SL Finder Manual

Index

1) Introduction

SL Finder is a four-step pipeline of bioinformatic analyses meant to identify putative Spliced Leader (pSL) sequences from Transcriptome data assembled *de novo*, without making use of known SL sequences and with minimal manual curation by the user:

- SLFinder-step0: Filters a Trinity assemblies and retrieves the firsts and last bp of the longer isoform of each assembled gene. According to Trinity contig naming convention.
- SLFinder-step1: Creates and evaluates “Hook” sequences that more likely represent SLs and retrieves contig regions, either in the 3’ or 5’ region for further analyses.
- SLFinder-step2: Aligns the sequences identified for each hook and automatically trims them to generate the more likely pSL sequences.
- SLFinder-step3: Blast the pSL sequences against a reference transcriptome looking to verify their sequence and identify their coding location and copy number and

identify the transcripts with validated pSL variants. If provided, it uses an external reference to annotate the genomic region with each pSLs and discard those with matches.

2) Citation

Please cite…

3) Download and Installation

Currently the pipeline has been tested on a Linux environment. The scripts can be downloaded from https://github.com/LBC-Iriarte/SLFinderand require execution permissions to work properly. These can be granted with the following command:

```
chmod +x SLFinder-step*
```

Additionally, the following programs and/or packages installed and added to the PATH:

1) Ncbi-blast: https://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=Download
2) Cd-Hit: https://github.com/weizhongli/cdhit
3) Jellyfish: https://github.com/gmarcais/Jellyfish
4) MAFFT: https://mafft.cbrc.jp/alignment/software/source.html
5) Salmon: https://github.com/COMBINE-lab/salmon
6) Seqkit (v0.12.0 or higher): https://github.com/shenwei356/seqkit
7) Trinity (v2.10.0 or higher): https://github.com/trinityrnaseq/trinityrnaseq/wiki
8) Trimal (v1.4rev59 recomended): http://trimal.cgenomics.org/trimal
9) Weblogos: http://weblogo.berkeley.edu

Please follow each program installation instructions before running this pipeline. In addition, programs commonly included as part of the Linux installation are also used for file manipulation and some calculations like awk, cut, grep, sed, split and wc.

4) Verifying installation

First decompress the test data running

```
tar -xzvf test_dataset.tar.gz
```

And then:

```
cd test_dataset
../SLFinder-Step0 -a _SL_transcripts.fasta -f _filtered
../SLFinder-Step1 -a _SL_transcripts.fasta -f _filtered -o test_dataset_out -kc 0.001
../SLFinder-Step2 -o test_dataset_out (Or if you have installed trimal 1.2rev59):
../SLFinder-Step2 -ts 0.001 -o test_dataset_out
../SLFinder-Step3 -g celegans.PRJNA13758.WBPS11.genomic_c5.fa -br
annotation.celegans.uniprot.3A6239.fasta -o test_dataset_out
```

Note: Step3 will throw many warnings if you have a newer version of seqkit than v0.10.2. It doesn't affect results and will be fixed after release.

5) Running SL Finder - Quick version

SL Finder is designed to work with several assemblies of the same species. For this end each assembly should be named with a distinct ID followed by an extension that identifies it as an assembly (ej: [Sample]_assembly.fasta) and all assemblies must be in the working directory. It is important that the assembly to be conducted following a de novo approach with no read normalization. Using a reference for transcript assembly is likely to exclude the SL from the transcripts as the reads fail to properly map to them, while a normalization strategy is likely to reduce the number of reads with SLs sequences since they are expected to be overrepresented when compared with a random sequence. Since Trinity is not only a popular de novo assembled but the pipeline requires it to work, we will assume to assemble your transcriptomes. Use the following command for the assemblies:

```
Trinity --seqType fq --left reads_1.fq --right reads_2.fq --
no_normalize_reads
```

In addition: this pipeline requires the species genome (or as close related one as possible) and a reference for protein annotation like Swiss-Prot from Uniprot.

