



**HAL**  
open science

# Evaluation des logiciels d'assemblage utilisant des lectures longues

Laurent Bouri, Dominique Lavenier

► **To cite this version:**

Laurent Bouri, Dominique Lavenier. Evaluation des logiciels d'assemblage utilisant des lectures longues. [Rapport de recherche] RT-0475, INRIA Rennes - Bretagne Atlantique. 2016. hal-01282892v2

**HAL Id: hal-01282892**

**<https://inria.hal.science/hal-01282892v2>**

Submitted on 22 Nov 2016

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



# Evaluation des logiciels d'assemblage utilisant des lectures longues

Laurent Bouri, Dominique Lavenier

**TECHNICAL  
REPORT**

**N° 475**

March 2016

Project-Team Genscale

ISRN INRIA/RT--475--FR+ENG

ISSN 0249-0803





## Evaluation des logiciels d'assemblage utilisant des lectures longues

Laurent Bouri \*, Dominique Lavenier †

Équipe-Projet Genscale

Rapport technique n° 475 — March 2016 — 41 pages

**Résumé :** Ce rapport compare plusieurs programmes d'assemblage de génomes qui utilisent les technologies de séquençage de 3ème génération (longues lectures). Les expérimentations ont été faites sur 4 génomes de référence et les résultats évalués avec le logiciel QAST. Les 11 logiciels d'assemblage évalués sont : Celera Assembler, Falcon, Miniasm, Newbler, SGA Assembler, Smartdenovo, Abruijn, Ra, DBG2OLC, Spades et Cerulean. Les 8 premiers n'utilisent que des longues lectures tandis que les 3 derniers mixent longues et courtes lectures.

**Mots-clés :** séquençage de 3ème génération, lectures longues, assemblage hybride

---

\* CNRS Engineer/ France génomique

† CNRS Research Director, GenScale team leader

**RESEARCH CENTRE  
RENNES – BRETAGNE ATLANTIQUE**

Campus universitaire de Beaulieu  
35042 Rennes Cedex

## Evaluation of assembly software based on long reads

**Abstract:** This report compares several genome assembly software that use 3rd generation sequencing technology (long reads). The experimentations have been performed on 4 reference genomes and the results evaluated with the QUASt software. The 11 software that have been evaluated are: Celera Assembler, Falcon, Miniasm, Newbler, SGA Assembler, Smartdenovo, ABruijn, Ra, DBG2OLC, Spades and Cerulean. The first 8 software only use long reads, while the 3 last software can merge long and short reads

**Key-words:** 3rd generation sequencing, long reads, hybrid assembly

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Contexte . . . . .	5
1.2	Assembleurs testés . . . . .	5
1.2.1	Assembleurs LRO ( <i>Long Read Only</i> ) . . . . .	5
1.2.2	Assembleurs SLR ( <i>Short and Long Read</i> ) . . . . .	6
1.3	Méthodologie . . . . .	6
1.3.1	Correction des lectures . . . . .	6
1.3.2	Jeux de données . . . . .	6
1.3.3	Ressources matérielles . . . . .	6
1.3.4	Evaluation des assemblages . . . . .	7
<b>2</b>	<b>Les assembleurs LRO (<i>Long Read Only</i>)</b>	<b>8</b>
2.1	Celera Assembler . . . . .	8
2.2	Falcon . . . . .	10
2.3	Miniasm . . . . .	13
2.4	Newbler . . . . .	14
2.5	SGA Assembler . . . . .	16
2.6	Smartdenovo . . . . .	18
2.7	Abruijn . . . . .	19
2.8	Ra . . . . .	20
<b>3</b>	<b>Les assembleurs SLR (<i>Short and Long Read</i>)</b>	<b>21</b>
3.1	DBG2OLC . . . . .	21
3.2	Spades . . . . .	24
3.3	Cerulean . . . . .	26
<b>4</b>	<b>Résultats des assemblages</b>	<b>28</b>
4.1	Génomes utilisés pour les tests . . . . .	28
4.2	Evaluation des assemblages . . . . .	28
4.3	Test 1 : Acinetobacter sp, ADP1, run5 ; Minion 10x . . . . .	29
4.4	Test 2 : Acinetobacter sp, ADP1, run6 ; Minion 20x . . . . .	30
4.5	Test 3 : Escherichia coli k-12, reads Pacbio 10x (P4-C2) . . . . .	31
4.6	Test 4 : Escherichia coli k-12, reads Pacbio 100x (P4-C2) . . . . .	32

4.7	Test 5 : Escherichia coli k-12, reads Pacbio 10x (P6-C4) . . . . .	33
4.8	Test 6 : Escherichia coli k-12, reads Pacbio 100x (P6-C4) . . . . .	34
4.9	Test 7 : Escherichia coli k-12, reads Minion 20x . . . . .	35
4.10	Test 8 : Saccharomyces cerevisiae W303, reads Pacbio 10x (P4-C2) . . . . .	36
4.11	Test 9 : Saccharomyces cerevisiae W303, reads Pacbio 100x (P4-C2) . . . . .	37
4.12	Test 10 : Saccharomyces cerevisiae W303, reads Minion 20x . . . . .	38
4.13	Test 11 : Caenorhabditis elegans, reads Pacbio 10x (P6-C4) . . . . .	39
4.14	Test 12 : Caenorhabditis elegans, reads Pacbio 100x (P6-C4) . . . . .	40

# 1 Introduction

## 1.1 Contexte

Les nouvelles technologies de séquençage proposées par Pacbio Science et Oxford Nanopore Technologies génèrent des lectures (*reads*) qui peuvent dépasser 10Kbp et qui peuvent ainsi avantageusement être utilisées pour l'assemblage des génomes. En effet, la grande taille des lectures peut faciliter l'assemblage des régions répétées. Cependant, ces lectures longues possèdent des taux d'erreur compris entre 10% et 15%, ce qui nécessite souvent une phase préalable de correction avant le processus d'assemblage.

On peut répertorier deux grandes familles d'assembleurs s'appuyant sur les lectures longues :

- les assembleurs LRO ;
- les assembleurs SLR.

Les assembleurs LRO prennent uniquement en entrée des lectures longues (LRO, *Long Reads Only*). Les assembleurs SLR (*Short and Long Reads*) demandent à la fois des lectures courtes et des lectures longues.

Certains assembleurs LRO demandent que les lectures longues aient été préalablement corrigées. Des logiciels de correction sont disponibles et reposent principalement sur deux stratégies. La première fait intervenir des lectures courtes, type Illumina, dont le taux d'erreur est nettement plus faible et qui servent de base à la correction des lectures longues. L'autre stratégie ne nécessite que des longues lectures et les aligne directement entre elles pour les corriger.

## 1.2 Assembleurs testés

### 1.2.1 Assembleurs LRO (*Long Read Only*)

Les 8 assembleurs LRO testés sont répertoriés dans le tableau ci-dessous. Nous indiquons les assembleurs qui demandent une correction préalable des lectures.

Assembleurs	lectures non corrigées	lecture corrigées
Celera [al12b]	non	oui
Falcon [Chi16]	oui	oui
Miniasm [Li15]	oui	oui
Newbler [al05]	non	oui
SGA [Sim11]	oui	oui
Smartdenovo	oui	oui
Abruijn [al16b]	oui	oui
Ra [al16a]	oui	oui

De manière générale, la stratégie de ces assembleurs consiste à produire des alignements multiples, puis à calculer le meilleur graphe de chevauchement et, enfin, à générer les séquences consensus des contigs à partir du graphe. l'assemblage est d'autant plus efficace que les lectures présentent un taux d'erreur faible.



### 1.2.2 Assembleurs SLR (*Short and Long Read*)

Les 3 assembleurs SLR testés sont les suivants :

- DBG2OLC [al15]
- Spades [al12a]
- Cerulean [al13b]

Schématiquement, les pipelines d'assemblage qui utilisent des lectures longues et des lectures courtes fonctionnent de la manière suivante : un pré-assemblage (production de contigs) est d'abord effectué à partir des lectures courtes, puis les lectures longues servent à améliorer ce pré-assemblage : fermeture des gaps, résolution des régions répétées, etc.

## 1.3 Méthodologie

### 1.3.1 Correction des lectures

Lorsqu'une correction des lectures est nécessaire, le logiciel LORDEC [Riv14] est utilisé. Ce dernier construit un graphe de de-Bruijn à partir des lectures courtes et produit une lecture longue corrigée en s'appuyant sur un parcours de graphe optimal.

### 1.3.2 Jeux de données

Le tableau ci-dessous indique les 4 génomes de références et les jeux de données utilisés pour les différents tests :

Génome	taille (M bp)	Test	Minion	PacBio	Illumina
Acinetobacter	3.9 M	1	10x, 3.4K reads		211K reads
		2	20x, 10K reads		211K reads
E.Coli	4.6 M	3		10x, P4C2, 36K reads	16M reads
		4		100x, P4C2, 91K reads	16M reads
		5		10x, P6C4, 8.7K reads	16M reads
		6		100x, P6C4, 87K reads	16M reads
		7	20x, 22K reads		16M reads
S. Cerevisa	11.6 M	8		10x, P4C2, 26K reads	3.8M reads
		9		100x,P4C2, 261K reads	3.8M reads
		10	20x, 47K reads		3.8M reads
C. Elegans	100 M	11		10x, P6C4, 92K reads	55M reads
		12		100x, P6C4, 740K reads	55M reads

### 1.3.3 Ressources matérielles

La plupart des assembleurs testés ayant besoin d'une importante quantité de ressources matérielles, leur évaluation est passée par la soumission de tâches sur le cluster de la plateforme Genouest.

