31	86.79	2501 (97 %)	27.58	13	4min22	1min53	4min28
	2D	329	3.50	2185.34	86.78	30177 (90 %)	71.66	52	13min29	3min40	10min53
3	1D	358	2.54	651.70	86.68	5513 (93 %)	21.41	2	1min21	1min11	3min43
	2D	1664	3.01	2389.46	87.13	25689 (92 %)	72.13	368	323min41	81min23	50min12
4	1D	544	3.35	691.48	86.79	8138 (91 %)	27.52	13	4min57	2min16	7min
	2D	949	2.77	3298.24	87.45	31973 (92 %)	80.45	321	215min41	40min18	37min18
5	1D	1884	3.38	688.09	86.96	7102 (94 %)	26.53	43	53min14	11min27	14min30
	2D	2879	2.30	4665.60	88.27	41396 (91 %)	89.29	1 222	1 917m	293min34	72min38

TABLE 2 – Résultats avant filtration (paramètres par défaut, Etape 1 uniquement).

1.2 Résultats après filtration des contigs de mauvaise qualité

Ensemble	Dimension	Reads	Contigs / *read*	avLen	avId	maxLen	avCov	NaS
1	1D	330	1,27	528	96,34	1 426 (98%)	24,82	10
	2D	195	1,52	2 507	94,61	10 808 (92%)	73	69
2	1D	200	1,425	571	96,33	2 371 (94%)	23,14	9
	2D	129	1,86	3 858	94,18	30 177 (90%)	75,15	42
3	1D	151	1,42	530	96,07	2 063 (92%)	13,58	1
	2D	793	1,74	4 342	94,16	25 689 (92%)	79,99	329
4	1D	202	1,63	633	96,18	8 138 (91%)	21,85	10
	2D	508	1,5	6 472	93,30	28 650 (90%)	90,26	276
5	1D	680	1,6	573	96,21	4 636 (91%)	21,29	29
	2D	1 809	1,5	7 093	92,81	41 396 (91%)	94,75	1 049

TABLE 3 – Résultats après filtration (paramètres par défaut, aucun read n'ayant un contig avec q < 90, Etape 1 ET Etape 2).

Ensemble	Dimension	Reads	Contigs / *read*	avLen	avId	maxLen	avCov	NaS
1	1D	453	1.41	551.55	96.23	3704 (90 %)	23.05	10
	2D	349	1.75	1860.35	94.99	10808 (92 %)	56.31	69
2	1D	433	2.24	616.63	96.09	2501 (97 %)	20.18	9
	2D	301	2.44	2417.28	94.71	30177 (90 %)	55.32	42
3	1D	292	1.97	608.14	96.05	5513 (93 %)	15.05	1
	2D	1518	2.19	2707.54	94.99	25689 (92 %)	60.02	329
4	1D	448	2.52	665.46	96.13	8138 (91 %)	19.55	10
	2D	858	2.05	3845.06	94.47	31973 (92 %)	69.23	279
5	1D	1572	2.49	656.16	96.10	7539 (94 %)	18.97	29
	2D	2590	1.85	5167.42	93.62	41396 (91 %)	80.47	1049

TABLE 4 – Résultats après filtration (paramètres par défaut, tous les contigs avec q > 90, Etape 1 ET Etape 2).

1.3 Comparaison avec NaS

	nbReads	Cumulative size	N50	minSize	maxSize	avgSize	avgIdentity	Temps (s)
Default	5	31 808	7 994	2 512	10 464	9,6	N.A	N.A
NaS (fast)	4	34 869	9 707	4 265	11 971	8 717,25	100	1 080 (sur PC perso)
NaS (sensitive)	5	37 517	9 743	1 982	12 787	7 503,4	N.A	300
Nous (default)	4	27 235	N.A	1 560	7 430	4 539,2	89,83	1,5 + 65 (BLAT)
Nous (-th=0.90)	4	27 371	N.A	1 579	8 208	4 561,8	87	1,5 + 65 (BLAT)

TABLE 5 – Comparaison de performances nous vs NaS sur le jeu exemple de NaS (5 reads longs). NaS = 2 * 598 485 SR de taille 250. Nous = 11 458 125 SR de taille min 250.

2 Fonctionnement CoLoRMap

A récupérer sur la partie Linux

3 Canu

Retravail de Celera, conçu pour les LR bruités, utilise de nouveaux algorithmes d'overlapping et d'assemblage

3 étapes, pouvant être exécutées indépendamment ou en série :

— Correction
— Trimming
— Assemblage

A chaque étape, Canu construit un index des reads, génère un histogramme des k-mers, et construit un index de tous les overlaps.

Correction : Sélectionne les meilleurs overlaps à utiliser pour la correction, estime la longueur des reads corrigés, génère les reads corrigés

Trimming : Détecte les régions non soutenues et trim les reads pour ne garder que les + longues régions soutenues

Assemblage : Dernière passe d'idendification des erreurs, construit un meilleur graphe de chevauchement et sort les contigs, un graphe d'assemblage et des stats résumées

3.1 Calcul des overlaps (MinHash)

Overlapping en 2 étapes :

— Listing des pairs de reads partageant une similarité
— Comparaison plus directe des paires

Les candidats sont trouvés en identifiant les k-mers partagés, mais Canu compare des sketches de reads plutôt que de simples k-mers, car à cause des répétitions, la fréquente

apparition de certains k-mers augmente le nombre de paires à passer à la seconde étape, et car ignorer les k-mers trop répétés entraîne une perte de détection de certains overlaps corrects.

Chaque sketch contient un sous ensemble de taille fixe de k-mers. Pour chaque entrée du sketch, w fonctions de hachage sont appliqués à chaque k-mer du read (w = poids du k-mer) et le k-mer produisant le hash minimum est choisi.

Un k-mer apparaissant souvent dans plusieurs reads aura un poids plus faible, et un k-mer apparaissant plusieurs fois dans un seul read aura un poids plus fort => tf-idf weighting

=> Ainsi, un k-mer de poids plus important sera haché plus de fois et aura plus de chance de faire partie du sketch

Utilisation de sketches pour la comparaison plus directe également (stratégie similaire à Mash), et formule de distance utilisée pour estimer le taux d'erreur des overlaps

3.2 Correction des reads

Canu utilise les infos fournies par les overlaps pour corriger les reads. 2 étapes pour déterminer quels overlaps seront sélectionnés pour corriger chaque read :

— Filtre global, chaque read choisit où il va fournir une potentielle correction
— Filtre local, chaque read accepte ou rejette la correction fournie par les autres reads

Chaque read est autorisé à contribuer à la correction d'au plus C autres reads, afin d'évité les biais dus à la qualité de séquençage et aux répétitions

Pour corriger un read, les reads le chevauchant son alignés dessus en utilisant l'algo ND de Myers. Un graphe acyclique et dirigé est créé à partir des alignements, et le chemin avec le poids le + élevé est suivi afin de générer la séquence servant de correction. Quand il n'y a pas assez de proposition de correction pour la correction d'un read, celui-ci est splitté.

3.3 Trimming

Après la correction, les overlaps sont recalculés pour les reads corrigés, et ceux-ci sont trimmed à la portion la plus longue couverte par des overlaps de taux d'erreur max E, de

longueur min L, sur une profondeur au moins C

Seconde passe de correction pour chercher les erreurs spécifiques à la technologie utilisée

3.4 Assemblage

Canu utilise le module Bogart, qui construit un graphe d'assemblage en utilisant une variante du Best Overlap Graph de Miller.

Les overlaps sont séparés en deux catégories :

— Containment : Toutes les bases d'un read sont alignées sur un autre read

— Dovetail : L'overlap n'implique que les extrémités des deux reads

Un meileur overlap est alors le + long dovetail vers un read donné. Chaque read a donc 2 meilleurs overlaps, un à chaque extrémité

Les meilleurs overlaps sont choisis après plusieurs étapes de filtration permettant de supprimer les overlaps à fort taux d'erreurs, les reads chimériques, et les reads dont les overlaps indiquent une anomalie dans la séquence => Permet une construction de graphe plus propre et plus précise

Un ensemble de meilleurs overlap est utilisé pour calculer un taux d'erreur maximum par overlap, les overlaps ayant un plus haut taux d'erreur ne sont alors pas utilisés pendant la construction du graphe (taux = 2% en général)

Bogart filtre les mauvais overlaps et les reads suspicieux :

— Les reads pas totalement couverts pas des overlaps en dessous du taux d'erreur max sont exclus

— Les meilleurs overlaps doivent être mutuels. Les overlaps non mutuels sont causés par une différence de longueur entre les overlaps => Les reads avec une importante différence sur la taille des overlaps sont exclus

L'ensemble des reads et de meilleurs overlaps restants définit alors le best overlap graph

Les contigs initiaux sont construits à partir du graphe comme avec Miller, et un profil de taux d'erreur est généré pour chaque contig, à partir du taux d'erreur moyen des overlaps impliqués dans sa construction. Ce profil est utilisé pour déterminer si les reads externes ont des overlaps valides avec le contig.

Bogart tente d'inclure dans les contigs les reads inclus ou filtrés précédemment. Les overlaps du contig vers ces reads sont utilisés pour calculer un ensemble de placements potentiels sur le contig, le placement dont l'overlap a le plus faible taux d'erreur est accepté, et le read est placé.

Les reads non placés sont alors marqués comme non assemblés.

4 Jabba

Assemblage des LR en un DBG, et alignement des LR sur le graphe

Approche par pseudo alignement avec seed-and-extend, en utilisant les MEM (maximal exact matches) entre un LR et un noeu de graphe comme seeds

Avantages des MEM :

— Seeds potentiellement plus long, permettent d'avoir une estimation sur comment le LR doit être aligné sur le graphe
— A partir d'un SA étendu, les seeds de longueur arbitraires peuvent être cherchés sans reconstruire l'index, contrairement à l'approche par k-mer où quand différentes valeurs de k sont utilisées, différent indexes doivent être construits
— Permet d'utiliser des valeurs de k arbitraires pour construire le DBG => Avantage car fort taux d'erreur des LR = facteur limitant de la taille min des seeds. => Permet donc d'utiliser des grandes valeurs de k et d'obtenir un DBG moins complexe

Etapes :

1. Assemblage des SR
   — SR corrigés avec Karect, afin de construire un DBG avec un plus grand k
   — Correction du graphe en retirant les bulles, les tips et les connections chimériques pour réduire la complexité du graphe et faciliter l'alignement des LR
2. Alignement des LR sur le DBG
   — Les MEM sont trouvés avec essaMEM et utilisés comme seeds pour l'alignement
   — Chainage des seeds en plusieurs passes sur le LR. Chaque intération considère toutes les régions de LR pas encore alignées.
   Pour chaque région, les seeds les + longs le considérés, et les noeuds sur lesquels la région courante se mappe sont déterminés. On considère alors la liste des seeds entre chaque noeud et cette région, et les seeds sont placés optimalement.
   Les mapping locaux (sur les noeuds) non super maximaux ou couvrant moins qu'une certaine fraction du noeud sont filtrés.
   Une fois les alignements locaux calculés => chainage des alignements entre

différents noeuds en suivant un unique chemin du graphe. Les deux directions sont étendus.

— Alignement final : Une fois toutes les passes exécutées, il y a souvent plusieurs alignements possibles => Celui qui couvre le mieux le LR est choisi.
Pour corriger les extrémités, l'alignement est étendu le long des chemins uniques du graphe. Si LR trop long => trim. Mais cette partie de la correction coûte cher

Résultats :

Accuracy : 99,84 - 99,97
Reads sans erreurs : 83,33 - 98,35 N50 : 4 183 - 15 563
Temps / read : 9 - 100 ms
Mémoire : 103 Mo - 5Go

Runtime Karect : 1,6 - 18,27 h et 9,7 - 60,4 Go mémoire

5 Spaced-seeds [...]

=> Papier orienté résultats

6 Guided Assembly (avec BLAST)

6.1 ADP1

Génome de référence => cumulativeSize = 3 598 621

Fichier de 11 458 125 SR => avLen = 293 ; avQual = xx,xx % ; cumulativeSize = 4 085 888 331 ; génome couvert = 3 598 621 (100 %)

Fichier de 3 080 contigs => avLen = 1 209 ; avQual = 57,97 % ; cumulativeSize = 3 850 503 ; génome couvert = 3 556 713 (98,83 %)

1 765 de ces contigs peuvent produire un scaffolds => avLen = 2 042 ; avQual = 95 % cumulativeSize = 3 604 378 ; génome couvert = 3 556 503 (98,83 %)

6.1.1 Mapping avec PBLAT

Ensemble	Dimension	Seeds	Prefs	Suffs
1	1D	2	1	0
	2D	1	0	0
2	1D	1	1	1
	2D	0	0	3
3	1D	0	0	0
	2D	0	4	6
4	1D	1	1	3
	2D	1	10	1
5	1D	6	6	5
	2D	2	7	4
6	1D	1	1	0
	2D	11	33	26

TABLE 6 – Résultats du mapping des LR sur les SR assemblés avec Minia. Avec 11 969 767 SR => 3 080 contigs, avSize = 1 209.