Once the assemblies and references are ready follow each step of the pipeline:

1) Filter assemblies – step0:

```
SLFinder-step0 -a "_assembly.fasta" -f "_filtered.fasta" -t n°
```

Here -a is the extension for the assemblies and -f the extension for the filtered files. Internally the pipeline runs "ls *_assembly.fasta" in the working directory to get the file name list. By default -a is "_Trinity.fasta" and -f "_Trinity.filtered". This script might take several minutes to a few hours, to improve running times assign several processors with "-t" parameter to run in parallel the programs that allow it.

Lastly, if you are using a different assembler than trinity or want to follow a different filtering strategy you can replace this step by simple introduced your custom filtered files to the next step.

2) Produce Hooks and retrieve matching sequences – Step1:

```
SLFinder-step1 -a "_assembly.fasta" -f "_filtered.fasta" -t n°
```

Details of how this script works are explained below, but to work the script needs again the original assemblies as well the filtered ones. The latter are used to generate the Hooks representing the pSL, then a BLAST search is carried out to identify matching regions in the original assembly. Results are filtered according to the consistence of their orientation in the contigs and relative frequency, then the sequence matches of the more promising Hooks is retrieved for further analyses.

This script will also generate an output folder to store the results of the entire pipeline (named by Default "SL-analysis"), with three folders inside: Results, Warnings and temp_files. Inside Results there will be three files with the hook information: "Step1-All_Hooks", "Step1-Best_Candidates" and "Best-Hook.fasta". As a safety measure, this script only runs if the output folder doesn't already exist but bear in mind that the following steps delete the results of their previous run stored in this folder.

3) Trimm Hook matches – Step2

```
SLFinder-step2 -t n°
```

This step only works with the files generated during Step1 and doesn't require additional inputs. However, this part of the pipeline is kept separated because several Trimal parameters might need fine-tuned to your specific dataset, and in some cases require manual curation. Basic information for the Trimal Fasta files is given in the file Step2_Trimal-Stats and unreliable alignments listed on "Step2_Manual-curation". To facilitate any manual curation, a sequence logos of each alignment is generated and stored on "Analysis directory"/Results/weblogos/aligments

Lastly, all Trimal filtered sequences are combined and clustered according sequence identity before been used in the next step.

4) Localize pSL sequences in a reference genome – Step3

```
SLFinder-step3 -g [Reference Genome] -br "external_reference" -t n°
```

The pSL sequences obtained so far are then located in a reference Genome by a BLAST search (with no gaps allowed and a 100% identity cutoff) to verify their sequence and identify potential donor sites (GT) at the 3' of the match or their reversed complement at the 5' (AC). Additionally, an external reference (provided by the command "-br") is used to annotate the loci encoding the pSL. Results are summarized in "Step3_Loci.tab" and the validated pSL sequences in "SL.fa".

6) The pipeline

Strategy, basic assumptions and problems to circumvent

While not requiring known SL sequences to work, this pipeline makes some basic assumptions about the SL mechanism:

1. The sequence is located in the 5' end of the transcript
2. It is present on the transcripts of many genes
3. It is not a palindrome who can be mapped to the same location in both possible orientations
4. There is at most one copy of it on each transcript
5. When mapped to the genome there is a canonical Splicing donor site at the 3'.

In addition, some features and limitations of the technology commonly used to assemble transcriptomes from Next Generation Sequencing data complicate using a simple pattern search to identify new SL sequences. For example, Poly-A capture used in many protocols to enrich RNA samples with eukaryotic mRNAs has the drawback of generating a lower read coverage toward the 5' region of the assembled transcript (where the SL is located). Another factor to consider is that several assemblers like Trinity can recover several isoforms for the same gene, something great when studying the biology of an organism but that for our purposes it's a source of noise if not properly filtered. And as an extra complication, unless a special protocol is used the strand of each assembled transcript is lost; meaning that we don't know at priori if we assembled the transcript or its reverse complement.

