NAME

SRAssembler – Selective and Recursive local Assembler

SYNOPSIS

SRAssembler [**-q** *query_file*] [**-s** *species*] [**-P** *parameter_file*] [**-f** *library_file*] [**-1** *left_end_file*] [**-2** *right_end_file*] [**-t** *query_type*] [**-o** *output_directory*] [**-R** *preprocessed_directory*] [**-m** *min_contig_length*] [**-M** *max_contig_length*] [**-e** *min_score*] [**-c** *min_coverage*] [**-k** *kmer*] [**-z** *insert_size*] [**-n** *num_rounds*] [**-x** *num_reads_per_file*] [**-A** *assembler_program*] [**-S** *spliced_alignment_program*] [**-G** *gene_finding_program*] [**-p**] [**-i**] [**-l**] [**-a** *assemble_round*] [**-r** *clean_round*] [**-v**] [**-h**]

DESCRIPTION

SRAssembler is a pipeline program that can assemble genomic DNA reads using homologous sequences as input. **SRAssembler** first aligns the reads that can locally be mapped onto those query sequences. These mapped reads are then assembled as contigs, which are used to find other reads partially mapped to them. This *in-silico* chromosome walking strategy can recursively gather reads that are associated with regions of interest. The gene structure of the final contigs is predicted by spliced alignment and *ab initio* gene finding programs.

The **SRAssembler** program was developed in the group of Prof. Volker Brendel. Information on program availability may be obtained at http://brendelgroup.org/bioinformatics2go/bioinformatics2go.php.

Correspondence relating to **SRAssembler** should be addressed to

Volker Brendel

School of Informatics and Computing

Indiana University, Bloomington, Indiana 47405

Tel: (515) 294-9884, Fax: (515) 294-6755

Email: vbrendel@indiana.edu

OPTIONS

-q *query_file*

Required. The FASTA format query file.

-s *species*

Required. Set species to select the most appropriate splice site models. Options: “human”, “mouse”, “rat”, “chicken”, “drosophila”, “nematode”, “fission_yeast”, “aspergillus”, “arabidopsis”, “maize”, “rice”, “medicago”.

-P *parameter_file*

Required. Parameter configuration file. **SRAssembler** allows users to specify the parameters of the programs used in **SRAssembler** such as **Vmatch**, **GenomeThreader**, **GeneSeqer**, **Exonerate** and **Snap**. The parameters for each program are grouped by **[program_name]**. For example:

```
[Vmatch_init]
e=1
l=11

[Vmatch]
e=0
l=30

[GenomeThreader]
gcmincoverage=10
prminmatchlen=15

[GeneSeqer]
x=14
y=14
z=25

[Exonerate]
percent=20

[Snap]
snaphmm=A.thaliana.hmm
```

For **Vmatch**, the **[Vmatch_init]** is the parameter settings for the initial round.**-l** option in **[Vmatch_init]** is the match length in the first round. Since the first round is the alignment of homologous genes, if you set it too high, you may get very few hits. The default value for protein query sequences is 10; 30 for cDNA query sequences. If your assembly has very poor spliced alignment results, you can decrease this value to gather more reads to improve your assembly results. **-e** option in **[Vmatch_init]** is the number of mismatches allowed in the first round. The default value is 1. If this value is too low, you may get very few matched reads if your query sequences are not very well conserved. On the other hand, when you set this value too high, some false positive reads may be fetched. **-l** option in **[Vmatch]** is the match length in recursive round. The default value is 30. This value controls the speed of chromosome walking. If your dataset is very deep, you can specify higher value and save the running time. **-e** option **in [Vmatch]** is the mismatches allowed in recursive round. The default value is 0.

For snap, please set the environment variable **ZOE** to the path of HMM files.

-f *library_file*

Library definition file. Each library is specified by **[LIBRARY]** section. Each section includes the following items:

insert_size: the insert size of the library. This value is used in paired-end reads. Default: 300.

direction: the sequencing direction of paired-end reads. 0 : forward-reverse; 1: reverse-forward. Default: 0.

r1: left-end reads file or single-end reads file.

r2: right-end reads file. Do not specify if your library is single-end.

format : “fastq” or “fasta”. Default: “fastq”.

Note that if your library contains both paired-end and single-end reads, please treat them as two libraries.