6.1.2 Mapping avec BLAST

Mapping bien plus efficace en utilisant BLAST : 243 453 hits au total, sur 3 080 contigs (|al| avec hit = 2 042 ; |al| sans hit = 91)

6.1.3 Multiples passages par un même noeud du graphe autorisés

On parcours le graphe, et le parcours se poursuit, même si le noeud suivant a déjà été visité

|al| min	Couples	avLen	avQual	genCov	Scaffolds	avLen	avQual	cumulativeSize	Gen. Couv.
100	257	13 755	99,51 %	98,17 %	192	104 565	63,69 %	20 076 539	78,01 %
90	364	9 737	99,65 %	98,37 %	284	84 180	63,21 %	23 907 228	80,32 %
80	414	8 571	99,66 %	98,44 %	340	79 519	54,60 %	27 036 294	81,49 %
70	445	7 979	99,68 %	98,47 %	373	87 281	54,75 %	32 555 752	81,53 %
60	759	4 703	99,47 %	98,73 %	558	85 592	51,08 %	47 593 066	82,48 %
50	842	4 245	99,38 %	98,76 %	637	72 103	49,31 %	45 929 448	85,15 %
40	929	3 851	99,39 %	98,77 %	699	58 974	53,77 %	41 222 848	84,81 %
30	1 680	2 144	97,41 %	98,83 %	831	54 645	48,09 %	45 409 690	85,53 %
20	1 763	2 044	95,03 %	98,83 %	811	56 605	51,80 %	45 906 856	85,88 %
10	1 763	2 044	95,03 %	98,83 %	811	56 605	51,80 %	45 906 856	85,88 %

TABLE 7 – Stats sur scaffolds produits par notre méthode. |al| max = 500 ; maxGap = 10k

6.1.4 Un seul passage par noeud du graphe autorisé v.1

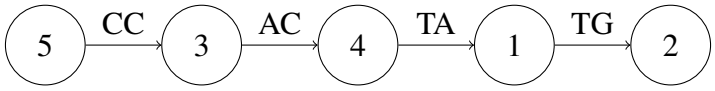
On parcours le graphe, et le parcours s'arrête lorsque le noeud suivant a déjà été visité

|al| min	Couples	avLen	avQual	genCov	Scaffolds	avLen	avQual	cumulativeSize	Gen. Couv
100	257	13 755	99,51 %	98,17 %	52	54 113	62,5 %	2 813 879	72,34 %
90	364	9 737	99,65 %	98,37 %	67	44 862	58,51 %	3 005 757	76,38 %
80	414	8 571	99,66 %	98,44 %	80	37 621	55,46 %	3 009 663	76,22 %
70	445	7 979	99,68 %	98,47 %	85	37 668	57,72 %	3 201 178	79,09 %
60	759	4 703	99,47 %	98,73 %	124	26 952	52,69 %	3 342 100	80,29 %
50	842	4 245	99,38 %	98,76 %	149	22 783	49,85 %	3 394 615	81,92 %
40	929	3 851	99,39 %	98,77 %	161	21 433	48,96 %	3 450 655	81,94 %
30	1 680	2 144	97,41 %	98,83 %	195	18 299	48,77 %	3 568 383	84,26 %
20	1 763	2 044	95,03 %	98,83 %	198	18 215	47,88 %	3 606 563	84,37 %
10	1 763	2 044	95,03 %	98,83 %	198	18 215	47,88 %	3 606 563	84,37 %

TABLE 8 – Stats sur scaffolds produits par notre méthode. |al| max = 500 ; maxGap = 10k

Problème : Ce parcours ne permet pas de produire le plus long scaffold possible.

Exemple :



Puisqu'on parcours le graphe dans l'ordre des noeuds, on va produire : 1.TG.2 ; 3.AC.4
Alors qu'on devrait produire : 5.CC.3.AC.4.TA.1.TG.2

6.1.5 Un seul passage par noeud du graphe autorisé v.2

On parcours le graphe, et lorsque le noeud suivant a déjà été visité, on produit le sacc-fold courant et on raboute le scaffold courant et le scaffold produit par le noeud suivant, si le noeud suivant marque le début d'un scaffold.

|al| min	Couples	avLen	avQual	genCov	Scaffolds	avLen	avQual	cumulativeSize	Gen. Couv
100	257	13 755	99,51 %	98,17 %	55	63 204	61,65 %	3 476 194	74,72 %
90	364	9 737	99,65 %	98,37 %	75	46 942	56,81 %	3 520 627	77,63 %
80	414	8 571	99,66 %	98,44 %	88	38 240	53,91 %	3 365 136	76,71 %
70	445	7 979	99,68 %	98,47 %	94	44 053	55,90 %	4 140 967	79,94 %
60	759	4 703	99,47 %	98,73 %	149	31 747	49,25 %	4 730 310	81,32 %
50	842	4 245	99,38 %	98,76 %	169	26 664	48,51 %	4 506 171	82,90 %
40	929	3 851	99,39 %	98,77 %	189	27 056	47,07 %	5 113 545	83,01 %
30	1 680	2 144	97,41 %	98,83 %	240	21 828	45,18 %	5 238 738	85,31 %
20	1 763	2 044	95,03 %	98,83 %	338	22 171	44,50 %	5 276 812	85,37 %
10	1 763	2 044	95,03 %	98,83 %	338	22 171	44,50 %	5 276 812	85,37 %

TABLE 9 – Stats sur scaffolds produits par notre méthode. |al| max = 500 ; maxGap = 10k

Les scaffolds ainsi obtenus sont légèrement plus nombreux (+ 10-70 avNb) plus longs (+ 100-1 000 avLen) plus précis (+ 1-5% avQual), et permettent une augmentation du taux de couverture du génome de référence (+ 1-4% avCov), par rapport à ceux obtenus sans raboutage.

Les scaffolds ainsi obtenus sont moins nombreux (- 3x avNb), beaucoup plus courts (-40-50k avLen), plus précis (+ 1-5 % avQual) et couvrent moins le génome de référence (-2-5% avCov), par rapport à ceux obtenus sans marquer les noeuds visités.

6.1.6 Un seul passage par noeud du graphe autorisé v.3

On parcours le graphe, et lorsque le noeud suivant a déjà été visité, on raboute le scaffold courant et le scaffold produit par le noeud suivant, si le noeud suivant marque le début d'un scaffold, et on ne produit que le scaffold final

|al| min	Couples	avLen	avQual	genCov	Scaffolds	avLen	avQual	cumulativeSize	Gen. Couv
100	257	13 755	99,51 %	98,17 %	48	60 463	60,92 %	2 902 200	74,70 %
90	364	9 737	99,65 %	98,37 %	64	47 898	56,19 %	3 065 451	77,56 %
80	414	8 571	99,66 %	98,44 %	74	41 008	53,93 %	3 034 592	76,54 %
70	445	7 979	99,68 %	98,47 %	81	41 699	55,86 %	3 377 608	79,99 %
60	759	4 703	99,47 %	98,73 %	117	30 867	50,55 %	3 611 472	81,15 %
50	842	4 245	99,38 %	98,76 %	142	25 190	49,68 %	3 576 912	82,76 %
40	929	3 851	99,39 %	98,77 %	152	24 368	47,61 %	3 703 926	82,97 %
30	1 680	2 144	97,41 %	98,83 %	193	19 963	46,30 %	3 852 842	85,12 %
20	1 763	2 044	95,03 %	98,83 %	198	19 714	44,78 %	3 903 325	85,26 %
10	1 763	2 044	95,03 %	98,83 %	198	19 714	44,78 %	3 903 325	85,26 %

TABLE 10 – Stats sur scaffolds produits par notre méthode. |al| max = 500 ; maxGap = 10k

6.1.7 Variations sur |al| max

Augmenter |al| max n'influe par sur le nombre de couples, mais influe sur le nombre de scaffolds, permet de produire des scaffolds avec une avLen plus grande, et de mieux couvrir le génome de référence, tout en augmentant la qualité des scaffolds produits

|al| min	|al| max	Couples	avLen	avQual	genCov	Scaffolds	avLen	avQual	cumulativeSize	Gen. Couv
1	avLenContig (1 209)	1 764	2 043	95,03 %	98,82 %	831	160 933	52,05 %	133 750 530	95,71 %
1	avLenCouple (2 042)	1 765	2 042	95,01 %	98,83 %	845	170 860	50,16 %	144 376 648	99,52 %
1	5 000	1 765	2 042	95,01 %	98,83 %	857	213 305	46,53 %	181 945 291	99,56 %
1	10 000	1 765	2 042	95,01 %	98,83 %	856	147 227	55,56 %	126 026 422	99,62 %
1	max	1 765	2 042	95,01 %	98,83 %	857	148 311	59,36 %	127 102 761	99,31 %
100	avLenContig (1 209)	257	13 755	99,51 %	98,17 %	222	121 258	59,98 %	26 919 357	88,21 %
100	avLenCouple (2 042)	257	13 755	99,51 %	98,17 %	231	159 476	56,00 %	36 838 853	95,98 %
100	5 000	257	13 755	99,51 %	98,17 %	234	182 313	55,03 %	42 661 201	98,06 %
100	10 000	257	13 755	99,51 %	98,17 %	230	112 656	62,15 %	25 910 953	98,22 %
100	max	257	13 755	99,51 %	98,17 %	231	140 815	57,48 %	32 528 332	98,02 %

TABLE 11 – Stats sur scaffolds produits par notre méthode. Multiples passages par un même noeud du graphe autorisés ; maxGap = 10k

|al| min	|al| max	Couples	avLen	avQual	genCov	Scaffolds	avLen	avQual	cumulativeSize	Gen. Couv
1	avLenContig (1 209)	1 764	2 043	95,03 %	98,82 %	190	20 744	47,25 %	3 941 413	94,18 %
1	avLenCouple (2 042)	1 765	2 042	95,01 %	98,83 %	187	21 870	45,44 %	4 089 602	98,20 %
1	5 000	1 765	2 042	95,01 %	98,83 %	186	21 574	46,76 %	4 012 750	97,49 %
1	10 000	1 765	2 042	95,01 %	98,83 %	186	20 557	48,08 %	3 823 543	96,92 %
1	max	1 765	2 042	95,01 %	98,83 %	184	20 665	45,20 %	3 802 391	97,65 %
100	avLenContig (1 209)	257	13 755	99,51 %	98,17 %	50	65 020	61,38 %	3 250 995	82,79 %
100	avLenCouple (2 042)	257	13 755	99,51 %	98,17 %	56	65 182	63,11 %	3 650 204	93,32 %
100	5 000	257	13 755	99,51 %	98,17 %	55	69 376	66,09 %	3 815 655	97,01 %
100	10 000	257	13 755	99,51 %	98,17 %	58	61 913	66,38 %	3 590 968	93,64 %
100	max	257	13 755	99,51 %	98,17 %	52	68 707	64,69 %	3 572 769	94,79 %

TABLE 12 – Stats sur scaffolds produits par notre méthode. Un seul passage par noeud du graphe, et arrêt quand noeud suivant visité ; maxGap = 10k

|al| min	|al| max	Couples	avLen	avQual	genCov	Scaffolds	avLen	avQual	cumulativeSize	Gen. Couv
1	avLenContig (1 209)	1 764	2 043	95,03 %	98,82 %	238	23 970	46,75 %	5 704 989	95,16 %
1	avLenCouple (2 042)	1 765	2 042	95,01 %	98,83 %	237	24 610	44,31 %	5 832 456	98,92 %
1	5 000	1 765	2 042	95,01 %	98,83 %	241	27 146	47,28 %	6 542 261	98,03 %
1	10 000	1 765	2 042	95,01 %	98,83 %	248	27 104	44,96 %	6 721 871	97,78 %
1	max	1 765	2 042	95,01 %	98,83 %	241	30 906	44,88 %	7 448 369	98,52 %
100	avLenContig (1 209)	257	13 755	99,51 %	98,17 %	56	68 512	58,64 %	3 836 683	83,88 %
100	avLenCouple (2 042)	257	13 755	99,51 %	98,17 %	62	75 597	60,81 %	4 709 357	94,64 %
100	5 000	257	13 755	99,51 %	98,17 %	63	84 587	61,27 %	5 328 960	97,61 %
100	10 000	257	13 755	99,51 %	98,17 %	65	73 381	61,06 %	4 769 744	96,26 %
100	max	257	13 755	99,51 %	98,17 %	60	86 467	63,13 %	5 188 010	97,15 %

TABLE 13 – Stats sur scaffolds produits par notre méthode. Un seul passage par noeud du graphe, et raboutage v.1 ; maxGap = 10k

|al| min	|al| max	Couples	avLen	avQual	genCov	Scaffolds	avLen	avQual	cumulativeSize	Gen. Couv
1	avLenContig (1 209)	1 764	2 043	95,03 %	98,82 %	189	23 356	46,32 %	4 414 243	95,15 %
1	avLenCouple (2 042)	1 765	2 042	95,01 %	98,83 %	188	24 575	45,01 %	4 620 159	98,91 %
1	5 000	1 765	2 042	95,01 %	98,83 %	191	24 693	47,26 %	4 716 336	98,01 %
1	10 000	1 765	2 042	95,01 %	98,83 %	192	24 198	45,42 %	4 645 923	97,72 %
1	max	1 765	2 042	95,01 %	98,83 %	192	26 167	44,10 %	5 023 971	98,44 %
100	avLenContig (1 209)	257	13 755	99,51 %	98,17 %	50	65 909	59,34 %	3 295 426	83,76 %
100	avLenCouple (2 042)	257	13 755	99,51 %	98,17 %	52	73 615	59,58 %	3 828 002	94,63 %
100	5 000	257	13 755	99,51 %	98,17 %	48	80 330	60,48 %	3 855 835	97,60 %
100	10 000	257	13 755	99,51 %	98,17 %	54	69 626	62,32 %	3 759 803	96,24 %
100	max	257	13 755	99,51 %	98,17 %	49	79 159	63,57 %	3 878 815	97,11 %

TABLE 14 – Stats sur scaffolds produits par notre méthode. Un seul passage par noeud du graphe, et raboutage v.2 ; maxGap = 10k

6.1.8 Variations sur maxGap

Diminuer maxGap augmente la qualité moyenne des scaffolds, mais diminue leur longueur moyenne et la couverture du génome de référence.

Augmenter maxGap diminue la qualité moyenne de scaffolds, mais augmente (légèrement) leur longueur moyenne et la couverture du génome de référence.

6.1.9 Overlaps

Autoriser des overlaps permet de générer plus de scaffolds, de qualité plus élevée, de couvrir d'avantage le génome de référence mais entraîne une baisse de leur longueur moyenne => Autoriser un overlap peut donc être intéressant, car il permet d'effectivement lier de plus nombreuses paires de contigs.