Assembleur	Nombre de threads
Celera Assembleur	8
Falcon	2 à 24
Miniasm	16
Newbler	1
SGA assembler	1
Smartdenovo	1
Abriijn	1
Ra	1
DBG2OLC	1
Spades	16
Cerulean	8

Configuration d'un noeud du cluster :

- **Nombre de CPU** : 40
- **Fréquence d'un CPU** : 2.6 GHz
- **Mémoire disponible** : 256 Gb

#### 1.3.4 Evaluation des assemblages

L'outil QUAST[al13a] (Quality ASsessment Tool) a été utilisé. Il évalue l'assemblage en calculant diverses métriques telles que le nombre de contigs, la longueur totale du génome assemblé, le N50, le nombre et la longueur des contigs mal assemblés ainsi que la fraction du génome de référence retrouvée parmi les contigs. Cette fraction est déduite en alignant le génome assemblé au génome de référence grâce au logiciel MuMmer [al99].

## 2 Les assembleurs LRO (*Long Read Only*)

### 2.1 Celera Assembler

#### Introduction

*Celera Assembler* est un assembleur de Type OLC (overlap-layout-consensus), originellement utilisé avec des données Sanger. Il supporte à présent les lectures longues et les lectures de type 454.

Site web : <http://wgs-assembler.sourceforge.net>

#### Installation

*Celera Assembler* peut être télé-chargé sous forme de code source à installer, ou de binaires pré-compilés.

Compilation et installation du code source

```
$ bzip2 -dc wgs-8.3rc2.tar.bz2 | tar -xf
$ cd wgs-8.3rc2
$ cd kmer && make install && cd ..
$ cd src && make && cd ..
$ cd ..
```

Extraction des fichiers binaires pré-compilés

```
$ bzip2 -dc wgs-8.3rc2-*.tar.bz2 | tar -xf
```

#### Format des données d'entrée

*Celera Assembler* requiert en entrée des fichiers fragments « frg » et un fichier contenant les options spécifiques à l'assemblage (non obligatoire). Des programmes tels que **fastatoCA** et **fastqtoCA** sont disponibles dans le package WGS pour effectuer des conversions à partir de formats communs tels que Fasta ou Fastq. La conversion du fichier fasta en .frg demande non seulement un fichier contenant la séquence mais aussi un fichier comportant des valeurs sur la qualité de celle-ci.

```
fastaToCA -l libraryname -s seq.fasta -q qlt.fasta > seq.frg
```

- l nom de la librairie
- s séquences au format Fasta
- q fichier renseignant sur la qualité de la séquence

```
fastqToCA -libraryname LIB -technology pacbio-corrected -reads seq.fastq > seq.frg
```

- libraryname : nom de la librairie
- technology : type de donnée (pacbio, illumina, 454,...)
- reads : séquences au format Fastq

## Usage

Le script runCA divise l'assemblage en 9 étapes. Chacune de ces étapes écrit dans un sous-dossier. *Celera Assembler* commence par charger les données des fragments dans une banque appelée « gkpStore ». A partir de chaque fragment, un histogramme de la fréquence des k-mers est généré. La valeur à partir de laquelle un k-mer seed devient non informatif et la profondeur de couverture sont alors calculées. Des alignements multiples sont ensuite effectués afin de corriger les éventuelles erreurs de séquençage et le meilleur graphe de chevauchement est calculé. Puis, les séquences consensus des contigs sont déduits du graphe. Enfin, *Celera Assembler* finit par une étape de scaffolding.

```
runCA -d directory -p prefix -s specfile <option=value> ... <input-files>
```

- d dossier où stocker les résultats
- p préfixe servant à nommer les fichiers de résultats
- s fichier contenant des options spécifiques à l'assemblage (non obligatoire)

## Messages d'erreur courants

```
Overlap job /root/wgs/Linux-amd64/bin/results/1-overlapper/001/000001  
FAILED. 1 overlapper jobs failed
```

**solution :** modifier l'option merSize afin de réduire la taille des graines utilisées par l'algorithme « seed and extend ».

---

```
1 unitig consensus jobs failed;  
remove /root/wgs/Linux-amd64/bin/results5/5-consensus/consensus.sh to try again
```

**solution :** supprimer le fichier consensus.sh, puis réessayer.

---

```
BEGIN failed-compilation aborted at  
/data/bill.crosby/apps/wgs-8.3rc1/Linux-amd64/bin/caqc.pl http ://caqc.pl/ ; line 18
```

**solution :** installer le module perl manquant : `sudo cpan Statistics::Descriptive`

## Données de sortie

La sortie de *Celera Assembler* comporte 9 sous-répertoires, un fichier ASM qui contient la description précise de l'assemblage avec notamment les séquences des scaffolds générés et un fichier de contrôle de qualité qui fournit quelques données statistiques sur les résultats. Ce dernier renferme notamment des informations sur les scaffolds (N50, ...), les lectures (quantité, ...) et les contigs (nombre de gaps, ...).

Le format de fichier ASM est le format de sortie standard de *Celera Assembler*. Cependant, il est possible de convertir le fichier au format FASTA. Pour ce faire, le package WGS-assembly contient le module asmOutputFasta.

```
asmOutputFasta -p prefix < output.asm
```

## 2.2 Falcon

### Introduction

*Falcon* est un assembleur de-novo, utilisant des lectures provenant des technologies Pacbio ou Minion. En effet, l'alignement de lecture longues servent à la construction d'un consensus, puis à l'assemblage du génome. *Falcon* est annoncé comme un assembleur de génomes diploïdes. Il est conseillé d'avoir une couverture de 100x de lectures Pacbio ou Minion pour un assemblage de-novo.

Site internet : <https://github.com/PacificBiosciences/FALCON>

### Installation

Falcon peut être télé-chargé sous forme de source code qu'il faut installer sur un cluster. L'installation requiert gcc (4.8.3+) et python (2.7+). Falcon a également besoin des sous-modules suivants : pypeFLOW, DAZZ\_DB et DALIGNER.

Création d'un environnement virtuel Python

```
$ FC=fc_env
$ virtualenv -no-site-packages -always-copy $FC
$ . $FC/bin/activate
```

Télé-chargement et installation de Falcon et des sous-modules

```
$ git clone git://github.com/PacificBiosciences/FALCON-integrate.git
$ cd FALCON-integrate
$ git submodule update --init
$ cd pypeFLOW
$ python setup.py install
$ cd ..
$ cd FALCON
$ python setup.py instal
$ cd ..
$ cd DAZZ_DB/
$ make
$ cp DBrm DBshow DBsplit DBstats fasta2DB $FC/bin/
$ cd ..
$ cd DALIGNER
$ make
$ cp daligner daligner_p DB2Falcon HPCdaligner LA4Falcon LAmerge LAsort $FC/bin
$ cd ..
```

### Format des données d'entrée

*Falcon* nécessite un fichier de configuration pour démarrer, nommé "fc\_run.cfg", répertoriant les lignes de commandes à lancer, sur quel noeud du cluster, avec quelles ressources et avec quelles données. Voici un exemple de fichier de configuration *Falcon* :

```
[General]
#job_type = local

# list of files of the initial bas.h5 files
input_fofn = input.fofn
#input_fofn = preads.fofn

input_type = raw
#input_type = preads

# The length cutoff used for seed reads used for initial mapping
length_cutoff = 12000

# The length cutoff used for seed reads usef for pre-assembly
length_cutoff_pr = 12000

jobqueue = your_queue
sge_option_da = -pe smp 8 -q %(jobqueue)s
sge_option_la = -pe smp 2 -q %(jobqueue)s
sge_option_pda = -pe smp 8 -q %(jobqueue)s
sge_option_pla = -pe smp 2 -q %(jobqueue)s
sge_option_fc = -pe smp 24 -q %(jobqueue)s
sge_option_cns = -pe smp 8 -q %(jobqueue)s

pa_concurrent_jobs = 32
ovlp_concurrent_jobs = 32

pa_HPCdaligner_option = -v -dal24 -t16 -e.70 -l1000 -s1000
ovlp_HPCdaligner_option = -v -dal24 -t32 -h60 -e.96 -l500 -s1000

pa_DBsplit_option = -x500 -s200
ovlp_DBsplit_option = -x500 -s200

falcon_sense_option = --output_multi --min_idt 0.70 --min_cov 4 --local_match_count
_threshold 2 --max_n_read 200 --n_core 6 --output_dformat

overlap_filtering_setting = --max_diff 100 --max_cov 100 --min_cov 20 --bestn 10 --n_core 24
```

Les paramètres "input-fofn" et "jobqueue" doivent obligatoirement être renseignés avant de lancer une tâche sur le cluster. Le fichier ".fofn" contient, à chaque ligne, le chemin vers un jeu de données de lectures longues au format Fasta.

## Usage

Avec l'environnement virtuel activé, lancez la commande suivante pour démarrer l'assembleur *Falcon* :

```
$ fc_run.py fc_run.cfg
```

La suite d'étapes suivantes est alors effectuée :

- Alignements entre les lectures longues (DALIGNER)
- Pre-assemblage et correction des erreurs
- Détection des chevauchements entre les lectures corrigées
- Filtrage des chevauchements
- Construction d'un graphe à partir des chevauchements
- Construction de contigs à partir du graphe

Enfin, si le paramètre “input-type” du fichier de configuration est positionné à la valeur “preads” au lieu de “raw”, *Falcon* considérera que les lectures sont déjà corrigées.