Here is an example of two libraries:

```
[LIBRARY]
insert_size=200
direction=0
r1=reads1_200.fq
r2=reads2_200.fq
format=fastq

[LIBRARY]
insert_size=1000
direction=0
r1=reads1_1000.fq
r2=reads2_1000.fq
format=fastq
```

-1 *left_end_file*

Required if you do not specify library file. The option can be used to specify the single-end file name or the left end file name for paired-end reads.

-2 *right_end_file*

Right end file name for paired-end reads.

-z *insert_size*

The insert size of the paired-end reads [default: 300]

-t *query_type*

Query file type: Options: “protein”, “cdna” [default: protein].

-o *output_directory*

Output directory [default: current directory]

-R *preprocessed_directory*

Preprocessing directory [default: output directory/reads_data]. For each **SRAssembler** run, reads data will be split into several parts. This preprocessing step is executed first time only. When you adjust parameters and rerun **SRAssembler**, the preprocessed reads will be reused. If you have several runs which share the same dataset, you can use this option to specify the directory of preprocessed reads.

-m *min_contig_length*

Minimum contig length to be reported [default: 200]

-M *max_contig_length*

Maximum contig length to be reported [default: 10000]. If the contig size is larger than this value, **SRAssembler** will stop assembling such contigs. Because they are long enough, we do not want to waste our time to keep assembling them again. We also remove the reads associated with these contigs, therefore improving the running time. This is done by the following steps:

1. In each round, we test if the contig length is larger than the maximum contig size. We trim the head and tail of the contigs and make their size be equal to the maximum contig size, and then copy these contigs to the candidate long contig file. Note that we do not remove them immediately, because we want to do the double check if these long contigs are correctly assembled. If such contigs are assembled again, we can confirm they are our final contigs.

2. In the next round, we align the candidate long contigs to the current assembled contig file (done by **Vmatch**). If matched, we move the contigs to the permanent long contig file.

3. We align current matched reads to the long contigs (done by **Bowtie**). If matched, those reads are removed from the reads pool.

4. Long contigs are removed from the query file of the next round.

-e *min_score*

Minimum score to determine the query genes have been assembled. [default: 0.5]. In each round, **SRAssembler** will align the query sequences to the assembled contigs. If the alignment score and coverage (see **-c** option) are above the specified threshold, the assembling of these genes is considered finished. These query sequences will be removed and **SRAssembler** will not assemble them anymore.

-c *min_coverage*

Minimum coverage to determine the query genes are assembled. [default: 0.5]. See the explanation of option **-e**.

-k *kmer*

The k-mers of assembler. **SRAssembler** can test multiple k-mers at the same time. The format is: **start_k:interval:end_k**. The start_k and end_k must be an odd value, and the interval must be an even value. For example, **15:10:45** means k-mer value 15, 25, 35, 45 will be tested. [default: 15:10:45]

-n *num_rounds*

Number of maximum rounds [default: 10]

-x *num_reads_per_file*

Number of reads in the split files [default: 500000]

-A *assembler_program*

The internal assembler program used in **SRAssembler**. Options: **0**=>**SOAPdenovo 1**=>**ABySS** [default: 0]

-S *spliced_alignment_program*

The spliced alignment program used in **SRAssembler**. Options: **0**=>**GenomeThreader**, **1**=>**GeneSeqer**, **2**=>**Exonerate** [default: 0]

-G *gene_finding_program*

The gene finding program used in **SRAssembler**. Options: **0**=>none, **1**=>Snap [default: 0]

-p Do the preprocessing only.

-i Do NOT start the assembling from the last checkpoint. By default, **SRAssembler** will continue the previous run if the **-n** option is larger than previous one. For example, if the previous run stops at round 6. You can continue previous run starting from 7 if **-n** is larger than 6. If you specify this **-i** option, **SRAssembler** will start the assembly from first round.

-l Do not check the genes are assembled. By default, **SRAssembler** will align the query sequences to the assembled contigs. If the alignment score (see **-e** option) and coverage (see **-c** option) are above the specified threshold, the assembling of these genes is considered finished. If you specify this **-l** option, **SRAssembler** will NOT check if genes are assembled. This is particular useful if you want to assemble longer contigs and do not want to stop the assembling when the coding

regions are covered. For example, if you want to assemble the UTR or promoter regions, you can turn this option on to get longer contigs.

-a *assemble_round*

The round number the assembling starts [default: 1].This option indicates which round **SRAssembler** starts to do the assembly. For the deeper dataset, we can start the assembly from round 1. But for some datasets with lower coverage, assembly in the early rounds may cause the wrong contigs, which will further affect your final results.