6.1.10 Distance entre les contigs liés par un LR

Pour maxGap = 10k, |al| min = 1-100 : moyenne = 500k => D'où la perte de qualité et de couverture lors de la production des scaffolds
=> Les contigs liés par un LR sont trop distants pour que le scaffold produit se mappe convenablement, et couvre toutes les zones où se mappent initialement les contigs => Certaines zones ne seront pas couvertes

La distance moyenne diminue quand on augmente |al| min (en moyenne 30k pour |al| min 1k) => D'où l'augmentation de qualité à mesure que |al| min augmente, car les contigs des paires se rapprochent
=> Mais la couverture diminue aussi, car bcp de contigs passent à la trappe

6.1.11 Distance entre les mappings des contigs et des scaffolds

Pour maxGap = 10k, |al| min 1-100 : Distance moyenne entre pos scaffold et pos contig le plus à gauche = 375k => D'où la perte de couverture, les scaffolds produits se mappent à des endroits "aléatoires
"

La distance diminue en augmentant |al| min (en moyenne 40k pour |al| min 1k) => D'où l'augmentation de la qualité à mesure que |al| min augmente, car les scaffolds se mappent plus près de l'endroit où ils devraient
=> Mais couverture diminue

6.1.12 Production des contigs non traversés

Ajouter les contigs non traversés lors du parcours du graphe dans le fichier résultat permet d'augmenter la couverture (0,4 - 1,25 % en plus, dans le cas ou |al| min = 1-100)

=> Trop de contigs lorsque |al| min est trop bas ; semble intéressant avec |al| min >= 100

Peut-être possible d'atteindre 100 % comme ça

6.1.13 Assemblage avec Abyss

Utilisé dans le papier de Karlsson sur le scaffoldin de LR, et semble produire de bons résultats

=> 355 contigs, avLen = 10 233, génome couvert = 3 595 706 (99,92 %)

Présence de très petits contigs dans l'assemblage produit (len = 30-50), après filtration :

=> 34 contigs, avLen = 106 339, avQual = 95,97 %, génome couvert = 3 588 376 (99,72 %)

=> Plus long à l'exécution que Minia, mais semble produire des résultats plus satisfaisants

Exécution de notre algorithme sur les contigs abyss :

Production de 12 contigs, avLen = 238 716, avQual = 74,83, cov = 78,93 (param : maxGap 10K, |al| = [100...max]
En sortant aussi les contigs non traversés : 15 contigs, avLen = 214 026, avQual = 75,8, cov = 88,32

Production de 12 contigs, avLen = 261 438, avQual = 74,17, cov = 86,32 (param : maxGap 10K, |al| = [1...max]
En sortant aussi les contigs non traversés : 15 contigs, avLen = 214 427, avQual = 75,3, cov = 88,31

Distance moyenne entre les contigs liés : 200-800k, ne diminue pas plus même en augmentant |al| min => Même problème, contigs liés trop distants pour que le scaffold se mappe convenablement => D'où la perte de qualité malgré le fait que les chemins du graphe soient plus courts qu'avec Minia, et que moins de bases de LR soient donc introduites

6.1.14 Gap min

Fixer un gapMin :

— Fait beaucoup baisser la couverture, même si gapMin est petit (Perte de 5-10 % avec gapMin = 100)

— Augmente la qualité moyenne quand gapMin est grand et |al| min <= 90 (+ 3-10 % qual avec gapMin = 1000) ; au delà de |al| min = 100 ; qualité comparable aux résultats sans gapMin

— Augmente la longueur moyenne quand |al| min est petit et diminue quand |al| min est plus grand (seuil limite autour de |al| min = 60 - 70)

6.1.15 Consensus

Nombre de LR différents permettant de lier 2 contigs :

— 2,19 (gapMax = 1K, |al| = [1...max])

— 2,14 (gapMax = 10K, |al| = [1...max])

— 6,42 (gapMax = 10K, |al| = [100...max])

— 7,43 (gapMax = 10K, |al| = [1000...max])

— 1,56 (gapMin = 100, gapMax = 10K, |al| = [1...max])

— 3,94 (gapMin = 100, gapMax = 10K, |al| = [100...max])

— 1,24 (gapMin = 1000, gapMax = 10K, |al| = [1...max])

— 1,60 (gapMin = 1000, gapMax = 10K, |al| = [100...max])

— 5,55 (gapMax = 10K, |al| = [1...max], overlap = 150)

— 7,00 (gapMax = 10K, al = [100...max], overlap = 150)

Fixer un gapMin diminue également le nombre moyen de LR permettant de lier 2 contigs

Consensus possible à partir de |al| min = 100 => On fait le consensus de toutes les portions de LR permettant de combler le gap entre 2 contigs en fixant la taille de chacune de ces portions à la taille du plus grand gap

Résultats avec consensus :

Minia : En général, qualité et couverture légèrement plus faibles que sans consensus

Abyss : Moins bonne qualité (- 5 %), couverture plus importante (+ 0,5 %) pour |al| min = 1 ; Moins bonne qualité (- 3 %), couverture plus importante (+ 8 %) pour |al| min = 100 => Impossible d'atteindre 100 % de couverture, même en sortant les contigs non traversés pendant le parcours du graphe ; max = 88,15 %

=> Consensus semble donc peu intéressant pour ADP1

6.1.16 Canu

Produit 2 contigs ; cov = 99,89 %, id = 96,33 %, en 6h

6.1.17 Résultats satifsaisants

— Le mapping LR sur contigs permets de différencier les bons des mauvais contigs (les bons ont un mapping de LR) et donc d'obtenir les contigs de meilleure qualité

— Les contigs reliés par un même LR sont correctement ordonnés. Càd le contig à gauche de la paire liée par un LR se trouve bien à gauche du contig droit de la paire lorsque les contigs sont mappés sur le génome de référence (dans 90 % des cas)

6.2 Jeu de données (Ecoli)

Génome de référence => cumulativeSize = 4 641 652

Fichier de 78 295 771 SR
Subsample de 12M ; avLen = 299 ; avQual = 96,62 % ; cumulativeSize = 3 593 112 347 ; génome couvert = 4 641 652 (100 %)

Jeu RL 1 : MAP005
44 540 RL, avLen = 5 560, très mauvais qualité, beaucoup de reads qui ne se mappent pas, avQual < 1 %
Canu ne réussit pas à run sur cet ensemble de RL
=> Inutilisable

Jeu RL 2 : MAP006
25 483 RL, avLen = 9 754, avQual = 70 %

6.2.1 Assemblage Minia

Fichier de 7 875 contigs => avLen = 615 ; avQual = 87,93 % ; cumulativeSize = 4 849 020 ; génome couvert = 4 579 871 (98,67 %)

7 427 de ces contigs peuvent produire un scaffold

Résultats obtenus, sans consensus :

Production de 786 contigs, avLen = 12 056, avQual = 55,13, cov = 97,94 (param : maxGap 10K, |al| = [1...max]
Production de 154 contigs, avLen = 30 501, avQual = 58,52, cov = 94,62 (param : maxGap 10K, |al| = [100...max]

Comme pour ADP, avec consensus => qualité et couverture en général légèrement plus faible que sans consensus

Résultats obtenus, comparés à ADP :

Sans consensus : Meilleure qualité quand |al| min petit (+ 11 % avec |al| min = 1), moins bonne quand |al| min grand (- 4,5 % avec |al| min = 100) ; couverture plus faible (- 0,5-2,5 %)

La qualité des LR ne semble pas vraiment influer sur les résultats produits, qui semblent assez aléatoires

=> Consensus donc peu intéressant

6.2.2 Assemblage Abyss

=> Fichier de 366 contigs => avLen = 12 675, avQual = 97,66 %, cov = 99,14 %

Présence de petits contigs dans l'assemblage produit après filtration :

=> 106 contigs, avLen = 43 449, avQual = 99,40 %, cov = 98,71 %

Résultats obtenus par la GA :

Production de 24 contigs, avLen = 171 772, avQual = 68,54 %, cov = 86,41 % |al| = [1...max]
Production de 26 contigs, avLen = 153 248, avQual = 71,65, cov = 82,11 % |al| = [100...max]

6.2.3 Canu

Run en 1h28, produit 1 contig, s'aligne sur 99,82 % du génome, identité = 99,32 %

6.3 Utilisation des LR "corrigés" par notre méthode

On corrige les LR avec la méthode mise en place pendant le stage pour en isoler les parties de bonne qualité, et on aligne ces parties sur les contigs, plutôt que d'aligner la totalité des LR.

6.3.1 Sur ADP1

33 830 portions "corrigées", avLen = 2 778, avQual = 98,53 %, cumulativeSize = 93 966 735, génome couvert = 3 598 621 (100 %)

Avec Minia Gain de qualité (+ 16 % pour |al min = 100 => 79 % ; + 13 % pour |al| min = 1 => 57 %), mais perte de couverture (- 7 % pour |al| min = 100 => 90,09 % ; - 7 % pour |al| min = 1 => 91,18 %)

Avec Abyss : Perte de qualité (- 0,1 % pour |al| min = 100 => 74,7 % ; - 2,3 % pour |al| min = 1 => 71,8 %), mais gain de couverture (+ 1,47 % pour |al| min = 100 => 80,40 % ; + 2,95 % pour |al| min = 1 => 89,27 %)

6.3.2 Sur Ecoli

=> Beaucoup trop long d'aligner les SR sur les contigs avec PBLAT => Moins efficace que Canu en terme de temps => Pas intéressant

6.4 Utilisation des LR corrigés par Canu

On corrige l'ensemble de LR avec Canu, et on aligne les LR corrigés sur les contigs.

6.4.1 Sur ADP1

xxxx reads "corrigées", avLen = xxx, avQual = xx, cumulativeSize = xx, génome couvert = (xx %)

Avec Minia : Gain de qualité (+ 9 % pour |al min = 100 => 72 % ; + 21 % % pour |al| min = 1 => 65 %), mais perte de couverture (- 14,7 % pour |al| min = 100 => 82,4 % ; - 16 % pour |al| min = 1 => 82,5 %)

Avec Abyss : Gain de qualité (+ 10 % pour |al| min = 100 => 85 % ; + 9 % pour |al| min = 1 => 83 %), mais perte de couverture (- 16 % pour |al| min = 100 => 62.9 % ; - 29,5 % pour |al| min = 1 => 56.8 %)

6.4.2 Sur Ecoli

xxxx portions "corrigées", avLen = xxx, avQual = xx, cumulativeSize = xx, génome couvert = (xx %)

Avec Minia : Gain de qualité (+ 12 % pour |al min = 100 => 70,3 % ; + 20,6 % % pour |al| min = 1 => 75,6 %), mais perte de couverture (- 9,2 % pour |al| min = 100 => 85,44 % ; - 29,5 % pour |al| min = 1 => 73,07 %)

6.5 Avec les reverse-complement des contigs, LR bruts

Sur ADP1, avec assemblage Abyss :

Production de 19 contigs, avLen = 159 269, avQual = 66,79 %, cov = 76,12 % (param : maxGap 10K, |al| = [100...max] => Pire que les résultats sans RC

Sans RC : Production de 12 contigs, avLen = 238 716, avQual = 74,83, cov = 78,93 (param : maxGap 10K, |al| = [100...max]

Sur Ecoli, avec assemblage Abyss :

=> Similaire aux résultats sans RC

6.6 Avec les reverse-complement des contigs et des LR, LR bruts

Sur ADP1, avec assemblage Abyss :

Production de 23 contigs, avLen = 159 989, avQual = 59,21 %, cov = 90,36 %
=> Pas intéressant

6.7 Filtration des alignements de faible idendité

Sur ADP1 :

Avec Minia => Augmentation de la qualité mais diminution de la couverture

Avec Abyss => Idem

Sur Ecoli => Idem

7 Extraction des parties correctes des LR, puis assemblage

7.1 Alignements des SR sur les LR

7.1.1 Ecoli

Aligneur	Temps index	Temps 1M	Temps 12M p-e	Temps 12M s-e	Options
PBLAT	N.A.	69 min 41 (94,97 %)	N.A.	N.A.	8 threads
BLAST	N.A.	233 min 51 (98,63 %)	N.A.	N.A.	N.A.
Bowtie	10 min	1 min (0,03 %)	12 min 35 (0,00 %)	12 min 54 (0,03 %) 8 threads	
Bowtie2	10 min	5 min 12 (100 %)	1 h 23 min (74,77 %)	1 h 36 min (91,32 %)	8 threads, –very-fast
SOAP2	5 min	4 min 19 (0,03 %)	N.A.	51 min 04 (0,25 %)	N.A.

7.2 ADP1, avec algo correction, assemblage Minia

33 830 portions "corrigées", avLen = 2 778, avQual = 98,53 %, cumulativeSize = 93 966 735, génome couvert = 3 598 621 (100 %)

7.2.1 Régions correctes seules

858 contigs, avLen = 4 155, avQual = 98,5 %, génome couvert = 98,51 %

En filtrant les petits contigs (< 300) : 272 contigs, avLen = 12 956, avQual = 99,5 %, génome couvert = 97,86 %

7.2.2 SR inutilisés

=> 1 105 contigs, avLen = 3 283, avQual = 70,88, cov = 98,40 %

7.2.3 Contigs régions correctes LR + SR inutilisés

1 030 contigs, avLen = 3 527, avQual = 69,80 %

En filtrant les petits contigs (< 300) : 210 contigs, avLen = 16 864, avQual = 93,59 %, cov = 97,91 %

=> Moins de contigs, gain longueur, perte qualité, petit gain couverture

7.2.4 Régions correctes + SR inutilisés

1 045 contigs, avLen = 3 481, avQual = 67,15 %
En filtrant les petits contigs (< 300) : 139 contigs, avLen = 25 478, avQual = 90,49 %, cov = 97,93 %

=> Moins de contigs, gain longueur, perte qualité, petit gain couverture

7.2.5 Contigs régions correctes + contigs SR inutlisés

861 contigs, avLen = 4 122, avQual = 98,16 %, cov = 98,23 %
En filtrant les petits contigs (< 300) : 445 contigs, avLen = 7 905, avQual = 99,69 %, cov = 97,66 %
=> Plus de contigs, perte longueur, petit gain qualité, petite perte couverture

7.2.6 Régions correctes + contigs SR inutilisés

747 contigs, avLen = 4 775, avQual = 96,54 %, cov = 98,59 %

En filtrant les petits contigs (< 300) : 149 contigs, avLen = 23 670, avQual = 99,18, cov = 97,96 %

=> Moins de contigs, gain longueur, petite perte qualité, petit gain couverture

7.2.7 ADP1, avec algo correction, assemblage Abyss, régions correctes seules

4 406 contigs, avLen = 806

En filtrant les petits contigs (< 300) : 2 753 contigs, avLen = 1 171

7.3 ADP1, juste avec filtration, Bowtie 12M SR s-e

=> 166 contigs, avLen = 97, avQual = 96,73 %

7.4 Ecoli, filtrage des parties correctes

7.4.1 Bowtie2, 1M SR

15-20 min => 95 945 portions correctes, avLen = 463, avQual = 93,43 %

6 828 contigs, avLen = 206, avQual = 99,59 %, génome couvert = 29,51 %

7.4.2 Bowtie2, 12M SR p-e

59 min => 243 311 portions correctes, avLen = 671

216 858 contigs, avLen = 75

7.4.3 Bowtie2, 12M SR s-e

59 min => 95 753, avLen = 450

7 038 contigs, avLen = 204

7.5 Ajout des contigs LR et contigs SR dans un même fichier puis assemblage

=> Pas intéressant, pour ADP1 / Minia : 813 contigs, avLen = 4 381, avQual = 96,36 %, génome couvert = 98,49 %

7.6 Ajout des régions correctes LR et contigs SR dans un même fichier puis assemblage

=> Pas intéressant, pour ADP1 / Minia : 803 contigs, avLen = 4 450, avQual = 95,00 %, génome couvert = 98,56 %

7.7 Ajout des contigs LR et SR dans un même fichier puis assemblage

=> Pas intéressant, pour ADP1 / Minia : 3 084 contigs, avLen = 1 208

7.8 Ajout des régions correctes LR et SR dans un même fichier puis assemblage

=> Pas intéressant, pour ADP1 / Minia : 4 257 contigs, avLen = 890

8 Comparaison Abyss / Abyss 2.0.2

2.0.2 plus efficace, produit moins de contigs, globalement plus longs => Mais une fois contigs courts filtrés, résultats similaires, 2.0.2 couvre seulement 0,2 % du génome en plus (pour ADP1)

9 Scaffolding / Gap Filling

=> Lecture SSPACE-LongRead : Même idée que nous, scaffolder les contigs en alignant des LR dessus, mais rempli les gaps avec des N plutôt qu'avec les bases des LR ayant servi à lier deux contigs.