### Messages d'erreur courants

*fasta2DB : Could not find file*

**Solution :** Changer les droits d'accès afin que fasta2DB puisse ouvrir le fichier fasta

---

*Pacbio header line name inconsistent*

**Solution :** Utiliser un script qui modifiera les header afin d'être identiques à des headers de données Pacbio

### Données de sortie

Le dossier “2-asm-falcon” contient les données du graphe ainsi que les contigs issus de l'assemblage. Le fichier contenant l'assemblage final se nomme “p-ctg.fa”

## 2.3 Miniasm

### Introduction

*Miniasm* est un assembleur de type OLC, capable d'assembler un génome en un temps très court à partir de lectures Pacbio ou Minion, corrigées ou non. *Miniasm* ne contient pas d'étapes de consensus ou de correction des lectures. Il aligne toutes les lectures entre elles, puis créer un graphe d'assemblage.

Site internet : <http://github.com/lh3/miniasm>

### Installation

*Miniasm* peut être télé-chargé sous forme de source code qu'il faut ensuite installer.

```
$git clone https://github.com/lh3/minimap && (cd minimap && make)
$git clone https://github.com/lh3/miniasm && (cd miniasm && make)
```

### Format des données d'entrée

*Miniasm* a besoin d'un fichier contenant les données d'alignement entre lectures obtenu avec le logiciel *Minimap*. Ce dernier comporte l'extension `.paf.gz`. Les lectures peuvent être au format Fasta ou Fastq.

### Usage

Dans un premier temps, *Minimap* trouve les positions de mapping des lectures entre elles. *Miniasm* se sert ensuite de ce résultat (contenu dans le fichier `.paf.gz`) pour créer un graphe d'assemblage au format `.gfa`.

```
$minimap/minimap -Sw5 -L100 -m0 -t8 reads.fq reads.fq | gzip -1 > reads.paf.gz
```

- `S` ne pas prendre en compte les lectures qui mappent contre elles mêmes
- `w` taille de fenêtre du *minimizer*
- `L` longueur minimale d'un match
- `m` fusionner deux chaînes si un pourcentage donné est partagé entre *minimizer*
- `t` nombre de threads à utiliser

```
$miniasm/miniasm -f reads.fq reads.paf.gz > reads.gfa
```

### Données de sortie

Enfin, le script `awk` suivant permet de convertir le fichier de sortie `.gfa` au format Fasta :

```
$ awk '/^S/{print ">"$2"\n"$3}' input_file.gfa | fold > output_file.fa
```



## 2.4 Newbler

### Introduction

*Newbler* est un ensemble de scripts. Il a été spécifiquement conçu pour l'assemblage à partir de données générées par les plateformes de pyroséquençage 454 vendues par 454 Life science.

Site web : <http://www.454.com/products/analysis-software/>

### Installation

Le lien de télé-chargement des binaires de la suite *Newbler* doit être demandé sur le site web en remplissant un formulaire. Il faut ensuite télé-charger puis extraire le fichier nommé "gsNewbler-2.9-1x8664.rpm"

### Format des données d'entrée

*Newbler* n'accepte que des lectures de taille supérieure à 2000 bp. Dans le cas de lectures Minion, leur taille excédant 2000 bp, celles-ci doivent être tronquées en conservant un maximum de chevauchement entre les nouvelles séquences ainsi formées (un chevauchement de 500 pb a été appliqué pour nos expérimentations).

### Usage

Dans un premier temps, le projet doit être créé :

```
$ newAssembly projectname
```

Ensuite, les lectures Minions découpées peuvent être chargées dans le nouveau projet :

```
$ addRun -lib minion projectname run_minion.fasta
```

Enfin, la commande suivante lance l'assemblage :

```
$ runProject -mi 96 -ml 60 -sl 22 projectname
```

- **mi** pourcentage d'identité minimum
- **ml** taille minimale des chevauchements
- **sl** taille de la graine

La première phase de l'assemblage consiste à trouver des chevauchements entre les lectures. Pour des raisons de gain de temps, *Newbler* produit des graines de 16-mers pour chaque lecture. Lorsque *Newbler* trouve un chevauchement entre deux lectures, il étend le chevauchement entre les lectures jusqu'à une taille minimum (40bp par défaut), et en considérant un pourcentage d'identité minimum (90 par défaut). *Newbler* finit alors par créer un graphe, puis extrait les contigs.

Il est également possible d'utiliser la commande suivante pour lancer *Newbler* :

```
$ runAssembly -o projectname -mi 96 -ml 60 -sl 22 run_minion.fasta
```

### **Données de sortie**

Le dossier de sortie de *Newbler* contient les contigs dans le fichier nommé "454AllContigs.fna".

## 2.5 SGA Assembler

### Introduction

*SGA* est basé sur le “string graph” de *Gene Myers* et utilise la transformée de Burrows-Wheeler/Fm-index pour trouver des chevauchements entre les lectures.

Site web : <https://github.com/jts/sga>

### Installation

La compilation de *SGA assembler* requiert l’installation préalable des dépendances suivantes :

- google sparse hash library (<http://code.google.com/p/google-sparsehash/>)
- the bamtools library (<https://github.com/pezmaster31/bamtools>)
- zlib (<http://www.zlib.net/>)

Installation de SGA :

```
$ git clone https://github.com/jts/sga.git
$ cd sga/src
$ ./autogen.sh
$ ./configure {with-sparsehash=<chemin vers sparsehash>\\
--withbamtools=<chemin vers bamtools>\\
--prefix=<chemin où SGA sera installé> && make && make install
```

### Format des données d’entrée

*SGA assembler* accepte en entrée des lectures longues au format Fasta ou Fastq.

### Usage

*SGA assembler* se divise en 6 étapes :

1. une étape préliminaire à l’assemblage, écartant les lectures contenant d’autres lettres que ATGC
2. la construction d’un FM-index à partir du fichier au format Fasta ou Fastq
3. la correction éventuelle des lectures en se basant sur d’éventuels chevauchements
4. la suppression des lectures dupliquées
5. la construction d’un « String Graph » à partir des chevauchements identifiés entre les lectures
6. assemblage à partir du graphe créé à l’étape précédente.

```
$ ./sga preprocess reads.fasta
$ ./sga index -a ropebwt reads.fasta
$ ./sga correct reads.fasta
$ ./sga filter reads.fasta
$ ./sga overlap -m 17 reads.fasta
```

— `m` taille des chevauchements en paires de bases

```
$ ./sga assemble reads.filter.pass.asq.gz
```

### Messages d'erreur courants

*substring read found during overlap computation*

**Solution** : lancer le sous-programme `rmdup`

```
$ ./sga rmdup reads.fasta
```

A noter que les commandes pour construire le graphe puis effectuer l'assemblage doivent comporter respectivement les fichiers suivants en entrée :

- “reads.filter.pass.rmdup.fa”
- “reads.filter.pass.rmdup.asq.gz”

### Données de sortie

La sortie de *SGA assembler* fournit un fichier “default-contigs.fasta” contenant les séquences résultant de l'assemblage. Le nom de ce fichier peut être modifié avec l'option “-o”.

## 2.6 Smartdenovo

### Introduction

*Smartdenovo* est un assembleur de novo sans étape de correction. Il produit une séquence consensus après avoir aligné les lectures contre elles-même.

Site web : <https://github.com/ruanjue/smartdenovo>

### Installation

*Smartdenovo* requiert uniquement un système linux.

Installation de *Smartdenovo*

```
$ git clone https://github.com/ruanjue/smartdenovo.git && (cd smartdenovo; make)
```

### Format des données d'entrée

*Smartdenovo* a besoin en entrée de lectures longues au format FASTA.

### Usage

Smartdenovo comporte plusieurs outils afin de trouver des chevauchements entre les lectures, d'identifier les régions chimériques ou de faible qualité, et enfin de construire une séquence consensus. Ces outils sont appelés en lançant le script perl "smartdenovo.pl" :

```
$ smartdenovo.pl -p prefix reads.fa > prefix.mak  
$ make -f prefix.mak
```

### Données de sortie

Les séquences corrigées sont dans un fichier de type "prefix.dmo.lay.utg".

## 2.7 Abruijn

### Introduction

*Abruijn* est un assembleur de novo, actuellement en cours de développement, basé sur la création d'un graphe de A-bruijn à partir de longues lectures.

Site web : <https://github.com/fenderglass/ABruijn>

### Installation

*Abruijn* requiert un système linux et l'installation préalable de *Blasr*.

Installation de *Blasr*

```
$ git clone git://github.com/PacificBiosciences/blasr.git blasr
$ ./configure.py --no-pbbam HDF5_INCLUDE=f1 HDF5_LIB=f2
$ make blasr
```

Installation de *Abruijn*

```
$ git clone https://github.com/fenderglass/ABruijn.git
$ cd Abruijn
$ make
```

### Format des données d'entrée

*Abruijn* a besoin en entrée de longues lectures au format FASTA, ainsi que de la couverture estimée du jeu de lectures.