-r *clean_round*

The round number to remove the unrelated contigs and reads. For example, "3" means **SRAssembler** do the cleaning at round 3, 6, 9, 12... [default: 3]. **SRAssembler** may assemble many contigs which are unrelated to the query sequences. So in the cleaning rounds, **SRAssembler** align the query sequences to the contigs. If the contigs have 0 hits, they will be deleted. The reads which fail to map to the survival contigs will be removed as well.

-v Debug mode. The detailed information will be printed.

-h Show the usage of **SRAssembler**.

OUTPUT

- **output_dir/output/summary.html:** The summary html file.
- **output_dir/output/all_contigs.fasta:** All assembled contigs.
- **output_dir/output/hit_contigs.fasta:** The contigs identified by spliced alignment program.
- **output_dir/output/output.aln:** The spliced alignment report.
- **output_dir/output/output.ano:** The *ab initio* gene prediction report.
- **output_dir/output/msg.log:** The detailed log file.
- **output_dir/output/intermediates directory:** All intermediate contigs are saved in this directory. For example, the contigs assembled in round 3 is contigs-3.fasta.
- **output_dir/reads_data directory** : The original reads data files will be split into smaller parts in FASTA format. We save these preprocessed files in this directory. If you want to rerun this dataset, **SRAssembler** will read the preprocessed data directly. This directory can also be specified by **-R** option.
- **output_dir/tmp directory** : This directory is used to keep all the temporary files. They serve as the checkpoint files so that SRAssembler can continue previous assembling. If this directory is removed, SRAssembler must start from scratch in the next run.

TEST FILES

We used SAMTools' wgsim program to simulate 2 libraries from region 300,000 ~ 400,000 of chromosome 1 of Arabidopsis with insert size 200bp and 1kb. Each library has 50,000 paired reads. The base error rate is 0.002 and the fraction of indels is 0.001. In data directory, several files are provided:

- input/LOC_Os06g04560.pep: the protein sequence of rice gene Os06g04560.1. FASTA format.
- input/reads1_200.fq: the left-end of simulated reads with insert size 200bp. The length is 70bp, FASTQ format.
- input/reads2_200.fq: the right-end of simulated reads with insert size 200bp. The length is 70bp, FASTQ format.
- input/reads1_1000.fq: the left-end of simulated reads with insert size 1000bp. The length is 70bp, FASTQ format.
- input/reads2_1000.fq: the right-end of simulated reads with insert size 1000bp. The length is 70bp, FASTQ format.
- input/reads_2.fastq: the right-end of simulated reads. The length is 70bp, FASTQ format
- libraries_200bp.config: library definition file for 200bp library.
- libraries_200bp_1kb.config: library definition file for 200bp and 1kb libraries.

RUNNING TESTS

In "data" folder, a simple script is provided: xtest. This script will run 5 tests. Type:

```
cd data
./xtest-all
```

- **Example 1 : (use -1, -2 options to assign reads files)**
  In this test, we use one libraries with insert size 200bp using **-1** and **-2** options. This script executes the following command:

```
../bin/SRAssembler -q LOC_Os06g04560.pep -1 reads1_200.fq -2 reads2_200.fq -z 200 -s arabidopsis -t protein -x 15000 -n 13 -P Params.conf -G 1 -o testout1
```

- **Example 2 : (use library definition file)**
  In this test, we use one libraries with insert size 200bp using a library definition file. This script executes the following command:

```
../bin/SRAssembler -q input/LOC_Os06g04560.pep -f libraries_200bp.config -s arabidopsis -t protein -x 15000 -n 13 -P Params.conf -G 1 -o testout2
```

- **Example 3 : (use two libraries)**
  In this test, we use two libraries with insert size 200bp and 1kb. You will see the assembly is very efficient. We need only 2 rounds to assemble the target region compared to 6 rounds in the Example 2.

```
./bin/SRAssembler -q input/LOC_Os06g04560.pep -f libraries_200bp_1kb.config -s arabidopsis -t protein -x 15000 -n 3 -P Params.conf -G 1 -o testout3
```

- **Example 4 : (test 200bp library with MPI feature)**
  In this test, we use 4 cores to run the **SRAssembler** simultaneously. You can notice the improvement of running time. (You may get errors if you do not install MPI environment properly)

```
mpirun -n 4 ../bin/SRAssembler_MPI -q input/LOC_Os06g04560.pep -f
libraries_200bp.config -s arabidopsis -t protein -x 15000 -n 13 -P Params.conf -G 1 -o
testout4
```