Nous :

Ajouter les RC des LR et des contigs => Permet d'obtenir plus de mappings mais est inutile, car BLAST mappe quer -> ref dans les deux sens, et va donc juste produire des mappings en doublon

Avec BLAST : Si alBeg > alEnd => L'alignement se fait avec le reverse complement

Poids d'une arrête de c1 vers c2 : Nombre de LR permettant de relier c1 à c2

=> Nombre de bons / mauvais liens équivalents

Ne prendre en compte que les alignements préfixes / suffixes des LR sur les contigs permet de bien ordonner, mais les contigs ne peuvent être groupés que par deux, et le successeur d'un contig est généralement trop éloigné sur le génome de ref et n'est pas le vrai successeur direct => Trop peu de liens pour conclure qqch

Compter le nombre de mapping sur contig / RC(contig) afin de déterminer l'orientation, avec résultats BLAST
=> En comptant le nombre de mappings forward / RC de chaque contig dans le fichier BLAST : Pas concluant aussi bien pour ADP que Ecoli
=> En comptant le nombre de contigs liés à gauche et à droite de chaque contig dans le graphe : Pas concluant aussi bien pour ADP que Ecoli

=> Téléchargement de BLASR, comme dans SSPACE-Longread, car fait pour aligner des reads longs avec indels comme erreurs principales

Poids d'une arrête de c1 vers c2 : Nombre de LR + longueurs et scores alignements c1 + longueurs et scores alignements c2 => À partir de là, possible de scaffolder comme SSPACE mais en remplaçant les gaps par les bases des LR

10 Tests SSPACE-Longread

10.1 Assemblage Abyss, paramètres par défaut

10.1.1 ADP1

89 011 MinION

30 contigs => 14 scaffolds

10.1.2 Ecoli

101 212 MinION

366 contigs => 169 scaffolds

10.2 Assemblage Abyss, paramètres selon papier Karlsson + adaptés

alMin = 3000, maxOverlap = 5000, minIdentity = 70

10.2.1 ADP1

89 011 MinION

30 contigs => 1 scaffold, qual = 99,98, cov = 99,92, 35 matches

10.2.2 Ecoli

101 212 MinION

366 contigs => 284 scaffolds

Résultats, en filtrant les contigs < 1000 et en ne gardant que le scaffold long unique :

19 scaffolds dont 1 long : qual = 99,97, cov = 98,68, 167 matches

25 483 MinION => 19 scaffolds dont 1 long : qual = 99,97, cov = 98,68, 169 matches

En fait, filtrer les courts contigs n'est probablement pas utiles, car les scaffolds supplémentaires sont constitués chacun d'un seul contig de logueur inférieure à alMin

10.3 Conclusion

En ajustant les paramètres, SSPACE permet déjà de produire des scaffolds uniques couvrant quasi tout le génome avec une très bonne qualité. Les scaffolds restant sont composés d'uniques "courts" contigs n'ayant pas passé les filtres.

Chaque contig est bien relié à son successeur direct.

Les résultats de qual / cov sont cependant obtenus avec l'outil dnadiff, et non à l'aide d'un aligneur.

Lors de l'alignement des (longs) scaffolds produits avec BWA, le meilleur hit donne :

ADP1, qual = 15 %
Ecoli, avec 101K MinION : qual = 20 %
Ecoli, avec 25K MinION : qual = 20 %

=> Faible qualité due à l'ajout de N lors du scaffolding ? Présents en forte densité pour Ecoli, mais peu présents (3 265) pour ADP1

Les résultats Ecoli sont similaires avec 25K ou 101K MinION => Il n'est pas nécessaire d'utiliser énormément de reads MinION pour scaffolder correctement

11 Guided Assembly (avec BLASR, filtre Abyss + paramètres Karlsson)

Avec notre parcours, on obtient des scaffolds regroupant les contigs dans le même ordre que les scaffolds produits par SSPACE, mais la qualité est supérieure, lorsqu'on force le scaffold à débuter par le contig débutant le scaffold dans le résultat SSPACE :

Pour ADP1 : qual = 99,96, cov = 99,99, matches = 26 avec dnadiff, qualBestHit = 66 % avec BWA

Pour Ecoli : qual = 99,74, cov = 99,87, matches = 64 avec dnadiff, qualBestHit = 60 % avec BWA ;

Les résultats ci-dessus sont obtenus sans consensus (aussi bien pour les gaps que pour les overlaps), mais il est clair qu'ajouter les bases des LR permet de couvrir d'avantage le génome et d'obtenir moins de matches avec dnadiff, au coût d'une légère perte de qualité, et d'obtenir une meilleure qualité lors de l'alignement avec BWA, et donc des génomes réellement plus proches

12 Tableaux comparatifs SSPACE / Guided Assembly

Génome	Scaffoldeur	nb scaffolds	cov	qual	matches	qualBestHit (BWA)
ADP1	SSPACE	1	99,92	99,98	35	15
	Nous	1	99,99	99,96	26	66
	NaS	1	100	99,99	8	45,17
	Miniasm	7	78,68	84,59	1120	N.A
Ecoli	SSPACE	1	98,68	99,97	167	20
	Nous	1	99,87	99,73	65	60
	NaS	1	100	99,99	8	59,52
	Miniasm	1	92,08	88,63	738	N.A.

Remarque : Même en conservant les scaffolds composés d'uniques courts contigs lors de la comparaison au génome de référence, SSPACE ne permet pas d'atteindre la même couverture que nous (alors que nous ne sortons pas les courts contigs non utilisés lors du scaffolding dans le fichier de résultats), quels que soient les paramètres utilisés

Test sur un plus grand génome (arabidopsis, 30M) => Abyss interminable

13 Autres aligneurs potentiels

13.1 Minimap

Permet de mapper instantanément (< 10 sec en multithreads) l'ensemble des LR sur les contigs de SR, mais quels que soient les paramètres, les liens déduits du mapping ne sont pas toujours corrects

13.2 DALIGN

Permet également de mapper très rapidement (< 3 min) l'ensemble des LR sur les contigs des SR, mais les liens déductibles des mapping semblent également être incorrects

13.3 LAST

Permet de mapper rapidement (< 20 min) l'ensemble des LR sur les contigs de SR, mais les liens déduits du mapping sont globalement incorrects

14 PBJelly2

Fait également du scaffolding et du gap-filling en se servant de LR mappés sur des contigs avec BLASR
=> Pas testé, pas réussi à installer

15 Comparaison k-mers LR

15.1 GkA

Bug : Quand on recherche un k-mer de taille > 63, il apparaît dans tous les reads, même ceux composés uniquement d'une même lettre répétée

Impossible de rechercher un k-mer à partir de sa séquence

15.2 PgSA

Construction beaucoup trop longue (plus de 12h)

15.3 CGkA

Une fois l'index construit, recherche de tous les k-mers des LR en < 10 min

15.4 Recherche de k-mers espacés avec GkA

Il suffit d'utiliser un spaced-SA

15.4.1 Tableaux

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
CR[j]	a	a	c	a	a	c	t	c	a	a	t	t	c	a	a	a	c	a	a	g	c
SA[j]	14	0	17	13	8	3	1	4	15	18	9	20	12	2	16	7	5	19	11	6	10
g(j)	0	1	2					3	4	5					6	7	8				

i	0	1	2	3	4	5	6	7	8
GkSA[i]	6	0	4	1	7	5	2	8	3
Rank	0		1	2	3	4	5	6	7

i	0	1	2	3	4	5	6	7	8
GkIFA[i]	0	2	5	7	1	4	0	3	6

l	-1	0	1	2	3	4	5	6	7
GkCGA[l]		2	1	1	1	1	1	1	1
GkCFPS[l]	0	2	3	4	5	6	7	8	9

15.4.2 Requêtes

Q1 : f = aa_aa, j = 0

$Ind_k = \emptyset$
t = 0
$l_f = 0$
$u_f = 2$
prev = -1
i = 0
readIndex = inf(g^{-1}(0) / m) = 0
$Ind_k = 0$
prev = 0
i = 1
readIndex = inf(g^{-1}(6) / m) = inf(14 / 7) = 2
$Ind_k = 0, 2$

Les reads contenant le k-mer aa_aa sont donc 0 et 2

Q3 : f = aa_aa, j = 0

$Pos_k = \emptyset$
t = 0
$l_f = 0$
$u_f = 2$
i = 0
readIndex = inf(g^{-1}(0) / m) = 0
posInRead = g^{-1}(GkSA(0)) % m = g^{-1}(0) % m = 0 Pos_k = (0,0)
i = 1
readIndex = inf(g^{-1}(6) / m) = inf(14 / 7) = 2
posInRead = g^{-1}(GkSA(1)) % m = g^{-1}(6) % m = 14 % 7 = 0 Pos_k = (0,0), (2,0)

Le k-mer aa_aa apparait donc dans le read 0 à l'indice 0 et dans le read 2 à l'indice 0

15.4.3 Calcul du spaced-SA

Il existe des algorithmes (notamment DisLex) adaptant la construction classique des SA au cas des SA espacés, en temps linéaire. L'implémentation de l'algo n'est cependant

pas disponible.
En pratique (dans LAST) un radix sort est utilisé, il est théoriquement inférieur, mais rapide en pratique.

15.5 ADP1

=> Tout ce qui est présenté avec les 32-mers est faux, bien que la différence avec les réelles valeurs soit faible (Sauf pour le calcul des k-mers-0-10 espacés, trimmed et calculs avec perM)

15.5.1 LR vs Gen. Ref

Avec des 64-mers :

k-mers présents dans le gen ref : 198 612
k-mers non présents dans le gen ref : 375 800 759

15.5.2 LR vs SR

Avec des 64-mers :

k-mers présents dans les SR : 436 257
k-mers non présents dans les SR : 375 563 114

15.5.3 LR vs contigs SR, k-mers contigus

Avec des 64-mers :

k-mers présents dans les contigs : 380 967
k-mers non présents dans les contigs : 375 618 404

Total = 375 999 371
Avec des 32-mers :

k-mers présents dans les contigs : 2 942 135
k-mers non présents dans les contigs : 332 894 134

Total = 335 836 269

Avec des 16-mers :

k-mers présents dans les contigs : 6 197 488
k-mers non présents dans les contigs : 316 296 428

Total = 322 493 916
Avec des 8-mers :

k-mers présents dans les contigs : 65 309
k-mers non présents dans les contigs : 67

Total = 65 376
=> Idéal = utilisation de 16-mers espacés ?

! ! ! Ici, le comptage des k-mers espacés est réalisé en cherchant séparément deux k/2-mers, et en analysant leur distance ! ! !

15.6 k-mers espacés : quelle définition ?

1. Utilisation d'un k-mer, en lui affectant des positions "jokers" qui peuvent match ou non
2. Utilisation d'un k-mer, en ajoutant un trou de taille fixé toutes les n bases

Exemple : Avec le 8-mer GATCGATC

1. On peut avoir les 8-mers suivants (entre autres) : GA*C*ATC, *ATCG**C, *AT**ATC (Convient plutôt pour des erreurs de subs.)
2. Avec un trou de taille 1 toutes les deux bases, on obtient le "8"-mer suivant : GA*TC*GA*TC (Convient plutôt pour des erreurs d'indels)

15.6.1 LR vs contigs SR, k-mers-1-espacés

Avec des 64-mers :

Number of reads found : 29 002
Number of reads unfound : 375 970 369

Avec des 32-mers :

Number of reads found : 332 329
Number of reads unfound : 335 503 940

Avec des 16-mers :

Number of reads found : 1 924 528
Number of reads unfound : 320 569 388

15.6.2 LR vs contigs SR, k-mers-2-espacés

Avec des 64-mers :

Number of reads found : 11 017
Number of reads unfound : 375 988 354

Avec des 32-mers :

Number of reads found : 135 974
Number of reads unfound : 335 700 295

Avec des 16-mers :

Number of reads found : 1 332 846
Number of reads unfound : 321 161 070

15.6.3 LR vs contigs SR, k-mers-3-espacés

Avec des 64-mers :

Number of reads found : 3 269
Number of reads unfound : 375 996 102

Avec des 32-mers :

Number of reads found : 43 314 Number of reads unfound : 335 792 955

Avec des 16-mers :

Number of reads found : 1 076 159
Number of reads unfound : 321 417 757

15.6.4 LR vs contigs SR, k-mers-0-10-espacés

Avec des 64-mers :

Number of reads found : 425 155
Number of reads unfound : 375 574 216

Avec des 32-mers :

Number of reads found : 3 859 742
Number of reads unfound : 369 963 314

Impossible de chercher des 21-mers car la recherche de k-mers espacés se fait pour le moment en coupant en deux parties égales le k-mer à rechercher, et en chargeant un mémoire l'index des k/2-mers dans lequel sont recherchées les deux parties.