### Usage

Le programme *Abruijn* commence par pré-assembler les longues lectures erronés. Pour cela, il construit un graphe de Abruijn à partir des kmers solides. Abruijn cherche ensuite un chemin dans le graphe de A-bruijn, donnant ainsi un génome pré-assemblé comportant des erreurs. Pour finir, l'alignement des longues lectures contre le génome pré-assemblé avec l'outil BLASR permet l'obtention d'un assemblage corrigé du génome.

*Abruijn* se lance à l'aide de la commande suivante :

```
$ python abruijn.py -t 8 -k 16 reads.fasta <dossier de sortie> <taux de couverture>
— t nombre de threads
— k taille des kmer
```

### Données de sortie

Les séquences corrigées sont dans un fichier de type "polished.fasta", contenu dans le dossier de sortie spécifié.

## 2.8 Ra

### Introduction

*Ra* est un assembleur de novo prenant en entrée des longues lectures. Il a la particularité d'utiliser un nouveau mappeur nommé Graphmap. Néanmoins, le développement de cet outil étant encore récent, celui-ci n'inclut pas d'étape de consensus.

Site web : <https://github.com/mariokostelac/ra-integrate>

### Installation

*Ra* est disponible pour Linux et requiert ruby 2.2, make, g++ (4.8+) et graphviz. Il est également possible de lancer *Ra* via l'utilisation d'un conteneur Docker.

Compilation de *Ra* :

```
$ git clone--recursive https://github.com/mariokostelac/ra-integrate.git
$ make
```

Compilation de l'image docker :

```
$ docker pull mariokostelac/ra-integrate:master
```

### Format des données d'entrée

*Ra* a seulement besoin de longues lectures au format FASTA.

### Usage

Afin de trouver les chevauchements entre les différents lectures qui serviront ensuite à l'assemblage, l'algorithme de Graphmap est divisé en 5 étapes :

1. sélection des régions candidates en identifiant les graines entre deux séquences
2. construction d'ancres à partir des graines
3. extension des ancres en cherchant à trouver la plus longue sous séquence
4. amélioration des alignements en utilisant un modèle de régression linéaire
5. construction de l'alignement final

*Ra* se lance à l'aide de la commande suivante :

```
$ script/run reads.fa
```

### Données de sortie

Le fichier contenant les séquences assemblées au format Fasta est dans un dossier de sortie de type "assembly.number".

## 3 Les assembleurs SLR (*Short and Long Read*)

### 3.1 DBG2OLC

#### Introduction

*DBG2OLC* est un assembleur qui utilise des contigs Illumina comme points d'ancrage pour construire un graphe de chevauchement avec des lectures PacBio.

Site web : <https://github.com/yechengxi/DBG2OLC>

#### Installation

*DBG2OLC* requiert un système linux 64-bit et l'installation préalable de *Blasr* et *Sparc*. *DBG2OLC*, *Blasr* et *Sparc* peuvent être télé-chargés sous forme de code source à installer. L'installation de *Blasr* requiert hdf 1.8.12 ou plus. Il est également possible d'installer *SparseAssembler* dans le but de construire des contigs à partir de lectures courtes.

Installation de *DBG2OLC*

```
$ git clone http://git.code.sf.net/p/dbg2olc/code dbg2olc-code
$ cd dbg2olc-code
$ g++ DBG2OLC.cpp -o DBG2OLC
```

Installation de *Blasr*

```
$ git clone git://github.com/PacificBiosciences/blasr.git blasr
$ ./configure.py --no-pbbam HDF5_INCLUDE=f1 HDF5_LIB=f2
$ make blasr
```

- f1 fichier header HDF5
- f2 fichier librairie HDF5

Installation de *Sparc*

```
$ git clone http://git.code.sf.net/p/sparc-consensus/code sparc-consensus-code
$ g++ Sparc.cpp -o Sparc
```

#### Format des données d'entrée

*DBG2OLC* accepte en entrée des contigs avec, soit des lectures longues (Pacbio, Nanopore) au format Fasta, soit des lecture de type illumina au format Fasta ou Fastq. Les contigs peuvent préalablement être construits en utilisant un assembleur DBG tel que *SparseAssembler*.



## Usage

Le programme *DBG2OLC* se divise en 5 étapes :

1. Construction d'un graph de de-Bruijn et création de contigs à partir de lectures courtes (*SparseAssembler*)
2. Alignement des contigs à chaque lecture longue. Ces derniers sont compressés en listes d'ancrages
3. Exécution d'alignements multiples afin de supprimer les lectures chimériques
4. Construction d'un graphe de chevauchements en utilisant les longues lectures compressées
5. Déduction d'une séquence consensus à partir du graphe

```
$ ./DBG2OLC k 17 KmerCovTh 2 MinOverlap 20 AdaptiveTh 0.002 Contigs Contigs.txt \\  
RemoveChimera 1 f <fichier_pacbio1> f <fichier_pacbio2>
```

- *KmerCovTh* nombre de k-mer alignés à un contig pour que ce dernier soit considéré comme une ancre
- *MinOverlap* nombre de k-mer entre chaque paire de lecture pour être considéré comme un chevauchement
- *AdaptiveTh* les k-mer alignés à un contig doivent être plus large que cette valeur pour être considérés comme tels
- *k* taille des k-mer
- *Contigs* le fichier de contigs au format Fasta
- *MinLen* taille minimum d'une lecture
- *RemoveChimera* supprime les lectures chimériques dans un jeu de données.

Les 3 paramètres suivants sont critiques pour les performances : *KmerCovTh*, *MinOverlap* et *AdaptiveTh*.

Pour des données PacBio, en fonction de la couverture, il est conseillé d'utiliser les valeurs suivantes :

Couverture	10x/20x	50x/100x
<i>KmerCovTh</i>	2-5	2-10
<i>MinOverlap</i>	10-30	50-150
<i>AdaptiveTh</i>	0.001 0.01	0.01-0.02

Un fichier nommé "*backbone\_raw.fasta*" contient les scaffolds construits par *DBG2OLC* au format fasta.

Enfin, dans le but de finaliser l'assemblage, le script "*split\_and\_run*" peut être exécuté. Il faut auparavant vérifier que *Blasr* ait été défini dans la variable *PATH* et que le binaire *Sparc* soit présent dans le même dossier que le script à exécuter. Les fichiers à préciser sont :

- le fichier de scaffolds produits par *DBG2OLC* ("*backbone\_raw.fasta*")
- le fichier consensus produit *DBG2OLC* ("*DBG2OLC\_Consensus\_info.txt*")
- le fichier contenant les contigs au format Fasta
- les lectures longues au format Fasta

Il faut ensuite réunir les lectures longues et les contigs dans un seul fichier

```
$ cat Contigs.txt pb_reads.fasta > ctg_pb.fasta
```

Enfin, le script "*split\_and\_run\_sparc.sh*" se trouvant dans le dossier "utility" doit être lancé.

```
$ sh ./split_and_run_sparc.sh backbone_raw.fasta DBG2OLC_Consensus_info.txt \\
ctg_pb.fasta ./consensus_dir > consensus_log.txt
```

### Messages d'erreur courants

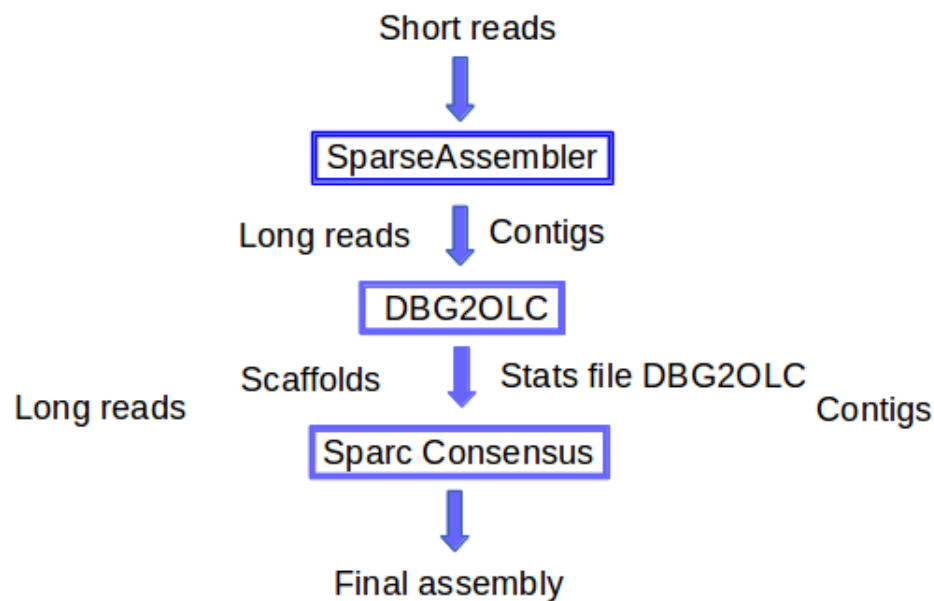
```
./split_and_run_sparc_r2.sh : 1 : eval : blasr : not found
```

**Solution :** modifier le script "*split\_and\_run\_sparc.sh*" de sorte que blasr soit reconnu.

### Données de sortie

Le script "*spli\_and\_run*" produit un fichier nommé "final\_assembly.fasta" dans le dossier *utility*, contenant la liste des scaffolds issus de l'assemblage.