The first step is to filter the transcriptome by retrieving both terminal regions of one transcript per gene. To this end provide the script Step0 to do it based on Trinity's coting naming convention, but other strategies are viable (instructions below). Next, we generate "Hook" sequences by first retrieving Kmers (string of characters, bases in this case, of a given length) with more than a user given threshold counts (0.005% by default or) and assemble them with the first module of Trinity of Inchworm. The resulting hooks represent the more common sequences existing on the filtered assembly. Including, if present, the SL sequences of the transcriptome. By changing the kmer size it is possible to establish a tradeoff between the number of hooks and artifacts in the assembly, with the number of transcripts that provide information about the pSL sequence.

Next, matching transcripts on the unfiltered transcriptome for these hooks are identified by a BLAST search, and the hits orientation and relative abundance is used to filter likely false positives. Working with the raw sequences also helps to minimize errors resulting from assembly artifacts caused by using so limited information.

Two filters are currently implemented: Consistency Orientation Index (coi) and Observation Count Cutoff (occ).

- Consistency Orientation Index

Here is when the assumption of the SL not been palindromic comes into play. We assume that if a Hook truly is an SL, it will be oriented in a consistent way in one of these two configurations:

1) The assembled hook is the SL and is oriented from 5’ to 3’ if located at the beginning of the contig, or 3’ to 5’ if found at the end (Fig. 1), meaning that the reverse complement of the transcript was assembled but the Hook is in the correct orientation.

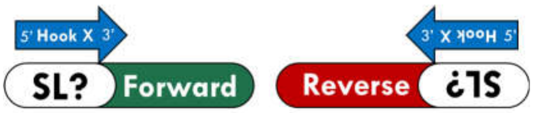


Fig 1: Expected disposition of a Hook for a true SL.

2) The assembled hook is the reverse complement of the SL and is oriented from 3’ to 5’ if located at the beginning of the contig, or 5’ to 3’ if found at the end (Fig. 2), meaning that the reverse complement of the transcript, and so is the Hook.

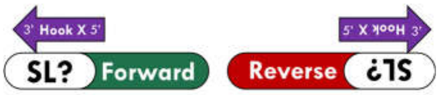


Fig 2: Expected disposition of a Hook for a true SL, but the produced hook is the reverse complement of the SL.

The coi consist in the following formula:

$$coi = \left| \left((Sf + Er) - (Sr + Ef) \right) \Big/ \left(Sf + Er + Sr + Ef \right) \right|$$

Here “Sf” is are the number of times the hook in question has a hit at the start of the coting in the same sense as it was assembled and “Er” is the how many times it was observed at the end in the reverse respective to it, “Sr” and “Ef” their counter parts: In reverse at the start and forward at the end. If the Hook is in fact an SL most if not all hits on the transcriptome are going to be in either one or the other configuration, so the result of the formula will be close to 1. Transcripts with multiple hits for the same Hook are not considered.

- Observation Count Cutoff

Testing indicates that, while effective, by its own the coi filter is not enough since Hook with few counts often have high coi values, regardless of their origin. For this reason, the pipeline by default sets an occ value (the median value of the hits found for all hooks).

Once filtered, the next step is to recover the correct sequence of the SLs. Its common that Hooks representing a true SL to include more bp on both 5’ and 3’ ends. This happens because of the limited information sequences used during their assembly or because several transcripts begin with the same sequence (i.e. a start codon ATG). Other anomalies include chimeric versions among different SLs, multiple hooks for the same SL or missing bases in the 3’ end (usually when there are two or more SL variants that are different on their 5’ but the 3’ is conserved). This is solved by retrieving the sequence of all the Best Hooks Matches, align them with MAFFT and trimming them automatically with Trimal (or manually if required). While not necessarily eliminating the problem (e.g. if 30 of the 50 transcripts with a true SL have a AT following, these bp are likely to remain in the sequence after the trimming) this process minimizes it.

After trimming, somewhere between dozens or several hundred of very similar sequences (referred as Hook variants) will be produced. They represent all distinct possible SLs sequences present on the transcriptome, unfortunately most of them are likely caused for sequencing errors. The next step is to use a reference genome to localize encoding loci via a BLAST search (no gaps, 100% identity and over a user defined threshold of the pSL coverage). Once the Loci are identified the script searched for potential donor sites (“GT” or “AC” depending on the strand) and the entire region annotated by another BLAST search against an appropriate external reference (e.g. Uniprot). The pSL Loci is considered valid if there is a potential donor sites and the BLAST search brings no results.