Avec des 22-mers :

Number of reads found : 6 447 002
Number of reads unfound : 356 466 973

Avec des 20-mers :

Number of reads found : 7 070 951
Number of reads unfound : 351 233 270

Avec des 16-mers :

Number of reads found : 16 036 610
Number of reads unfound : 306 457 306

15.6.5 LR vs contigs SR, k-mers-0-10-espacés, en recherchant ensuite les k/2-mers non trouvés

64-mers :

Number of reads found : 425 155
Number of reads unfound : 375 574 216

32-mers :

Number of reads found : 3 848 293
Number of reads unfound : 369 758 181

16-mers :

Number of reads found : 15 984 115
Number of reads unfound : 306 069 664

=> Un peu mieux que de tout chercher séparément, mais toujours pas satisfaisant

15.6.6 LR 8-trimmed vs contigs SR, k-mers-0-10-espacés

64 mers :

Number of reads found : 425 045
Number of reads unfound : 374 437 074

32 mers :

Number of reads found : 3 859 051
Number of reads unfound : 368 643 639

=> Les erreurs des LR ne sont donc pas en début / fin, mais aléatoires

15.6.7 LR vs contigs SR, avec perM (2 mismatches autorisés)

64-mers :

Number of reads found : 902 601
Number of reads unfound : 375 096 770

32-mers :

Number of reads found : 9 021 370
Number of reads unfound : 364 801 686

23-mers (21 bases solides) :

Number of reads found : 15 231 625
Number of reads unfound : 349 523 882

22-mers (20 bases solides) :

Number of reads found : 16 478 453
Number of reads unfound : 346 435 522

16 mers :

Number of reads found : 209 728 714
Number of reads unfound : 112 765 202

=> On trouve plus de 16-mers que précédemment car une erreur autorisée par 8-mer, alors que lorsqu'on recherchait "nos" k-mers espacés précédemment, aucune erreur n'est tolérée à part le gap entre les deux parties du 16-mer espacé. Couper nos k-mers espacés plus de deux parties pour avoir de meilleurs résultats ?

15.6.8 contigs SR vs LR, k-mers contigus

Avec des 64-mers :

Number of reads found : 363 440
Number of reads unfound : 3 189 979

Avec des 32-mers :

Number of reads found : 2 307 010

Number of reads unfound : 1 246 142

Avec des 16-mers :

Number of reads found : 3 345 672
Number of reads unfound : 198 626

Avec des 8-mers :

Number of reads found : 65 177
Number of reads unfound : 158

15.6.9 contigs SR vs LR, k-mers-1-espacés

Avec des 64-mers :

Number of reads found : 8 894
Number of reads unfound : 3 544 525

Avec des 32-mers :

Number of reads found : 137 805
Number of reads unfound : 3 415 347

Avec des 16-mers :
=> + de 48h à run => Abandon
Number of reads found :
Number of reads unfound :

15.6.10 Cause des mauvais résultats

En fait, les reads indexés das GkA / CGkA / PgSA doivent être de même longueur => Incidence sur les résultats ?
Lors de la recherche d'un k-mer dans l'index composé de reads de longueurs différentes, les positions des occurrences ne sont pas correctes, mais le nombre d'occurrences, lui, semble l'être (i.e. si un k-mer est présent, il sera trouvé, mais sa position ne sera pas toujours bonne)

15.6.11 NaS vs contigs SR

Avec des 64-mers :

k-mers présents dans les contigs : 7 068 450
k-mers non présents dans les contigs : 45 986

15.7 Recherche des spaced k-mers des LR DANS les LR ?

=> Peut potentiellement permettre de détecter les k-mers présents dans gen. ref, mais qui ont été victimes d'erreurs d'indels

16 PgSA-NaS-like correction

16.1 Idée

Mapper les SR sur les LR pour obtenir des seeds (avec PBLAT, 8 threads), puis comme pour l'idée du stage, trier les seeds par position et les fusionner si leurs positions d'alignement se chevauchement, puis tenter de relier ces seeds par des extensions à l'aide de PgSA, car c'est la seule structure permettant de faire des requêtes avec des valeurs de k variables, et donc de rechercher des chevauchements exacts de longueurs k, puis si aucun n'a été trouvé, de longueur k - 1, etc

Deux approches pour relier les seeds :

1. Extension à l'aide de SR complets => Résultats peu satisfaisants, car les SR sont de bonne qualité mais contiennent tout de même quelques erreurs, ce qui peut amener à l'impossibilité de relier deux seeds, à cause de l'absence de chevauchements exacts. De plus, PgSA ne supporte pas les reads de longueur variable.
2. Utilisation des k-mers des SR (et de leurs RC) pour l'extension => Meilleurs résultats, car erreurs présentes en moins grand nombre dans les k-mers, et permet de palier le fait que PgSA ne supporte pas les reads de longueur variable sans perdre d'information => Approche retenue

16.2 Observations

Étendre un seed à l'aide de k-mers se chevauchant parfaitement se fait aisément et rapidement et ce jusqu'à une assez grande distance (jusque 370k en moins de 10 secondes)

En revanche, relier un seed à un autre (i.e. relier le dernier k-mer du seed de gauche au premier k-mer du seed de droite) est plus difficile et nécessite du backtracking.

De plus, pour relier 2 seeds provenant du mapping des SR sur les LR, il semble nécessaire qu'une grande partie des bases des SR s'alignent correctement sur le LR, afin de s'assurer de la bonne qualité des seeds utilisés (on veut éviter les seeds contenant des erreurs, et donc privilégier les seeds s'alignant à 100 % sur le gén. ref., afin d'être sûr de pouvoir relier les seeds à l'aide de chevauchements exacts) => Nécessité d'ajuster les paramètres de PBLAT

Ces filtres, si la contrainte sur le nombre de bases alignées est trop forte (testé avec minScore=150), rendent cependant difficile voire impossible la correction de nombreux

LR, à cause d'un trop faible nombre de seeds par LR, ou de l'absence totale de seeds pour certains LR => Nécessaire d'utiliser "autre chose" que des SR comme seeds ? Testé avec des k-mers, mais pas concluant

Il est également possible de corriger les SR, par exemple avec Quorum, qui permet de gagner 1,xx % de qualité moyenne supplémentaire en 2 minutes (10 min CPU) sur le jeu de SR de longueur 250 utilisé par NAS, et en 12 minutes (171 min CPU) sur le jeu de SR complet
Quorum permet alors d'obtenir des seeds de bonne qualité (s'alignant à 100 % sur le gen. ref.) à chaque fois, même sans fixer de seuil sur le nombre de bases alignées sur le LR, et présents en plus grande quantité sur chaque LR, permettant ainsi une correction plus aisée, et applicable à plus de LR.

Les tests suivants on été réalisés avec le sous-ensemble de SR de longueur 250 utilisé par NaS.

16.3 Tests partie 1 : SR bruts

16.3.1 Test 1

LR 2D, len = 34 565, identité = 24,38 %

SR mappés avec minScore = 150

Nombre de seeds = 25, mais le 24ème et le 25ème ne semblent pas pouvoir être reliés => Pour les besoins de l'exemple, LR tronqué pour éliminer le 25ème seed.

LR tronqué : len = 28 674, identité = 29,39 %

L'application de la procédure de correction produit alors un LR synthétique de longueur 21 719, s'alignant avec 4 erreurs, en < 2 sec + 1 sec fusion / overlapping des seeds

La longueur du LR synthétique obtenue est plus petite que celle du LR initial, mais il est également possible de forcer l'extension du LR synthétique obtenu, à gauche (du seed le + à gauche) et à droite (du seed le + à droite), pour gagner un peu plus de longueur.

16.3.2 Test 2

LR 2D, len = 32 436, identité = 0,21 %

SR mappés avec minScore = 150 => Aucun seed, correction impossible

Sans minScore => Correction impossible car présence de seeds de trop faible identité (75 %)

16.3.3 Test 3

LR 2D, len = 8 878, identité = 3,05 %

SR mappés avec minScore = 150

Nombre de seeds = 2

LR obtenu : len = 2 492, identité = 100 %, temps : < 1 sec + 1 sec fusion / overlapping des seeds (LR très court à cause du faible nombre de seeds)

SR mappés sans minScore

Nombre de seeds = 18, mais le 6ème et le 7ème, et le 10ème et le 11ème semblent ne pas vouloir se relier => Pour les besoins de l'exemple, lie donc les seeds de 1 à 6, de 7 à 10 et de 11 à 18 séparément

LR1 obtenu : len = 1 985, identité = 99,75 %
LR2 obtenu : len = 2 038, identité = 96,96 %
LR3 obtenu : len = 4 112, identité = 99,93 %
Temps total : < 0,7 sec + 1 sec fusion / overlapping des seeds

Les LR obtenus sont de moins bonne qualité à cause du choix de seeds ne s'alignant pas à 100 % sur le gén. ref., et sont fragmentés à cause des erreurs présentes sur les seeds.

16.4 Tests partie 2 : SR corrigés

16.4.1 Test 1

LR 2D, len = 34 565, identité = 24,38 %

Nombre de seeds = 68

LR1 obtenu : len = 35 433, identité = 100 %
Temps : < 0,6 sec + 1 sec fusion / overlapping des seeds

16.4.2 Test 2

LR 2D, len = 32 436, identité = 0,21 %

Nombre de seeds = 56, mais le 41eme et le 42eme se mappent en position 900k sur le gén. réf. alors que les autres se mappent en position 300k => Pour les besoins de l'exemple, on supprime ces deux seeds de la liste de seeds à relier

LR obtenu : len = 26 181, identité = 100 %
Temps : < 1 sec + 1 sec fusion / overlapping des seeds

Cependant, le premier seed se mappe en position 4 069 sur le LR, en étendant le LR synthétique obtenu, à gauche, il est donc possible d'obtenir un LR synthétique de longueur environ 30 250, assez proche de la longueur du LR initial.

16.4.3 Test 3

LR 2D, len = 8 878, identité = 3,05 %

Nombre de seeds = 16

LR obtenu : len = 8 534, identité = 100 %, temps : < 0,5 sec + 1 sec fusion / overlapping des seeds

Ici aussi, il est possible d'étendre le LR synthétique obtenu, à gauche, sur une longueur de 262, et donc d'obtenir un LR synthétique de longueur 8 796, assez proche de la longueur du LR initial.

16.5 Tests partie 3 : Avec NaS + SR bruts

La correction produit un fichier unique, et au format fasta, même lorsque la correction se fait sur des reads pairés. Comme NaS nécessite des reads pairés séparés en deux fichier ET en format fastq, les tests ont été réalisés avec les SR bruts.

16.5.1 Test 1

LR 2D, len = 34 565, identité = 24,38 %

NaS : len = 37 179, identité = 100 %, temps (correction) = 2min

16.5.2 Test 2

LR 2D, len = 32 436, identité = 0,21 %

NaS : len = 27 627, identité = 100 %, temps (correction) = 2min

16.5.3 Test 3

LR 2D, len = 8 878, identité = 3,05 %

NaS : len = 10 612, identité = 100 %, temps (correction) = 1min

16.6 Conclusion tests sur ces 3 LR

=> Sur ces 3 exemples, on semble donc être envrion 30 à 60 fois plus rapide que NaS.

De plus, les SR corrigés semblent permettre de produire de bien meilleurs résultats au prix d'un pré-traitemet de correction, très peu coûteux en temps. Les SR seront donc utilisés corrigés dans les tests suivants.

16.7 Tests partie 4 : sur le jeu d'exemple NaS, avec notre méthode + SR corrigés

M1 : < 0,3 sec, len template = 7 380, len res = 7 253, id res = 100 %
M2 : Pas de seeds (NaS ne le corrige pas non plus)
M3 : 0,3 sec, len template = 7 994, len res = 7 622, id res = 100 %

M4 : < 0,4 sec, len template = 10 464, len res = 10 103, id res = 100 %
M5 : 0,2 sec, len template = 2 512, len res = 2 433, id res = 100 %

Taille totale : 28 350

Il est possible d'étendre les LR produits un peu à droite et à gauche pour se rapprocher de la longueur initiale du template.

Temps total : 4 * 1 sec (Fusion et overlapping des seeds) + 1 sec (relier seeds pour les 4 reads) + 5 * 4 sec (mapping SR -> LR) = 25 sec (dont 5 sec de correction), contre 4min13 avec NaS (dont 3min37 de correction), produisant une taille totale de 31 008 => Correction 43,4 fois plus rapide qu'avec NaS

16.8 Tests partie 5 : Sur un jeu de LR 2D de ADP1

Jeu initial : 602 LR, avLen = 5 197, cumSize = 3 128 834, avId = 8,83 %

Corrigé avec NaS : 445 LR, avLen = 5 798, cumSize = 2 580 256, avId = 99,83 temps = 5h49min, dont 5h40 de correction

Corrigé avec notre méthode : 540 LR, dont 25 fragmentés, avLen = 4 901, cumSize = 2 778 596 avId = 99,84 %, temps = 40 min

En étendant à gauche et à droite : 540 LR, dont 25 fragmentés, avLen = 5 853, cumSize = 3 318 575, avId = 99,85 %, temps = 40 min

=> En fait, cette méthode de mapper sur chaque LR séparément n'est pas viable si on corrige un ensemble de LR de plus grande taille, car elle aura un coût en temps beaucoup trop important

En mappant directement les SR sur tous les LR, on obtient les résultats suivants : 402 LR, dont 20 fragmentés, avLen = 4 996, cumSize = 2 118 182, avId = 99,85 %, temps = 6min52

En étendant à gauche et à droite : 402 LR, dont 20 fragmentés, avLen = 5 910, cumSize = 2 505 855, avId = 99,86 %, temps = 7min

On remarque que les résultats sont différents quand on mappe tous les SR d'un coup,

ou séparément sur chaque template => Parce quand on mappe tout d'un coup, certains LR n'ont qu'un seul seed, et la correction n'est pas amorcée.
=> Pour corriger plus de LR en mappant tout d'un coup : Autoriser la correction d'un LR même s'il n'a qu'un seed. Dans ce cas, on ne fait qu'étendre ce seed au maximum à gauche et à droite

En mappant tout d'un coup, et en étendant les seeds uniques : 443 LR dont 22 fragmentés, avLen = 5 627, cumSize = 2 633 407, avId = 99,91 %, temps = 8min23

16.9 Tests partie 6 : Sur un jeu de LR 1D de ADP1

Jeu initial : 10 567 LR, avLen = 1 873, cumSize = 19 788 858, avId = 3,68 %

Corrigé avec NaS : 784 LR, avlen = 3 764, cumSize = 2 951 256, avId = 99,76 %, temps = 11h20

Corrigé avec notre méthode : 786 LR dont 59 fragmentés, avLen = 9 871, cumSize = 8 420 218, avId = 99,59 %, temps = 23min55

16.10 Tests partie 7 : Sur tous les LR 1D et 2D de ADP1

16.10.1 Reads 1D

Jeu initial : 70 314 LR, avLen = 2 530, 373 alignés avec BWA, avId = 3,84 %

Corrigé avec NaS : => Long

Corrigé avec notre méthode : => 8 230 LR dont 499 fragmentés, avLen = 9 748, avId = 99,57 %, temps = 4h55

16.10.2 Reads 2D

Jeu initial : 18 697, avLen = 10 884, 13 636 alignés avec BWA, avId = 37,07 %

Corrigé avec NaS :

Corrigé avec notre méthode :

=> Long

16.11 Tests partie 8 : Sur un sous-ensemble de LR 2D de Ecoli

Jeu initial : 500 LR, avLen = 5 812, cumSize = 2 906 183, avId = 40,13 %

Corrigé avec NaS : 495, avLen = 7 786, cumSize = 3 853 897, avId = 99,80 %, temps = 7h18

Corrigé avec notre méthode : 496 LR dont 56 fragmentés, avLen = 5 095, cumSize = 2 898 952, avId = 99,82 %, temps = 14min33

16.12 Tests partie 9 : Sur des sous-ensembles de LR de Yeast

16.12.1 Sous-ensemble de LR 1D

Jeu initial : 500 LR, avLen = 5 660, cumSize = 2 829 976, avId = 2,87 %

Corrigé avec NaS : 139 reads, avLen = 4 359, cumSize = 605 864, avId = 96,67 %, temps = 5h39min

Corrigé avec notre méthode : 109 LR dont 12 fragmentés, avLen = 5 367, cumSize = 654 831, avId = 98,91 %, temps = 13min40

Avec BLASR : 157 LR dont 33 fragmentés, avLen = 5 586, cumSize = 1 156 217, avId = 98,63 %, temps = idk

16.12.2 Sous-ensemble de LR 2D

Jeu initial : 500 LR, avLen = 6 154, cumSize = 3 076 935, avId = 8,08 %

Corrigé avec NaS : 279 LR, avLen = 7 520, cumSize = 2 098 154, avId = 97,51 %, temps = 11h14

Corrigé avec notre méthode : 233 LR dont 41 fragmentés, avLen = 6 127, cumSize = 1 740 039 , avId = 99,76 %, temps = 35min24

16.13 Conclusion des tests

— L'intérêt de corriger les SR avec Quorum avant de les utiliser pour relier les seeds est donc clair, surtout au vu du faible coût en temps de cet étape.