### Principe de DBG2OLC



## 3.2 Spades

### Introduction

*SPAdes* (St. Petersburg genome assembler) est un assembleur à l'origine créé pour l'assemblage de petits génomes. Il supporte les lectures Illumina/IonTorrent, et possède maintenant une option pour réaliser un assemblage hybride avec des lectures longues.

site web : <http://bioinf.spbau.ru/spades>

### Installation

*SPAdes* requiert un système linux 64-bit ou Mac OS et l'installation préalable de Python. *SPAdes* peut être télé-chargé sous forme de code source ou de binaires pré-compilés.

Télé-chargement et extraction des fichiers binaires

```
$ wget http://spades.bioinf.spbau.ru/release3.6.0/SPAdes-3.6.0-Linux.tar.gz
$ tar -zxf SPAdes-3.6.0-Linux.tar.gz
$ cd SPAdes-3.6.0-Linux/bin/
```

Télé-chargement et installation du code source

```
$ wget http://spades.bioinf.spbau.ru/release3.6.0/SPAdes-3.6.0.tar.gz
$ tar -zxf SPAdes-3.6.0.tar.gz
$ cd SPAdes-3.6.0
$ ./spades_compile.sh
```

### Format des données d'entrée

*SPAdes* accepte en entrée des lectures pairées, non-pairées et *mate-pairs* au format Fasta ou Fastq. Pour les données IonTorrent, *SPAdes* prend également en charge les lectures non pairées au format BAM. Des lectures Sanger et PacBio CCS peuvent être utilisées au format Fasta ou Fastq. Il est aussi possible de rajouter des contigs pour faciliter l'assemblage.

### Usage

*SPAdes* construit un graphe de de-Bruijn à partir de k-mers de différentes tailles. Ainsi, une plus petite valeur de k dans les régions à faible couverture minimise la fragmentation, alors qu'une plus grande valeur de k dans les régions fortement couvertes réduit le nombre de contigs chimériques. Des contigs sont alors déduits à partir du graphe. Enfin, les lectures longues sont utilisées pour la fermeture des gaps et l'identification des régions répétées.

```
$ spades.py --only-assembler --nanopore <file_name> -s <file_name> \\  
--trusted-contigs <file_name> -o <output_dir>
```

- `--only-assembler` démarre uniquement le module d'assemblage
- `--nanopore` fichier de lectures longues

- `-s` fichier contenant des lectures courtes
- `--trusted-contigs` fichier de contigs
- `-o` répertoire où sont stockés les résultats

### **Données de sortie**

SPAdes produit plusieurs fichiers dans le répertoire précisé par l'option `-o` :

- "scaffolds.fasta" : fichier FASTA contenant les différents scaffolds issus de l'assemblage.
- "param.txt" : liste des paramètres utilisés
- "warning.log" : fichier de logs

### 3.3 Cerulean

#### Introduction

*Cerulean* étend les contigs assemblés à partir de lectures courtes, telles Illumina, en utilisant des lectures longues.

site web : <http://sourceforge.net/projects/ceruleanassembler/>

#### Installation

Cerulean requiert un système linux 64-bit et l'installation préalable des programmes suivants :

- Python 2.7.1
- numpy et matplotlib (bibliothèques pour python)
- Abyss assembler <http://www.bcgsc.ca/platform/bioinfo/software/abyss>
- SMRT Analysis toolkit (Blasr) : <http://pacbiodevnet.com/>
- Pbjelly : <https://sourceforge.net/projects/pb-jelly/>

Télé-chargement et extraction des scripts

```
$ wget http://sourceforge.net/projects/ceruleanassembler/files/Cerulean_v_0_1.tar.gz
$ tar -zxf Cerulean_v_0_1.tar.gz
```

#### Format des données d'entrée

*Cerulean* requiert en entrée les contigs assemblés, issus de l'assemblage de lectures courtes avec *Abyss*, ainsi que le mapping des lectures Pacbio sur ces contigs, effectué avec *Blasr*.

Assemblage à partir de reads Illumina

```
$ abyss-pe k=64 n=10 in='reads1.fastq reads2.fastq' name=${dataname}$
```

- k taille du k-mer
- n nombre de paires minimum pour la construction de contigs

2 fichiers seront alors créés :

- "<dataname>-contigs.fa", contenant la séquence des contigs
- "<dataname>-contigs.dat", contenant la structure du graphe

Mapping des lectures longues sur les contigs en utilisant *Blasr* :

```
$ blasr <dataname>_pacbio.fa <dataname>-contigs.fa -minMatch 10 -minPctIdentity 70 \\  
-bestn 30 -nCandidates 30 -maxScore 500 -nproc <numthreads> -noSplitSubreads \\  
-out <dataname>_pacbio_contigs_mapping.fasta.m4
```

- minMatch taille minimum de la graine
- minPctIdentity pourcentage d'identité minimum
- bestn afficher les n meilleurs alignements
- nCandidates afficher les n meilleurs alignements
- maxScore score maximum à afficher

- `nproc` nombre de threads utilisés
- `noSpliSubreads` les reads ne sont pas subdivisés

## Usage

*Cerulean* requiert que tous les fichiers d'entrée soient dans le même dossier :

- `<dossier>/<dataname>-contigs.fa`
- `<dossier>/<dataname>-contigs.dot`
- `<dossier>/<dataname>_pacbio_contigs_mapping.fasta.m4`

*Cerulean* est lancé avec la commande suivante :

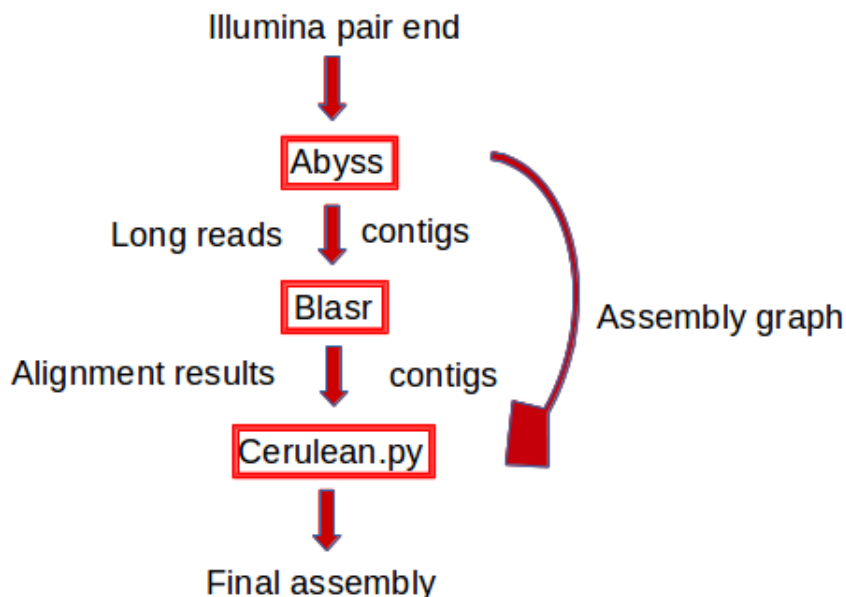
```
$ python src/Cerulean.py --dataname <dataname> --basedir <basedir> --nproc <numthreads>
```

- `dataname` nom du jeu de données
- `basedir` chemin vers le dossier contenant les trois fichiers d'entrée
- `nproc` nombre de threads utilisés

## Données de sortie

*Cerulean* génère en sortie de programme le fichier "`<dossier>_cerulean.fasta`", contenant le résultats de l'assemblage. Pour finir, les auteurs de *Cerulean* conseillent d'utiliser *PBJelly* afin de fermer les gaps restant.

## Principe de Cerulean



## 4 Résultats des assemblages

### 4.1 Génomes utilisés pour les tests

Nom du génome	Nombre de chromosomes	taille
Acinetobacter DP1	1 chromosome	3 976 747 pb
Escherichia Coli K12 MG1655	1 chromosome	4 641 652 pb
Saccharomyces Cerevisiae W303	16 chromosomes	11 633 571 pb
Caenorhabditis elegans	6 chromosomes	100 272 607 pb

### 4.2 Evaluation des assemblages

Les génomes de référence permettent d'évaluer la qualité des différents assemblages en se basant sur les résultats des métriques produits par le logiciel QUASt. Les différentes métriques sont listées ci-après :

- **# contigs (> 1000pb)** : nombre total de contigs excédant 1000pb après assemblage.
- **Largest contig** : longueur du contig le plus large de l'assemblage.
- **Total length** : nombre total de bases dans l'assemblage.
- **N50** : longueur pour laquelle la collection de tous les contigs de cette longueur ou plus grande, couvre au moins la moitié de l'assemblage.
- **# misassembled contigs** : nombre de contigs contenant un événement de mauvais assemblage. Un mauvais assemblage satisfait au moins un de ces critères :
  - les séquences adjacentes du côté gauche s'alignent à plus de 1 Kbp de différence de la séquence adjacente à droite sur la référence.
  - les séquences adjacentes se chevauchent sur plus de 1 Kbp
  - les séquences adjacentes s'alignent sur différents brins ou différents chromosomes.
- **Misassembled contigs length** : somme des longueur des différents contigs mal assemblés.
- **Genome fraction (%)** : pourcentage de bases alignées sur le génome de référence. Quast utilise alors l'outil d'alignement Nucmer issu de MUMmer v3.23.