- **Example 5 : (test the checkpoint feature)**
  In this test, we first test 3 rounds. Then the **-n** option is changed to 7. The **SRAssembler** will starts from 4th round. Finally, we use **-i** option to force SRAssembler to start the assembling from scratch:

```
../bin/SRAssembler -q input/LOC_Os06g04560.pep -f libraries_200bp.config
-R ./reads_data -s arabidopsis -t protein -x 15000 -n  3 -G 1 -P Params.conf -o testout5
../bin/SRAssembler -q input/LOC_Os06g04560.pep -f libraries_200bp.config
-R ./reads_data -s arabidopsis -t protein -x 15000 -n  7 -G 1 -P Params.conf -o testout5
../bin/SRAssembler -q input/LOC_Os06g04560.pep -f libraries_200bp.config
-R ./reads_data -s arabidopsis -t protein -x 15000 -n  7 -i -G 1 -P Params.conf -o
testout5
```

- **Example 6 : (use different score (-e) and coverage (-c))**
  In this test, we use higher score and coverage to check if the genes are assembled:

```
../bin/SRAssembler     -q input/LOC_Os06g04560.pep -f libraries_200bp.config
-R ./reads_data -s arabidopsis -t protein -x 15000 -n  10 -c 0.9 -e 0.6 -P Params.conf -o
testout6
```

- **Example 7 : (use GeneSeqer as spliced alignment program (-S 1))**
  In this test, we use GeneSeqer as the spliced alignment program. You might get errors if GeneSeqer is not properly installed:

```
../bin/SRAssembler     -q input/LOC_Os06g04560.pep -f libraries_200bp_1kb.config
-R ./reads_data -s arabidopsis -t protein -x 15000 -n  3 -S 1 -P Params.conf -o testout7
```

- **Example 8 : (use Exonerate as spliced alignment program (-S 2))**
  In this test, we use Exonerate as the spliced alignment program. You might get errors if Exonerate is not properly installed. Note the coverage here refers to the total nucleotides covered by query sequences:

```
../bin/SRAssembler     -q input/LOC_Os06g04560.pep -f libraries_200bp_1kb.config
-R ./reads_data -s arabidopsis -t protein -x 15000 -n  3 -S 2 -c 2000 -e 1000 -P
Params.conf -o testout8
```

- **Example 9 : (use Snap as *ab initio* gene prediction program (-G 1))**
  In this test, we use Snap as *ab initio* gene prediction program. You might get errors if Snap is not properly installed:

```
../bin/SRAssembler     -q input/LOC_Os06g04560.pep -f libraries_200bp_1kb.config
-R ./reads_data -s arabidopsis -t protein -x 15000 -n  3 -G 1 -P Params.conf -o testout9
```

- **Example 10 : (do preprocessing only)**

In this test, we do the preprocessing only. SRAssembler will stop right after the preprocessing is complete:

```
../bin/SRAssembler     -q input/LOC_Os06g04560.pep -f libraries_200bp_1kb.config
-s arabidopsis -t protein -x 15000 -n  3 -P Params.conf -o testout10 -p
```

- **Example 11 : (remove unrelated contigs and reads every 2 rounds (-r 2))**
  In this test, SRAssembler will clean unrelated reads and contigs every 2 rounds (round 2, 4, 6 …):

```
../bin/SRAssembler     -q input/LOC_Os06g04560.pep -f libraries_200bp.config
-R ./reads_data -s arabidopsis -t protein -x 15000 -n  3 -r 2 -P Params.conf -o testout11
```

- **Example 12 : (assemble contigs from 2nd round (-a 2))**
  In this test, SRAssembler will start assembling from 2nd round. SRAssembler will not assemble the first round and all the hit reads in the first round will be the "seed" reads and will be used to find adjacent reads associated with them:

```
../bin/SRAssembler     -q input/LOC_Os06g04560.pep -f libraries_200bp.config
-R ./reads_data -s arabidopsis -t protein -x 15000 -n  3 -a 2 -P Params.conf -o testout12
```

- **Example 13 : (do not check if the query genes are assembled (-l))**
  In this test, SRAssembler will not check if the query genes are assembled:

```
../bin/SRAssembler     -q input/LOC_Os06g04560.pep -f libraries_200bp.config
-R ./reads_data -s arabidopsis -t protein -x 15000 -n  3 -P Params.conf -l -o testout13
```

AUTHOR

Hsien-chao Chou <hsienchao.chou@gmail.com>