— Il semble possible de produire des reads synthétiques de très haute qualité, environ 20 à 30 fois plus rapidement qu'avec NaS

— Après vérification, nos LR synthétiques couvrent autant (voir un peu +) le génome de référence que les LR de NaS, et se mappent aux mêmes endroits => Atteste de la bonne qualité de nos LR

16.14 Baisse du seuil d'accuracy de BLAT

Sur un jeu de données de 602 LR 2D ADP1 :

Avec paramètres initiaux (comme NaS, tileSize=10, stepSize=5) : 443 LR dont 22 fragmentés, avLen = 5 627, cumSize = 2 633 407, avId = 99,91 %, temps = 8min23

minIdentity à 70% => 465 LR dont 20 fragmentés, avLen = 5 563, cumSize = 2 731 412, avId = 99,81 % , temps = 13min24

minScore à 15 => 455 LR dont 150 fragmentés, avLen = 3 172, cumSize = 2 112 758, avId = 99,87 %, temps = 23min30

maxGap à 3 => 444 LR dont 21 fragmentés, avLen = 5 636, cumSize = 2 648 742, avId = 99,87 %, temps = 14min25

Tiles jamais overused => Pareil que paramètres de base

=> Pas concluant

16.15 BLASR

BLASR dispose d'une sortie au format SAM (obsolète sur la version utilisée ici, mais impossible d'installer la dernière version de BLASR), et a donc testé comme aligneur pour

potentiellement remplacer BLAT.

16.15.1 Sur un jeu de données de 602 LR 2D ADP1

Avec BLAT : 443 LR dont 22 fragmentés, avLen = 5 627, cumSize = 2 633 407, avId = 99,91 %, temps = 8min23

Temps de mapping BLAT : 30 sec Temps de mapping BLASR : 7 min

Avec BLASR, paramètres par défaut : 492 LR dont 18 fragmentés, avLen = 5 486, cumSize = 2 825 415, avId = 99,96 %, temps = 19min18

Avec BLASR, minMatch=10 + maxScore=-150 : 513 LR dont 33 fragmentés, avLen = 5 169, cumSize = 2 843 729, avId = 99,96 %, temps = 28min20

En comparaison, NaS, en mode sensitive, permet de corriger au plus 490 reads, avLen = 6 702, temps = 2h avec 8 threads

16.15.2 Sur un jeu de données de 10 567 LR 1D ADP1

Avec BLAT : 786 LR dont 59 fragmentés, avLen = 9 871, cumSize = 8 420 218, avId = 99,59 %, temps = 23min55

Temps de mapping BLAT : 30 sec
Temps de mapping BLASR : 13 min

Avec BLASR, paramètres par défaut : 1 323 LR dont 107 fragmentés, avLen = 9 340, cumSize = 13 711 143, avId = 99,61 %, temps = 1h33

Avec BLASR, minMatch=10 + maxScore=-150 : 1 503 dont 259 fragmentés, avLen = 7 390, cumSize = 13 655 833 avId = 99,67 %, temps = 4h

En comparaison, NaS, en mode sensitive, permet de corriger au plus 1 166 reads, avLen = 6 019, avId = 99,61 %, temps = 6h25 avec 8 threads

16.15.3 Remarques

— BLASR permet donc effectivement de corriger plus de LR que BLAT, au prix d'une augmentation du temps d'exécution. Il sera donc choisi comme aligneur pour le reste des tests.

— BLASR est plus lent que BLAT pour mapper les SR sur les LR => D'où l'augmentation du temps d'exécution

— BLASR permet de corriger beaucoup + de LR, tout en conservant des résultats de bonne qualité

— Les temps d'exécution ne sont pas fiables, car au moment des tests, beaucoup d'autres processus en train de run sur le serveur

— Temps d'exec tout de même plus long avec BLASR qu'avec BLAT, car corrige + de LR, et car plus de seeds sont présents sur chaque LR, donc plus d'opérations de reliures sont nécessaires.

— Si quand il est impossible de relier 2 seeds, on essaye de relier le seed de gauche au seed de droite suivant => Énorme perte de temps

— Ajustement des paramètres => A part l'option présentée (minMatch + maxScore) pas de grand changement / grande amélioration quels que soient les paramètres ajustés

— Nécessité de trouver une valeur idéale pour la taille des *k*-mers, testé avec 32, 128, 50 => Mauvais résultats

— Autre gros avantage de notre méthode : Aucun fichier temporaire, et pas besoin d'espace disque supplémentaire, alors que NaS génère un fichier d'alignement pour chaque LR brut (+ de 100Go utilisés pour corriger tout ADP1)

16.16 Problèmes et solutions potentielles

— Quand on cherche à relier 2 k-mers, fixer une taille max à la séquence de SR permettant la reliure, afin de ne pas passer trop de temps dans le backtracking pour rien. Pour l'instant, taille limite = taille du template. Testé avec taille = 130 / 100 * gapSize, mais pas concluant

16.17 Ajustement paramètres sur M12D

k	Nb reads	avLen	cumSize	avId	Cov
32 (min overlap 25)	492 (16 fragmentés)	5 504	2 823 450	99,18	1 972 924
32 (min overlap 30)	492 (15 fragmentés)	5 618	2 859 582	98,90	1 973 887
42	492 (15 fragmentés)	5 539	2 830 674	99,844	1 979 966
43	492 (14 fragmentés)	5 551	2 831 240	99,837	1 979 751
44	492 (14 fragmentés)	5 540	2 830 832	99,838	1 980 307
45	492 (14 fragmentés)	5 540	2 830 854	99,838	1 980 320
46	492 (15 fragmentés)	5 516	2 829 796	99,846	1 976 007
48	492 (16 fragmentés)	5 517	2 830 061	99,85	1 976 021
56	492 (16 fragmentés)	5 516	2 829 914	99,84	1 977 316
64	492 (18 fragmentés)	5 486	2 825 415	99,960	1 976 214
68	492 (17 fragmentés)	5 483	2 823 564	99,962	1 973 151
72	492 (17 fragmentés)	5 476	2 825 441	99,965	1 974 181
74	492 (18 fragmentés)	5 464	2 824 974	99,943	1 975 095
76	492 (18 fragmentés)	5 463	2 824 478	99,944	1 972 813
80	492 (19 fragmentés)	5 457	2 821 324	99,942	1 973 409
88	492 (18 fragmentés)	5 470	2 827 949	99,91	1 976 253
96	491 (19 fragmentés)	5 489	2 827 059	99,81	1 976 027
112	491 (17 fragmetés)	5 501	2 827 390	99,89	1 971 901
128	484 (16 fragmentés)	5 550	2 808 125	99,89	1 960 289

Diminuer la valeur de k provoque une légère baisse de qualité, mais permet de moins fragmenter, et de mieux couvrir le génome de référence, et d'obtenir une taille moyenne plus grande.

Inversement, augmenter la valeur de k provoque une légère augmentation de la qualité (qui diminue à nouveau au delà d'un certain seuil), mais une diminution de la couverture du génome de référence, et une diminution de la longueur moyenne (qui semble réaugmenter au delà d'un certain seuil)

=> Puisqu'on cherche, au final, à assembler, plus intéressant de choisir un k permettant de bien couvrir le génome ? De plus, la baisse de qualité est très faible (de l'ordre de 0,1 %)

16.18 Qualité des LR produits (k = 64)

16.18.1 ADP1 (k = 64)

NaS (fast) : 24 063 LR, avLen = 8 840, cumSize = xxx, avId = 99,82 %

NaS (sensitive) : 28 492, avLen = 9 530, cumSize = xxx, avId = 99,83 %, temps = un peu moins de 72h avec 8 threads

Nous (BLASR, besthit=30) : 22 999 LR (dont 1 233 fragmentés), avLen = 10 465, cumSize = xxx, avId = 99,55 %, temps = 35h

16.18.2 E. Coli (k = 45)

NaS (fast) : 21 818 LR, avLen = 7 926, cumSize = xxx, avId = 99,86 %

NaS (sensitive) : 22 144 LR, avLen = 8 307, cumSize = xxx, avId = 99,86 %

Nous (BLASR, paramètres par défaut) : 21 921 LR (dont 1 572 fragmetés), avLen = 5 544, cumSize = xxx, avId = 99,70

16.18.3 Yeast

...

16.19 Qualité de l'assemblage

Avec notre méthode, en conservant les paramètres par défaut, impossible d'obtenir un seul contig. En revanche, en ajustant les paramètre, il est possible d'obtenir un contig unique

16.19.1 ADP1 (k = 64)

NaS (fast et sensitive, paramètres par défaut) : 1 contig , cov = 100 %, id = 99,98 %

En corrigeant tout d'un coup : 1 contig, cov = 99,94 %, id = 99,97 %

=> Probablement possible d'obtenir de meilleurs résultats en ajustant davantage les paramètres, aussi bien de Canu que de notre méthode

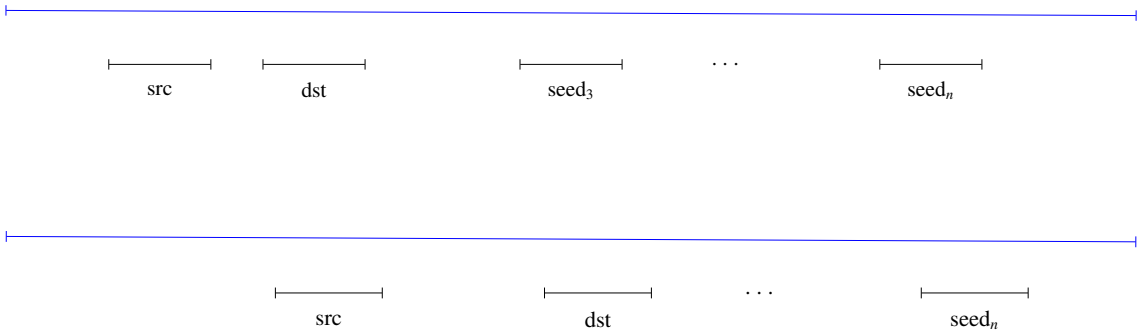
16.19.2 Ecoli (k = 45)

=> Impossible d'obtenir un unique contig avec notre méthode, même en modifiant la valeur de k, et les paramètres de Canu

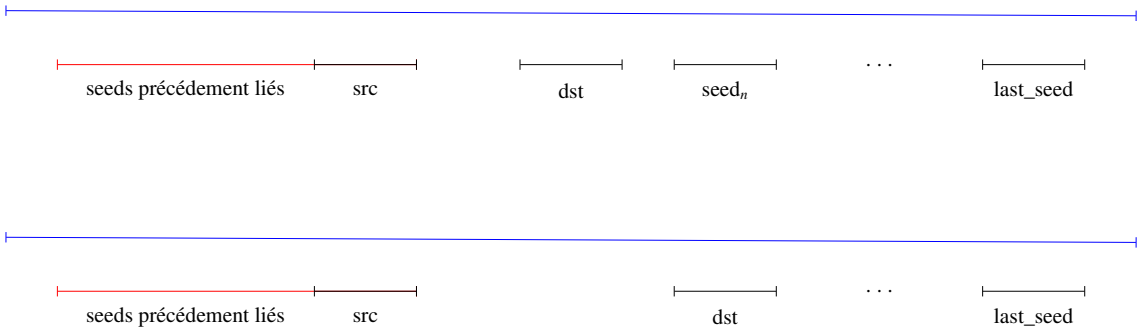
16.20 Solution pour ne pas produire de LR fragmentés

S'il est impossible de relier un couple de seeds, on change la cible et/ou la source, en différenciant deux cas :

— Cas 1 : Aucun seed n'a été relié pour le moment => On change la cible et la source



— Cas 2 : On a déjà précédemment relié des seeds => On change uniquement la cible



Résultats observés, sur un ensemble de LR 2D de ADP1 :

Version fragmenté :

avLen = 5 513,27 ; cumSize = 2 839 335 ; avId = 99,87 ; genCov = 1 981 134 ; time = 4min10

Version non fragmentée :

avLen = 5 755,9 ; cumSize = 2 849 170 ; avId = 99,81 ; genCov = 1 988 995 ; time = 4min19

Résultats observés, sur tout ADP1 :

Version fragmenté :

avLen = ; cumSize = ; avId = ; genCov = ; time =

Version non fragmentée :

avLen = ; cumSize = ; avId = ; genCov = ; time =

16.20.1 Recherche du nb de backtracks

2000 : avLen = 5 755,9 ; cumSize = 2 849 170 ; avId = 99,81 ; genCov = 1 988 995 ; time = 4min19