### 4.3 Test 1 : Acinetobacter sp, ADP1, run5 ; Minion 10x

Jeu de données :

- reads Oxford NanoPort (MinIon) corrigés avec Lordec : 3427 reads
- reads Illumina (MiSeq) : 211219 reads de taille 150 pb à 3 kbp
- contigs générés par Minia à partir des reads Illumina (233 contigs)

#### Assembleurs LRO

Metrics	Celera	Falcon	Miniasm	Newbler
contigs ( $\geq 1000$ pb)	–	237	34	910
N50	–	57840	112086	487000
Largest contig	–	658902	340549	244182
Execution time	–	112m58s	3.6sec	8m20s
Total length	–	10509831	2712071	5203730
misassembled contigs	–	8	4	2
misassembled contigs length	–	963953	488237	71592
Génome fraction	–	96.5	71.63	98.9

Metrics	SGA	Smartdenovo	Abruijn	Ra
contigs ( $\geq 1000$ pb)	–	26	26	2
N50	–	164126	130182	2510403
Largest contig	–	363376	345439	2510403
Execution time	–	37s	1248s	47s
Total length	–	3341599	3072900	3383054
misassembled contigs	–	5	2	2
misassembled contigs length	–	946100	275790	3383054
Génome fraction	–	89.02	84.69	86.75

L'absence de résultats pour les assembleurs Celera et SGA se traduit par la création d'un fichier de sortie vide.

#### Assembleurs SRL

Metrics	DBG2OLC		Spades		Cerulean
	run 1	run 2	run 1	run 2	run 1
contigs ( $\geq 1000$ pb)	1	48	3	44	8
N50	3636390	201371	3601686	191200	1474160
Largest contig	3636390	323681	3601686	377379	1845480
Execution time	38s	1m12s	7m10s	20m12s	12min
Total length	3636390	3631852	3607931	3554907	10700143
misassembled contigs	1	4	1	8	7
misassembled contigs length	3636390	454208	3601686	1143113	9680456
Génome fraction	99	99	99.99	98.4	99.77

Run1 = Minion+contigs+illumina

Run2 = contigs+illumina



#### 4.4 Test 2 : Acinetobacter sp, ADP1, run6 ; Minion 20x

Jeu de données :

- reads Oxford NanoPort (MinIon), corrigés avec lordec : 10116 reads
- reads Illumina (MiSeq) : 211219 reads de taille 150 pb à 3 kbp
- contigs générés par Minia à partir des reads Illumina (233 contigs)

##### Assembleurs LRO

Metrics	Celera	Falcon	Miniasm	Newbler
contigs ( $\geq 1000$ pb)	–	29	1	2056
N50	–	3651707	3594439	2321
Largest contig	–	356596	3594439	325922
Execution time	–	138m3s	13s	27m1s
Total length	–	3651707	3594439	7636549
misassembled contigs	–	5	1	1
misassembled contigs length	–	1172354	3594439	125508
Génome fraction	–	90.6	91.08	98.9

Metrics	SGA	Smartdenovo	Abruijn	Ra
contigs ( $\geq 1000$ pb)	–	1	1	4
N50	–	3622997	3601882	1215050
Largest contig	–	3622997	3601882	1717290
Execution time	–	–	1329s	4m29s
Total length	–	3622997	3601882	3648098
misassembled contigs	–	1	1	3
misassembled contigs length	–	3622997	3601882	3292433
Génome fraction	–	97.22	100	88.45

L'absence de résultats pour les assembleurs Celera et SGA se traduit par la création d'un fichier de sortie vide.

##### Assembleurs SRL

Metrics	DBG2OLC		Spades		Cerulean
	run 1	run 2	run 1	run 2	run1
contigs ( $\geq 1000$ pb)	3	48	3	44	7
N50	2743656	201371	3602046	191200	881591
Largest contig	323681	323681	3602046	377379	2302560
Execution time	1m27s	1m12s	11m25s	20m12s	33min
Total length	3341152	3631852	3608291	3554907	7041048
misassembled contigs	2	4	1	8	7
misassembled contigs length	3235582	454208	3602046	1143113	7041048
Génome fraction	88.35	99	99.99	98.4	99.7

Run1 = Minion+contigs+illumina

Run2 = contigs+illumina

#### 4.5 Test 3 : Escherichia coli k-12, reads Pacbio 10x (P4-C2)

Jeu de données :

- reads Pacbio corrigés avec lordec (couverture 10x) : 36355 reads
- reads Illumina (MiSeq) : 16759877 reads de taille 150 pb à 3 kbp
- contigs générés par sparse assembler à partir des reads Illumina (1876792 contigs)

##### Assembleurs LRO

Metrics	Celera	Falcon	Miniasm	Newbler
contigs ( $\geq 1000$ pb)	332	–	161	7537
N50	47318	–	19379	1999
Largest contig	157193	–	54170	31521
Execution time	101m6s	–	15s	21m42s
Total length	5097702	–	1921945	18789586
misassembled contigs	22	–	16	299
misassembled contigs length	161184	–	369367	606999
Génome fraction	97.7	–	55.7	98.69

Metrics	SGA	Smartdenovo	Abriijn	Ra
contigs ( $\geq 1000$ pb)	27519	93	–	152
N50	89891795	33149	–	59639
Largest contig	20571	95016	–	265054
Execution time	45m12s	18s	–	3m11s
Total length	91585601	2787937	–	5638785
misassembled contigs	1709	13	–	173
misassembled contigs length	7978716	583840	–	4118425
Génome fraction	99.9	53.04	–	87.13

L'absence de résultats pour les assembleurs Falcon et Abriijn se traduit par la création d'un fichier de sortie vide.

##### Assembleurs SLR

Metrics	DBG2OLC		Spades		Cerulean
	run 1	run 2	run 1	run 2	run1
contigs ( $\geq 1000$ pb)	167	502	33	90	25
N50	24576	729	423200	126572	284387
Largest contig	66617	4730	584914	221710	855359
Execution time	153m16s	66m15s	1206m40s	2287m47s	182m9s
Total length	3285170	3006518	4752529	4551234	5536433
misassembled contigs	2	1	13	11	8
misassembled contigs length	65776	599	2581028	971587	2447404
Génome fraction	72.2	61	99.1	97.9	99.388

Run1 = Pacbio+contigs+illumina

Run2 = contigs+illumina

#### 4.6 Test 4 : Escherichia coli k-12, reads Pacbio 100x (P4-C2)

Jeu de données :

- reads Pacbio corrigés avec lordec (100x) : 91394 reads
- reads Illumina (MiSeq) : 16759877 reads de taille 150 pb à 3 kbp
- contigs générés par sparse assembler à partir des reads Illumina ( 1876792 contigs)

##### Assembleurs LRO

Metrics	Celera	Falcon	Miniasm	Newbler
contigs ( $\geq 1000$ pb)	317	9070	1	16421
N50	71566	15423	4685365	1999
Largest contig	281759	69760	4685365	35033
Execution time	830m	129 min	2m41s	125m3s
Total length	6968300	111813089	4685365	36154946
misassembled contigs	93	3063	1	974
misassembled contigs length	714575	50547586	4685365	1922459
Génome fraction	99.9	100	96.98	99.6

Metrics	SGA	Smartdenovo	Abrijn	Ra
contigs ( $\geq 1000$ pb)	80568	12	1	332
N50	7366	535879	4642185	48453
Largest contig	29821	1303952	4642185	8062545
Execution time	283m27s	798s	2756s	53m17s
Total length	458775476	4840284	4642185	8062545
misassembled contigs	12193	6	8	560
misassembled contigs length	76952755	3730610	4642185	6790624
Génome fraction	100	95.90	99.9	94.42

##### Assembleurs SRL

Metrics	DBG2OLC		Spades		Cerulean
	run 1	run 2	run 1	run 2	run1
contigs ( $\geq 1000$ pb)	361	502	27	90	17
N50	32616	729	488422	126572	613288
Largest contig	206766	4730	960524	221710	737401
Execution time	160min	66m15s	724m21s	2287m47s	234m47s
Total length	7520958	3006518	4941241	4551234	5278430
misassembled contigs	43	1	9	11	6
misassembled contigs length	1817923	599	3223591	971587	2029652
Génome fraction	90.8	61	99.55	97.9	99.38

Run1 = Pacbio+contigs+illumina

Run2 = contigs+illumina

#### 4.7 Test 5 : Escherichia coli k-12, reads Pacbio 10x (P6-C4)

Jeu de données :

- reads Pacbio corrigés avec lordec (couverture 10x) : 8746 reads
- reads Illumina (MiSeq) : 16759877 reads de taille 150 pb à 3 kbp
- contigs générés par sparse assembler à partir des reads Illumina (1876792 contigs)

##### Assembleurs LRO

Metrics	Celera	Falcon	Miniasm	Newbler
contigs ( $\geq 1000$ pb)	60	–	26	2136
N50	167523	–	350193	5198
Largest contig	357039	–	703052	79937
Execution time	181m47s	–	10s	10m
Total length	4838203	–	4585882	8129512
misassembled contigs	6	–	7	47
misassembled contigs length	277088	–	2172415	107564
Génome fraction	97.3	–	95.93	98.97

Metrics	SGA	Smartdenovo	Abruijn	Ra
contigs ( $\geq 1000$ pb)	178	9	10	12
N50	13410	1207236	857514	583331
Largest contig	28016	1490628	1717212	295905
Execution time	82m	83s	609s	3m17s
Total length	1281475	4650012	4576044	4893126
misassembled contigs	6	7	9	30
misassembled contigs length	66873	3343687	2394849	4245053
Génome fraction	16.04	94.86	97.2	94.39

L'absence de résultats pour l'assembleur Falcon se traduit par la création d'un fichier de sortie vide.