1500 : comme 2000

1250 : comme 2000

1125 : comme 2000

1000 : avLen = 5 739,6 ; cumSize = 2 841 121 ; avId = ; genCov = ; time =

16.20.2 Recherche valeur de k idéale sur M12D

k	Nb reads	avLen	cumSize	avId	Cov
32	495	5 740	2 841 261	99,45	1 978 200
45	495	5 750	2 846 483	99,77	1 989 078
50	495	**5 759**	**2 850 501**	99,77	1 982 317
54	495	5 755	2 848 741	99,77	1 989 101
55	495	5 755	2 848 581	99,77	1 989 105
56	495	5 755	2 848 593	99,77	1 984 678
60	495	5 756	2 849 248	99,81	1 989 084
62	495	5 756	2 849 142	99,81	1 988 985
64	495	5 756	2 849 170	99,81	1 988 995
66	495	5 755	2 848 699	99,81	1 988 156
68	495	5 754	2 848 703	99,81	1 984 116
72	495	5 746	2 844 265	99,81	1 983 770
75	495	5 735	2 838 856	99,81	1 983 993
76	495	5 745	2 843 621	99,81	1 982 969
80	495	5 747	2 844 278	99,86	1 987 252
82	495	5 730	2 836 478	99,86	1 985 493
84	495	5 730	2 836 486	99,86	1 985 879
86	495	5 730	2 836 413	99,85	1 985 334
87	494	5 748	2 845 316	99,85	1 987 926
88	495	5 753	2 847 488	99,83	1 989 964
89	495	5 752	2 847 482	99,83	1 989 595
90	495	5 753	2 847 515	99,83	1 989 656
91	495	5 753	2 847 720	**99,87**	**1 990 233**
92	495	5 753	2 847 729	**99,87**	1 989 808
93	495	5 753	2 847 732	**99,87**	1 989 803
94	495	5 753	2 847 731	**99,87**	1 985 750
95	494	5 761	2 846 109	**99,87**	1 986 128
96	494	5 761	2 845 851	**99,87**	1 989 785
128	486	5 807	2 821 961	99,78	1 977 528

16.20.3 Test valeur idéale sur 500 LR Ecoli

k	Nb reads	avLen	cumSize	avId	Cov
50	500	6 517	3 258 541	99,53	2 299 374
53	500	6 525	3 262 594	**99,64**	2 307 474
54	500	6 543	3 271 252	99,56	2 308 899
55	500	6 527	3 263 660	99,46	2 311 302
56	500	6 528	3 264 114	99,36	2 311 736
57	500	**6 563**	**3 281 596**	99,21	**2 311 776**
58	500	6 483	3 241 683	99,46	2 296 632
60	500	6 466	3 232 802	99,63	2 293 984
64	500	6 448	3 224 087	99,54	2 295 948
76	500	6 484	3 241 991	99,40	2 295 804
80	500	6 476	3 238 008	99,41	2 296 665
91	500	6 500	3 250 194	99,46	2 293 124

16.20.4 Test valeur idéale sur 500 LR Yeast

k	Nb reads	avLen	cumSize	avId	Cov
53	129	7 948	1 025 240	97,83	960 417
54	125	8 128	1 015 984	97,80	951 538
55	125	8 297	1 037 149	97,79	970 523

16.21 Extension en dehors des templates

=> Permet d'obtenir des LR bien plus longs, et qui s'aligne avec une bonne identité.
=> MAIS, l'assemblage obtenu à partir de ces LR est moins satisfaisant que celui obtenu en n'étendant pas en dehors des templates

16.22 Coverage Yeast

scf7180000000084|quiver scf7180000000085|quiver scf7180000000086|quiver 15.65177 12.88643 15.63457 scf7180000000087|quiver scf7180000000088|quiver scf7180000000089|quiver 14.62544 13.05174 13.42334 scf7180000000090|quiver scf7180000000091|quiver scf7180000000092|quiver 18.36019 23.34035 12.80410 scf7180000000093|quiver scf7180000000094|quiver scf7180000000095|quiver 18.24795 18.16750 17.82141 scf7180000000096|quiver scf7180000000097|quiver scf7180000000098|quiver 1942.50467 68.81935 13.96040 scf7180000000099|quiver scf7180000000100|quiver scf7180000000101|quiv 77.24690 13.15148 15.50174 scf7180000000102|quiver scf7180000000103|quiver scf7180000000104|quiver

14.81758 66.14499 14.16969 scf7180000000105|quiver scf7180000000106|quiver scf7180000000107|quiver
16.19484 1419.08264 1490.21924 scf7180000000108|quiver scf7180000000109|quiver scf7180000000110|qu
79.65250 21.06729 37.45842 scf7180000000111|quiver scf7180000000112|quiver scf7180000000113|quiver
38.51891 21.84528 1507.71796

17 Comparaison de k-mers LR bruts / synthétiques

17.1 LR corrigés VS LR bruts

Sur un jeu de données de 602 LR 2D ADP1

17.1.1 64-mers

219 64-mers des LR NaS dans les LR bruts (sur 2 105 304 64-mers)

287 64-mers de nos LR dans les LR bruts (sur 2 180 160 64-mers)

17.1.2 32-mers

17 771 32-mers des LR NaS dans les LR bruts (sur 2 113 899 32-mers)

18 461 32-mers de nos LR dans les LR bruts (sur 2 189 657 32-mers)

17.1.3 16-mers

174 681 16-mers des LR NaS dans les LR bruts (sur 2 112 144 16-mers)

183 749 16-mers de nos LR dans les LR bruts (sur 2 189 705 16-mers)

17.1.4 16-mers 0-10 espacés

236 319 16-mers espacés des LR NaS dans les LR bruts (sur 2 112 144 16-mers)

247 505 16-mers espacés de nos LR dans les LR bruts (sur 2 189 705 16-mers)

17.2 LR bruts VS LR corrigés

17.2.1 64-mers

212 (sur 3 090 908) 64-mers des LR bruts dans les LR NaS

212 (sur 3 090 908) 64-mers des LR bruts dans nos LR

17.2.2 32-mers

14 178 (sur 3 110 087) 32-mers des LR bruts dans les LR NaS

14 240 (sur 3 110 087) 32-mers des LR bruts dans nos LR

17.2.3 16-mers

142 603 (sur 3 113 655) 16-mers des LR bruts dans les LR NaS

145 101 (sur 3 113 655) 16-mers des LR bruts dans nos LR

17.3 Conclusion

Puisque les k-mers des LR corrigés proviennent des SR, comparer les k-mers des LR bruts aux k-mers des LR corrigés est assez similaire aux comparaisons réalisées précédemment entre les k-mers des contigs SR et les k-mers des LR bruts, et est donc effectivement peu concluant.

Cependant le fait de ne pas trouver les mêmes résultats en cherchant les k-mers des LR bruts dans les LR corrigés, et vice-versa, est étrange ?

17.4 Comparaison des k-mers LR NaS / Nous

Sur un jeu de données de 602 LR 2D ADP1 corrigés => 445 LR

17.4.1 64-mers

1 943 454 64-mers de nos LR dans les LR NaS (sur 2 180 160)

1 889 631 64-mers des LR NaS dans nos LR (sur 2 105 304)

=> Atteste bien de la qualité des LR produits par notre méthode. De plus, comme dit précédemment, nos LR couvrent autant, voire un peu +, le génome de référence que les LR NaS.

17.5 Comparaison de similarité des LR bruts / LR corrigés

Commet calcule le % de similarité entre deux (ou plusieurs) ensemble de reads, en considérant que deux reads sont similaires s'ils partagent au moins 2 k-mers. Plante quand on teste avec k > 35, donc test réalisés avec k = 32

Test, sur un jeu de données de 602 LR 2D ADP1 corrigés => 445 LR :

17.5.1 NaS vs Nous, 32-mers

444 / 445 reads NaS ont un read similaire dans nos reads corrigés, et 466 / 468 de nos reads corrigés ont un read similaire dans NaS

=> Cohérent avec la comparaison des k-mers réalisée précédemment

17.5.2 NaS vs LR bruts

317 / 445 LR NaS ont un read similaire dans les LR bruts, et 243 / 445 LR bruts ont un read similaire dans NaS

17.5.3 Nous vs LR bruts

333 / 468 de nos LR ont un read similaire dans les LR bruts, et 245 / 445 LR bruts ont un read similaire dans nos LR

17.5.4 Conclusion

En comparant deux jeux de reads n'étant pas related (i.e. l'un n'est pas la correction de l'autre), on ne trouve aucun read similaire d'un jeu à l'autre. Commet nous permet donc au moins de déterminer de quel jeu de LR raw provient un ensemble de LR corrigé

18 Distribution de fréquence des k-mers contigus des LR

18.1 Distribution des k-mers, k = 64

Sur un jeu de données de 602 LR 2D ADP1 corrigés => 445 LR :

18.1.1 LR bruts

1 fois : 3 090 908

18.1.2 LR corrigés (nous)

1 fois : 1 814 926

2 fois : 312 486

3 fois : 46 967

4 fois : 5 781

=> Pas de 64-mers fréquents dans les LR bruts

18.2 Distribution des k-mers, k = 32

18.2.1 LR bruts

1 fois : 3 110 002

2 fois : 85

18.2.2 LR corrigés (nous)

1 fois : 1 820 286

2 fois : 315 442

3 fois : 48 000

4 fois : 5 916

5 fois : 13

=> 47 32-mers fréquents des LR bruts sont fréquents dans les LR corrigés, et 63 32-mers fréquents des LR corrigés sont fréquents dans les LR bruts

18.3 Distribution des k-mers, k = 16

18.3.1 LR bruts

1 fois : 3 107 921

2 fois : 5 633

3 fois : 64

4 fois : 19

5 fois : 13

6 fois : 3

7-26 fois : <= 3

18.3.2 LR corrigés (nous)

1 fois : 1 816 000

2 fois : 318 207

3 fois : 49 225

4 fois : 6 157

5 fois : 79

6-50 fois : <= 30

=> 2 343 16-mers fréquents des LR bruts sont fréquents dans les LR corrigés, et 2 526 16-mers fréquents des LR corrigés sont fréquents dans les LR bruts

18.4 Conclusion

Les LR bruts contiennent très peu de k-mers fréquents, et après correction, ces k-mers fréquents ne se retrouvent pas tous dans les LR corrigés (une moitié seulement en moyenne) => Pas concluant de récupérer les k-mers fréquents des LR bruts pour les travailler

19 Assemblage de k-mers fréquents LR

=> Les k-mers fréquents (apparaissant > 1 fois) couvrent au maximum 80 % du gén. ref. (pour k = 32) => Pas concluant

Les k-mers non-fréquents ont une identité moyenne de 80 % => Pas concluant non plus

20 Alignement des LR entre eux pour détecter les régions correctes

=> Très long (+ de 3h avec 8 threads) et peu d'alignements. Les séquences déduites des alignements ne s'alignent pas toutes sur le gén. ref., et celles pouvant s'y aligner ont une identité d'environ 92 % seulement

21 Créer des SR à partir des LR

Idée : Récupérer les "k-mers" de taille 150-250 des LR, les corriger, et utiliser ces k-mers comme des SR dans les méthodes hybrides => Pas concluant, beaucoup trop de k-mers, fichier de 87Go

22 Comparaison k-mers espacés LR / SR, sans seuil de fréquence

Espacement : Un trou de longueur 0 à 10 au milieu

=> ! ! ! FAUX ! ! ! Car les k-mers espacés de GkAmpi n'étaient pas calculés en canoniques. Mais beaucoup trop de k-mers sans seuil de fréquence, donc résultats non corrigés

22.1 16-mers espacés LR dans SR (insertions)

=> Ici, on supprime des nucléotides des k-mers des LR (qui sont erronés), pour mimer des corrections aux erreurs d'insertion. Le fichier fasta produit contient donc des k-mers corrects, dans lesquels les erreurs d'insertion ont été corrigées.

1 577 603 428 16-mers espacés au total dans les LR
10 859 736 16-mers espacés des LR dans les SR
12 337 928 16-mers contigus dans les SR

22.2 16-mers espacés SR dans LR (suppressions)

=> Ici, on supprime des nucléotides des k-mers des SR (qui sont corrects) pour mimer des erreurs de suppression. Le fichier fasta produit contient donc des k-mers erronés, dans lesquels ont été introduits des erreurs de suppression.

124 643 012 16-mers espacés au total dans les SR
29 020 215 16-mers espacés des SR dans les LR
300 818 977 16-mers contigus dans les LR

Parmi les 29M de 16-mers espacés des SR présents dans les LR, seuls 5M sont bien présents dans les SR (les autres étant erronés, à cause de l'ajout des erreurs de suppression)
=> Permet de détecter les k-mers victimes d'erreurs de suppression dans les LR, mais pas de récupérer les "bons" k-mers originaux

22.3 Union insertions / suppressions (16-mers LR dans SR U 16-mers SR dans LR)

34 457 930 16-mers au total
10 859 736 16-mers trouvés dans les SR

12 337 928 16-mers contigus dans les SR

=> Impossible de récupérer tous les 16-mers des SR, à cause des 16-mers espacés des SR trouvés dans LR, qui contiennent des erreurs de suppression.

22.4 16-mers espacés communs SR et LR

1 519 006 173 16-mers espacés dans les LR
113 958 391 16-mers espacés dans les SR
98 487 415 16-mers communs

23 Comparaison k-mers espacés LR / SR, seuil de fréquence : au moins 5

Espacement : Un trou de longueur 0 à 10 au milieu

23.1 16-mers espacés LR dans SR (insertions)

=> Ici, on supprime des nucléotides des k-mers des LR (qui sont erronés), pour mimer des corrections aux erreurs d'insertion. Le fichier fasta produit contient donc des k-mers corrects, dans lesquels les erreurs d'insertion ont été corrigées.

28 462 918 16-mers espacés au total dans les LR
2 565 388 16-mers espacés des LR dans les SR
3 536 267 16-mers contigus dans les SR

23.2 16-mers espacés SR dans LR (suppressions)

=> Ici, on supprime des nucléotides des k-mers des SR (qui sont corrects) pour mimer des erreurs de suppression. Le fichier fasta produit contient donc des k-mers erronés, dans lesquels ont été introduits des erreurs de suppression.

34 457 718 16-mers espacés au total dans les SR
2 873 925 16-mers espacés des SR dans les LR
3 865 005 16-mers contigus dans les LR

Parmi les 2.8M de 16-mers espacés des SR présents dans les LR, seuls 2.5M sont bien présents dans les SR (les autres étant erronés, à cause de l'ajout des erreurs de suppression) => Permet de détecter les k-mers victimes d'erreurs de déletion dans les LR, mais pas de récupérer les "bons" k-mers originaux

23.3 Union insertions / suppressions (16-mers LR dans SR U 16-mers SR dans LR)

2 903 918 16-mers au total
2 565 388 16-mers trouvés dans les SR
3 536 267 16-mers contigus dans les SR

=> Impossible de récupérer tous les 16-mers des SR, à cause des 16-mers espacés des SR trouvés dans LR, qui contiennent des erreurs de suppression

23.4 16-mers espacés communs SR et LR

25 019 925 16-mers espacés dans les LR
31 103 857 16-mers espacés dans les SR
17 879 288 16-mers communs

24 Comparaisons k-mers, k-mers "étendus", LR dans SR (suppressions) + k-mers espacés, LR dans SR (insertions)

=> Ici, on insert des nucléotides en plus dans les k-mers des LR (qui sont erronés), pour mimer des corrections aux erreurs de suppression. Le fichier fasta produit contient donc des k-mers corrects, dans lesquels les erreurs de suppression ont été corrigées.

Par ex, pour le 4-mer AAAA et le seed 11011, on obtiendra les quatre 5-mers suivants : AAAAA, AACAA, AAGAA, AATAA.

24.1 16-mers, un trou d'emplacement variable, freq = 5

=> En ne gardant qu'un trou, mais en le déplaçant (ex : 1011, 1101), on obtient des k-mers différents

243 547 316 16-mers étendus au total dans les LR
2 440 934 16-mers étendus des LR dans les SR
3 536 267 16-mers contigus dans les SR

En faisant l'union des 16-mers "étendus" obtenus ici, et des 16-mers espacés des LR, avec un trou de taille fixe et d'emplacement variable :

Trou de longueur 1 :
x xxx xxx 16-mers espacés / étendus dans les LR
3 092 709 16-mers trouvés dans les SR
3 536 267 16-mers contigus dans les SR

Trou de longueur 1 à 2 :
x xxx xxx 16-mers espacés / étendus dans les LR
3 461 162 16-mers trouvés dans les SR
3 536 267 16-mers contigus dans les SR

=> Tout les k-mers sont quasiment trouvés.

24.2 20-mers, un trou d'emplacement variable, freq = 5

Trou de longueur 1 :
2 728 200 20-mers trouvés dans les SR
3 553 127 20-mers contigus dans les SR

Trou de longueur 1 à 2 (canonique GkAmpi) :
3 117 367 20-mers trouvés dans les SR
3 553 127 20-mers contigus dans les SR

=> Pas trop mal

24.3 Étude du GC-content des k-mers non trouvés

24.3.1 16-mers

=> À REFAIRE
Trou de longueur 1 :
256 201 432 16-mers non trouvés dans les SR
6 509 040 16-mers majoritairement GC
72 685 961 16-mers majoritairement AT
177 006 431 16-mers proportions semblables

Trou de longueur 1 à 2 :
Blabla

24.4 16-mers R9.4

Trou de longueur 1 :
280 061 162 16-mers espacés / étendus dans les LR
2 052 493 16-mers trouvés dans les SR
3 536 267 16-mers contigus dans les SR

Trou de longueur 1 à 2 :
1 158 136 516 16-mers espacés / étendus dans les LR
3 127 883 16-mers trouvés dans les SR
3 536 267 16-mers contigus dans les SR

25 Fusion des index Jellyfish (contigus / espacés / étendus) et étude des fréquence

Moins de 100 (sous-représenté) :
Trop de k-mers

100 à 1000 (autres) :
387 380 k-mers found
2 197 274 k-mers unfound

Plus de 1000 (sur-représentés) :
109 k-mers found
3 944 k-mers unfound

50 à 100 :
1 421 721 k-mers found
28 885 718 k-mers unfound

500 à 1000 :
Trop peu de k-mers

100 à 500 :

386 860 k-mers found
2 188 436 k-mers unfound

1 à 5 :
Trop de k-mers

5 à 10 :
Trop de k-mers

25 à 50 :
Trop de k-mers

26 Assemblage des LR bruts avec Canu

Impossible avec les LR bruts, trop de LR de taille < 1000.
=> En les filtrant, on arrive à une taille moyenne de 9 961, mais Canu ne parvient toujours pas à calculer d'overlaps

27 Récupération des LR s'alignant avec "bonne" identité

90% => Trop peu de LR (700), ne couvrent pas tout le gen ref

70% => 3k LR, couvrent 99,99% du gen ref : seuil retenu

27.1 Extraction des 16-mers contigus / étendus / espacés de ce sous ensemble

On extrait ensuite les 16-mers (contigus, sans seuil de fréquence) et on cherche dans les bons 16-mers des SR (nb : 3.5M) :

Avec tous les LR, 300 818 977 16-mers au total :

3 330 620 k-mers found
297 488 357 k-mers unfound

Avec seulement les LR avec une bonne identité, 25 259 623 16-mers au total (sans seuil fréquence) :

2 907 636 k-mers found
22 351 987 k-mers unfound

Avec seulement les LR avec une bonne identité, 3 576 573 16-mers au total (freq 2) :

2 223 699 k-mers found
1 352 874 k-mers unfound

Avec tous les LR, freq 5, 3 865 005 16-mers au total :

2 535 395 k-mers found
1 329 610 k-mers unfound

=> Semble plus simple / plus efficace de simplement mettre un seuil sur le nombre de 16-mers, on en obtient moins au total, mais avec une proportion similaire bien identifiée aux SR

=> En fait, les résultats obtenus avec les 16-mers contigus ne sont pas si mauvais. On ne s'en rendait pas compte avant car on cherchait à identifier les 16-mers des LR à tous les 16-mers des SR (nb : 12M), dont ceux contenant des erreurs. En terme de proportions, on parvenait donc à identifier seulement 25-50% des 16-mers des SR, mais on identifiait bien

une grande partie des bons. Mais dans tous les cas, espacer les k-mers permet de récupérer les infos qui manquent dans les k-mers cotigus.

28 Extraction des MAW des SR, et éliminations des LR contenant ces MAW

=> Pourrait potentiellement speed-up HG-CoLoR en virant les LR de très mauvais qualité ? Notamment au niveau du mapping

=> Besoin de trouver comment rechercher ces MAW dans les LR, car on ne peut pas indexer les LR avec PgSA (pas tous de même longueur), et GKA / CgKA sont trop lents à construire l'index

29 Extraction des MAW des LR, et élimination de k-mers

=> 13M de MAWs dans les LR

Ceux avec une haute fréquence d'apparition (1k-10k) sont assez courts, et sont des séquences apparaissant correctement dans les SR / le gen ref, mais n'apparaissant pas dans grand nombre de LR, probablement à cause des nombreuses erreurs

Ceux avec une faible fréquence d'apparition (< 1k) sont plus longs et sont, pour beaucoup, des séquences n'apparaissant pas dans les SR / le gen ref

=> Peut permettre de se débarrasser des mauvais k-mers, éventuellement sans utiliser les SR ?

29.1 MAW fréquents

Pour les tests suivants, on élimine les MAW inclus dans d'autres. meanLength d'un MAW = 8.96 , nb de MAWs : 202 373

En ne conservant que les k-mers contenant au moins un MAW :
=> Pas concluant, presque tous les k-mers contiennent au moins un MAW

En filtrant les MAW trop courts (len < 9) => seuls 3k mauvais k-mers des LR ne contiennent aucun MAW
=> En montant la len minimale (même à 10) => trop peu de MAWs, donc trop de k-mers ne contiennent aucun MAW, et on passe à côté de beaucoup de bons k-mers

En regardant le nombre de MAW présent dans chaque k-mer, après filtration des MAW de len < 9 :
Recherche de MAW dans les 16-mers des SR : 6.8 MAW par 16-mer en moyenne
Recherche de MAW dans tous les 16-mers des LR : 7.09 par 16-mer en moyenne
Recherche de MAW dans les mauvais 16-mers des LR : 7.2 par 16-mer en moyenne
Recherche de MAW dans les bons 16-mers des LR : 7.03 par 16-mer en moyenne
=> Convserver les MAW + courts ne change rien à ces propotions

En regardant la couverture (i.e. nombre moyen de bases couvertes) des k-mers par les MAW : Pas concluant non plus, bons et mauvais k-mers sont couverts à peu près pareil

=> Malgré tout, les MAW apportent de l'information ? Car ils sont tous présents dans les SR

=> En fait non, car les MAW fréquents sont trop courts (avLen = 8.96) et que presque tous les 8-9-10-11-mers sont présents dans le GR et dans les SR. Besoin d'identifier des choses plus grandes pour pouvoir identifier les bons / mauvais k-mers.
=> Jusqu'à k = 11, au moins 61% des k-mers possibles sont bien présents dans les SR / dans le GR

=> Étudier les MAWs non fréquents, qui, eux, sont plus longs et ne sont pour beaucoup par présents dans les SR / le GR ?

29.1.1 MAW non fréquents

freq <= 10 : 13M MAWs, avLen = 13.393
=> En filtrant les MAWs de len < 13 : 4M MAWs, 435 637 présents dans les SR / le gen ref
=> En filtrant les MAWs de len < 14 : 1.7M MAWs, 66 357 présents dans les SR / le gen ref

freq <= 5 : 13M MAWs, avLen = 13.4457
=> En filtrant les MAWs de len < 13 : 4M MAWs, 428 015 présents dans les SR / le gen

ref
=> En filtrant les MAWs de len < 14 : 1.7M MAWs, 65 307 présents dans les SR / le gen ref

freq = 1 : 10M MAWs, avLen = 14.1485
=> En filtrant les MAWs de len < 13 : 3.6M MAWs, 357 534 présents dans les SR / le gen ref
=> En filtrant les MAWS de len < 14 : 1.6M MAWs, 60 516 présents dans les SR / le gen ref
=> En filtrant les MAWs de len < 15 : 788K MAWs, 9 318 présents dans les SR / le gen ref
=> En filtrant les MAWs de len < 16 : 473K MAWs, 2 601 présents dans les SR / le gen ref
=> En filtrant les MAWs de len < 17 : 322K MAWs, 1 483 présents dans les SR / le gen ref
=> En filtrant les MAWs de len < 18 : 242K MAWs, 1 122 présents dans les SR / le gen ref
=> En filtrant les MAWs de len < 20 : 147K MAWS, 743 présents dans les SR / le gen ref

=> freq = 1 et len >= 20 semble bien. Test de différenciation bon / mauvais k-mer (ici, si un MAW est dans un k-mer, le k-mer est mauvais), avec des 32-mers, apparaissant au moins 2 fois :

Ne donne toujours rien, très peu de 32-mers contiennent un MAW, + 1/4 des 32-mers contenant un MAW sont en fait présents dans les SR (probablement à cause du peu de MAWs de l'ensemble qui sont bien présents dans les SR)
=> Test discrimination avec 1 apparition et len 13 : Permet d'écarter 2.5M de 32-mers, parmi ces 32-mers écartés, 1M sont bien dans les SR (et parmi les 1M considérés comme bons, dans lesquels on ne trouve pas de MAW, 500K sont mauvais)
=> Test discrimination avec 1 apparition et len 14 : Permet d'écarter 650K de 32-mers, parmi ces 32-mers écartés, 250K sont bien dans les LR
=> Test discrimination avec 1 apparition et len 15-18 : Pareil que pour len = 20, permet d'écarter trop peu de 32-mers

En regardant le nombre de MAW présent par 32-mer : Impossible de différence bon / mauvais 32-mers, environ autant de MAW par 32-mer dans les bons que dans les mauvais 32-mers

En regardant la couverture : Idem

=> Semble également difficile de déduire qqch des MAWs non fréquents

30 Comparaison de LR bruts avec Compareads

=> Pour identifier les LR provenant d'une même région du GR ?

Prodécure de test :

— Sélectionner un LR
— Récupérer ses LR similaires
— Recherche les LR corrigés correspondant
— Vérifier qu'ils se mappent bien sur la même région du GR.

Résultat :

En demandant deux 20-mers communs => 24 LRs similaires
Récupération des LRs corrigés correspondants : 11 LR récupérés, 8 se mappent effectivement à peu près sur la même région

=> Intéressant, pour ensuite faire une étude de k-mers / MAWS / consensus ?

31 Bugs dans PgSA

Pour un index donné, quand on lance une requête avec certains reads => seg fault

32 Problème avec NaS

=> Pas les mêmes résultats entre les fichiers dispo sur le site du Génoscope, et ceux obtenus à la main

33 Extraction de bons k-mers, puis construction de (2*k)-mers avec DBG

33.1 16-mers

=> Extraction de tous les 16-mers espacés possibles de LR avec un seul trou se déplaçant

=> Même en montant très haut le seuil de fréquence nécessaire à la conservation d'un k-mer, beaucoup de k-mers extraits sont erronés (non présents dans le gen ref)
=> Notamment, en montant à 100, on trouve plus que de k-mers erronés que de k-mers corrects

=> Quand on essaye de construire les 32-mers à partir des 16-mers avec un DBG, on en construit donc beaucoup plus qu'il n'y en a dans le gen ref, et l'identité moyenne est donc très mauvaise (96%)

33.1.1 sur le gen ref

=> Quand on extrait les 16-mers du gen ref, et que l'on essaye de construire les 32-mers, on en obtient un peu trop, mais une identité moyenne de 99,82%

=> Probablement possible de corriger le soucis en jouant avec les fréquences des k-mers extraits et / ou des k-mers construits

33.2 8-mers

=> Impossible de construire l'index de PgSA

34 k-mers uniques gen ref

34.1 k-mers ADP1

34.1.1 64

99.45 % uniques

34.1.2 32

99.39 % uniques

34.1.3 24

99.37 % uniques

34.1.4 20

99.35 uniques

34.1.5 16

99.12 % uniques

35 HALC

Sur ADP1 : Corrige plus de LR, ne prend que 4h, mais pire qualité que nous (96.61 au mieux)
Sur Coli : Corrige autant de LR, ne prend que 26min, mais légèrement pire qualité que nous (99.50 au mieux)
Sur Yeast : Corrige plus de LR, ne prend que 3h, mais pire qualité que nous (98.45)

=> + dans tous les cas, HALC produit des reads soit de bonne qualité mais très courts (pour le split) soit longs mais de mauvaise qualité (corrected et trim)

=> Pas intéressant ?

36 Etude nombre k-mers SR / cor SR / gen ref

Étude du nombre de 16-mers :
- 12M dans les SR bruts
- 3.5M dans le gen ref
- 7.1M dans les SR corrigés

Étude du nombre de 32-mers :
- 17M dans les SR bruts

- 3.5M dans le gen ref
- 7.1M dans les SR corrigés

Étude du nombre de 64-mers :
- 22M dans les SR bruts
- 3.5M dans le gen ref
- 7.1M dans les SR corrigés