##### Assembleurs SRL

Metrics	DBG2OLC		Spades		Cerulean
	run 1	run 2	run 1	run 2	run1
contigs ( $\geq 1000$ pb)	68	502	27	90	27
N50	64971	729	472057	126572	400533
Largest contig	208657	4730	708505	221710	610814
Execution time	152m55s	66m15s	875m23s	2287m47s	185m24s
Total length	3618860	3006518	5171060	4551234	4774395
misassembled contigs	9	1	13	11	9
misassembled contigs length	775274	599	3628070	971587	2463561
Génome fraction	75	61	99.4	97.9	99.28

Run1 = Pacbio+contigs+illumina

Run2 = contigs+illumina

#### 4.8 Test 6 : Escherichia coli k-12, reads Pacbio 100x (P6-C4)

Jeu de données :

- reads Pacbio corrigés avec lordec (100x) : 87497 reads
- reads Illumina (MiSeq) : 16759877 reads de taille 150 pb à 3 kbp
- contigs générés par sparse assembler à partir des reads Illumina ( 1876792 contigs)

##### Assembleurs LRO

Metrics	Celera	Falcon	Miniasm	Newbler
contigs ( $\geq 1000$ pb)	451	33	2	27179
N50	36715	291737	4680508	1999
Largest contig	370387	938257	4680508	15870
Execution time	9051m49s	53m26s	332 sec	7h6m
Total length	10746953	4845711	4734166	57510328
misassembled contigs	79	12	2	616
misassembled contigs length	1239281	3259563	4734166	1213563
Génome fraction	99.55	94.5	96.53	99.77

Metrics	SGA	Smartdenovo	Abriijn	Ra
contigs ( $\geq 1000$ pb)	1410	12	1	120
N50	3352	535879	4642563	130683
Largest contig	18357	1303952	4642563	415320
Execution time	799m	1846s	7049s	2h45m4s
Total length	4425758	4840284	4642563	7907056
misassembled contigs	1	6	8	135
misassembled contigs length	9397	3730610	4642563	5345118
Génome fraction	49.6	95.90	99.9	92.49

##### Assembleurs SLR

Metrics	DBG2OLC		Spades		Cerulean
	run 1	run 2	run 1	run 2	run1
contigs ( $\geq 1000$ pb)	134	502	23	90	72
N50	81570	729	449002	126572	121474
Largest contig	386914	4730	707712	221710	287981
Execution time	163m30s	66m15s	830m18s	2287m47s	265m26s
Total length	6439845	3006518	4948122	4551234	4772924
misassembled contigs	15	1	12	11	68
misassembled contigs length	1457662	599	3337622	971587	4467714
Génome fraction	91.57	61	99.56	97.9	98.9

Run1 = Pacbio+contigs+illumina

Run2 = contigs+illumina

## 4.9 Test 7 : Escherichia coli k-12, reads Minion 20x

Jeu de données :

- reads Minion corrigés avec lordec (10x) : 22270 reads
- reads Illumina (MiSeq) : 16759877 reads de taille 150 pb à 3 kbp
- contigs générés par sparse assembler à partir des reads Illumina ( 1876792 contigs)

### Assembleurs LRO

Metrics	Celera	Falcon	Miniasm	Newbler
contigs ( $\geq 1000$ pb)	189	–	1	3769
N50	87431	–	4665895	3191
Largest contig	313881	–	4665895	33550
Execution time	223m50s	–	21.6sec	15m40s
Total length	5758951	–	4665895	7798978
misassembled contigs	5	–	1	11
misassembled contigs length	352118	–	4665895	31040
Génome fraction	99.2	–	88.85	98.5

Metrics	SGA	Smartdenovo	Abruijn	Ra
contigs ( $\geq 1000$ pb)	17680	2	5	7
N50	9481	4650531	4618085	994753
Largest contig	47228	4650531	2055696	1509502
Execution time	136m37s	205s	28554s	4m23s
Total length	134072452	4707245	4618085	4737110
misassembled contigs	178	1	9	9
misassembled contigs length	1950884	4650531	3673671	3400471
Génome fraction	99.99	91.15	99.29	86.40

L'absence de résultats pour l'assembleur Falcon se traduit par la création d'un fichier de sortie vide.

### Assembleurs SLR

Metrics	DBG2OLC		Spades		Cerulean
	run 1	run 2	run 1	run 2	run 1
contigs ( $\geq 1000$ pb)	129	502	26	90	13
N50	50709	729	472057	126572	2891290
Largest contig	135085	4730	1030461	221710	2891290
Execution time	155min	66m15s	1235min	2287m47s	306m39s
Total length	4107939	3006518	5264214	4551234	5011011
misassembled contigs	5	1	14	11	4
misassembled contigs length	216246	599	3876758	971587	3727198
Génome fraction	73	61	99.3	97.9	99.18

Run1 = Pacbio+contigs+illumina

Run2 = contigs+illumina

#### 4.10 Test 8 : *Saccharomyces cerevisiae* W303, reads Pacbio 10x (P4-C2)

Jeu de données :

- reads Pacbio corrigés avec Lordec (10x) : 26196 reads
- reads Illumina (MiSeq) : 3815678 reads de taille 150 pb à 3 kbp
- contigs générés par sparse assembler à partir des reads Illumina (10055 contigs)

##### Assembleurs LRO

Metrics	Celera	Falcon	Miniasm	Newbler
contigs ( $\geq 1000$ pb)	576	1157	158	11385
N50	50224	16175	34485	1999
Largest contig	245675	68708	86751	52210
Execution time	203min42sec	134s	16sec	102m36s
Total length	14792911	11458559	4996345	31879905
misassembled contigs	225	111	72	410
misassembled contigs length	9484293	1048482	2560401	2517006
Génome fraction	95.3	97.24	40.024	97.4

Metrics	SGA	Smartdenovo	Abriijn	Ra
contigs ( $\geq 1000$ pb)	810	174	44	121
N50	8396	60876	60553	159052
Largest contig	21663	215286	118337	427616
Execution time	177m3s	91s	1285s	17m36s
Total length	5073045	8456624	2705209	11991617
misassembled contigs	75	106	88	351
misassembled contigs length	658950	5884917	1716698	10688334
Génome fraction	20.7	66.95	22.56	85.61

##### Assembleurs SLR

Metrics	DBG2OLC		Spades		Cerulean
	run 1	run 2	run 1	run 2	run1
contigs ( $\geq 1000$ pb)	58	1157	256	564	259
N50	427868	16175	83770	37681	153850
Largest contig	989517	68633	369700	145780	444461
Execution time	1min32s	2min22s	64min	24min30s	183m16s
Total length	11952152	11458559	11760318	11518863	12849306
misassembled contigs	47	111	161	130	99
misassembled contigs length	11715802	1048482	9252972	2880183	10130544
Génome fraction	92.398	97.24	98.44	98.09	94.8

Run1 = Pacbio+contigs+illumina

Run2 = contigs+illumina

#### 4.11 Test 9 : *Saccharomyces cerevisiae* W303, reads Pacbio 100x (P4-C2)

Jeu de données :

- reads pacbio corrigés avec lordec (100x) : 261964 reads
- reads Illumina (MiSeq) : 3815678 reads de taille 150 pb à 3 kbp
- contigs générés par sparse assembler à partir des reads Illumina (10055 contigs)

##### Assembleurs LRO

Metrics	Celera	Falcon	Miniasm	Newbler
contigs ( $\geq 1000$ pb)	5599	17857	25	64900
N50	16276	18670	755806	1999
Largest contig	89231	78140	1271021	42447
Execution time	1921m12s	936h	444s	9969m
Total length	52637674	267821084	12022519	140588558
misassembled contigs	1114	6585	24	1716
misassembled contigs length	16653903	122742110	11952416	3627190
Génome fraction	88.1	99.28	94.26	98.4

Metrics	SGA	Smartdenovo	Abriijn	Ra
contigs ( $\geq 1000$ pb)	251627	20	26	–
N50	8721	820758	750435	–
Largest contig	32571	1543855	1531123	–
Execution time	1408m23s	3081s	36267	–
Total length	1492957147	12209277	12207471	–
misassembled contigs	30597	19	557	–
misassembled contigs length	266835653	12187629	11838914	–
Génome fraction	99.52	95.26	97.80	–

L'absence de résultats pour l'assembleur Ra se traduit par la création d'un fichier de sortie vide.

##### Assembleurs SLR

Metrics	DBG2OLC		Spades		Cerulean
	run 1	run 2	run 1	run 2	run1
contigs ( $\geq 1000$ pb)	103	1157	188	564	240
N50	165784	16175	129444	37681	190934
Largest contig	423865	68633	372443	145780	638854
Execution time	46m35s	2min22s	369m44s	24min30s	214min24s
Total length	9393403	11458559	11944790	11518863	12497420
misassembled contigs	69	111	138	130	89
misassembled contigs length	8584214	1048482	10489544	2880183	10040521
Génome fraction	68.8	97.24	98.5	98.09	94.7

Run1 = Pacbio+contigs+illumina

Run2 = contigs+illumina



## 4.12 Test 10 : Saccharomyces cerevisiae W303, reads Minion 20x

Jeu de données :

- reads Minion corrigés avec lordec (20x) : 47027 reads
- reads Illumina (MiSeq) : 3815678 reads de taille 150 pb à 3 kbp
- contigs générés par sparse assembler à partir des reads Illumina (10055 contigs)

### Assembleurs LRO

Metrics	Celera	Falcon	Miniasm	Newbler
contigs ( $\geq 1000$ pb)	894	–	169	35687
N50	20733	–	76389	1999
Largest contig	89592	–	291498	34166
Execution time	174m32s	–	17s	125m46s
Total length	13766378	–	10152501	87137754
misassembled contigs	243	–	109	426
misassembled contigs length	5059927	–	7547458	1421080
Génome fraction	87.9	–	68.99	96.8

Metrics	SGA	Smartdenovo	Abruijn	Ra
contigs ( $\geq 1000$ pb)	37932	156	–	155
N50	7991	84777	–	127765
Largest contig	42223	222924	–	349350
Execution time	572m51s	138s	–	13m21s
Total length	247200021	10456229	–	12573431
misassembled contigs	2209	109	–	283
misassembled contigs length	21144966	8223992	–	9670996
Génome fraction	98.8	77.93	–	69.06

L'absence de résultats pour les assembleurs Falcon et Abruijn se traduit par la création d'un fichier de sortie vide.

### Assembleurs SRL

Metrics	DBG2OLC		Spades		Cerulean
	run 1	run 2	run 1	run 2	run1
contigs ( $\geq 1000$ pb)	–	1157	268	564	327
N50	–	16175	79613	37681	94617
Largest contig	–	68633	370360	145780	361438
Execution time	–	2min22s	90m31s	24min30s	186m41s
Total length	–	11458559	11917307	11518863	15628680
misassembled contigs	–	111	172	130	154
misassembled contigs length	–	1048482	9543654	2880183	11814671
Génome fraction	–	97.24	98.293	98.09	93.78

L'absence de résultats pour l'assembleur DBG2OLC se traduit par la création d'un fichier de sortie vide.

Run1 = Pacbio+contigs+illumina

Run2 = contigs+illumina

### 4.13 Test 11 : *Caenorhabditis elegans*, reads Pacbio 10x (P6-C4)

Jeu de données :

- reads Pacbio corrigés avec lordec (10x) : 92597 reads
- reads Illumina (MiSeq) : 55070232 reads de taille 150 pb à 3 kbp
- contigs générés par sparse assembler à partir des reads Illumina (1022387 contigs)

#### Assembleurs LRO

Metrics	Celera	Falcon	Miniasm	Newbler
contigs ( $\geq 1000$ pb)	45	–	1056	104079
N50	1166	–	77770	1999
Largest contig	2181	–	499152	5415
Execution time	52min45sec	–	127sec	1122m21s
Total length	56811	–	71910795	208178837
misassembled contigs	3	–	45	180
misassembled contigs length	4229	–	3070192	359444
Génome fraction	0.048	–	12.3	45.5

Metrics	SGA	Smartdenovo	Abruijn	Ra
contigs ( $\geq 1000$ pb)	–	784	39	–
N50	–	151698	58948	–
Largest contig	–	546464	137901	–
Execution time	–	682s	1043s	–
Total length	–	91256859	2391267	–
misassembled contigs	–	36	0	–
misassembled contigs length	–	3859867	0	–
Génome fraction	–	13.16	0	–

L'absence de résultats pour les assembleurs Falcon et Ra, et SGA se traduit respectivement par la création d'un fichier de sortie vide et la création d'un fichier contenant l'ensemble des longues lectures non assemblées.

#### Assembleurs SRL

Metrics	DBG2OLC		Spades		Cerulean
	run 1	run 2	run 1	run 2	run1
contigs ( $\geq 1000$ pb)	490	17569	–	10330	–
N50	428709	8680	–	18309	–
Largest contig	1392481	115038	–	142723	–
Execution time	297m	415m42s	–	15min34s	–
Total length	103432617	95943011	–	96585462	–
misassembled contigs	18	2410	–	688	–
misassembled contigs length	2519846	10395574	–	9733061	–
Génome fraction	17.94	88.1	–	93.06	–

L'absence de résultats pour les assembleurs Spades et Cerulean se traduit respectivement par une insuffisance de mémoire RAM disponible et un temps d'exécution trop important.

Run1 = Pacbio+contigs+illumina

Run2 = contigs+illumina

#### 4.14 Test 12 : *Caenorhabditis elegans*, reads Pacbio 100x (P6-C4)

Jeu de données :

- reads Pacbio corrigés avec lordec (100x) : 740776 reads
- reads Illumina (MiSeq) : 55070232 reads de taille 150 pb à 3 kbp
- contigs générés par sparse assembler à partir des reads Illumina (1022387 contigs)

##### Assembleurs LRO

Metrics	Celera	Falcon	Miniasm	Newbler
contigs ( $\geq 1000$ pb)	6895	18031	140	410532
N50	58720	18602	1921931	1999
Largest contig	726905	82715	5921636	297726
Execution time	477 heures	908h	2h10min	528h20m
Total length	135257379	268424680	107385322	728306963
misassembled contigs	1115	0	110	5169
misassembled contigs length	34916259	0	101221508	10361505
Génome fraction	86.8	0	89.33	98.7

Metrics	SGA	Smartdenovo	Abriijn	Ra
contigs ( $\geq 1000$ pb)	–	90	–	–
N50	–	2249996	–	–
Largest contig	–	4716467	–	–
Execution time	–	30763s	–	–
Total length	–	107109382	–	–
misassembled contigs	–	74	–	–
misassembled contigs length	–	101746008	–	–
Génome fraction	–	91.17	–	–

L'absence de résultats pour l'assembleur SGA se traduit par la création d'un fichier contenant l'ensemble des longues lectures non assemblées.

L'absence de résultats issus de Abriijn et Ra se traduit par une erreur lors de l'assemblage.

##### Assembleurs SRL

Metrics	DBG2OLC		Spades		Cerulean
	run1	run2	run1	run2	run1
contigs ( $\geq 1000$ pb)	435	17569	–	10330	–
N50	644079	8680	–	18309	–
Largest contig	2338886	115038	–	142723	–
Execution time	662m52s	415m42s	–	15min34s	–
Total length	115756904	95943011	–	96585462	–
misassembled contigs	313	2410	–	688	–
misassembled contigs length	104520876	10395574	–	9733061	–
Génome fraction	90.96	88.1	–	93.06	–

L'absence de résultats pour les assembleurs Spades et Cerulean se traduit respectivement par une insuffisance de mémoire RAM disponible et un temps d'exécution trop important.

Run1 = Pacbio+contigs+illumina - Run2 = contigs+illumina

## Références

- [al99] Delcher AL et AL. “Alignment of whole genomes”. In : *Nucleic Acids Res.* 1999 Jun 1;27(11) :2369-76. PMID :10325427; PMCID :PMC148804 (1999).
- [al05] Margulies M et AL. “Genome sequencing in microfabricated high-density picolitre reactors.” In : *Nature* 2005, 437 : 376-380. (2005).
- [Sim11] Durbin R SIMPSON JT. “Efficient de novo assembly of large genomes using compressed data structures.” In : doi : 10.1101/gr.126953.111 *Genome Res.* (2011).
- [al12a] Anton Bankevich et AL. “SPAdes : A New Genome Assembly Algorithm and Its Applications to Single-Cell Sequencing”. In : *Journal of Computational Biology.* 19(5). (2012).
- [al12b] Sergey Koren et AL. “Hybrid error correction and de novo assembly of single-molecule sequencing reads”. In : *Nature Biotechnology* 30(7) :693-700 (2012).
- [al13a] Alexey Gurevich et AL. “QUAST : quality assessment tool for genome assemblies”. In : *Bioinformatics* 29(8), 1072-1075. (2013).
- [al13b] Viraj Deshpande et AL. “Cerulean : A hybrid assembly using high throughput short and long reads”. In : *arXiv :1307.7933.* (2013).
- [Riv14] Salmela L et RIVALS E. “LoRDEC : accurate and efficient long read error correction”. In : *Bioinformatics.* 2014;30(24) :3506-14. doi : 10.1093/bioinformatics/btu538 pmid :25165095; PubMed Central PMCID : PMC4253826. (2014).
- [al15] Chengxi Ye et AL. “DBG2OLC : Efficient Assembly of Large Genomes Using the Compressed Overlap Graph”. In : *arXiv :1410.2801.* (2015).
- [Li15] Heng LI. “Minimap and miniasm : fast mapping and de novo assembly for noisy long sequences”. In : *arXiv :1512.01801.* (2015).
- [al16a] Ivan Sović et AL. “Fast and sensitive mapping of nanopore sequencing reads with GraphMap”. In : *Nature Communications* 7, Article number :11307;doi :10.1038/ncomms11307 (2016).
- [al16b] Yu Lin et AL. “Assembly of Long Error-Prone Reads Using de Bruijn Graphs”. In : <http://dx.doi.org/10.1101/048413> (2016).
- [Chi16] Jason CHIN. *Falcon Genome Assembly Tool Kit Manual*. <https://github.com/PacificBiosciences/FALCON/wiki/Manual>. Jan. 2016.



**RESEARCH CENTRE  
RENNES – BRETAGNE ATLANTIQUE**

Campus universitaire de Beaulieu  
35042 Rennes Cedex

Publisher  
Inria  
Domaine de Voluceau - Rocquencourt  
BP 105 - 78153 Le Chesnay Cedex  
[inria.fr](http://inria.fr)

ISSN 0249-0803