Pipeline implementation

Each step in the Pipeline is summarized in Fig 3. Now we will explain how each step is implemented on each step and the options available to change its behavior by the user.

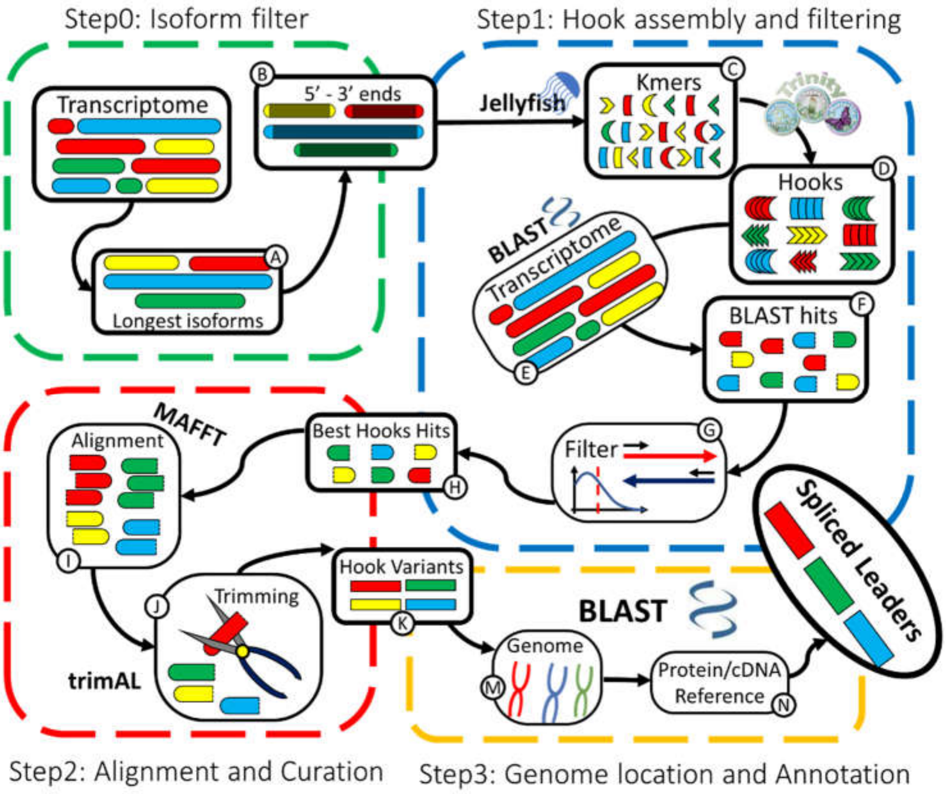


Fig. 3: The pipeline is divided in 4 scripts (Step0 to 3) to facilitate user intervention. Step0 reduces a Trinity assembly redundancy based on the contigs IDs and extract transcripts ends. Step1 generates Hook sequences for the putative Splice Leaders based on the Kmers present on the filtered sequences generated in Step0, identifies matching sequences in the original assembly and selects the Best Hooks based on their overall sequence orientation and frequency. The Best Hook Hits sequences are then aligned and trimmed in Step2 to generate Hook Variants, and Step 3 localizes them on a Reference genome and classify hits according to the presence of a potential Donor Site. In detail, longer isoforms are retrieved from the transcrptiome (A), both terminals regions are keep (B) and kmers are counted with Jellyfish and selected according to their abundance (C). Selected kmers are assembled into Hooks using Inchworm (D) and then Hooks are blasted against the original unfiltered transcriptome (E) and then filtered according to coi and abundance. The Best Hook matching sequences are retrieved (H). These sequences are then aligned with MAFFT (I) and trimmed automatically with trimAL (J), results are Hook Variants that represent possible versions of the repeated sequence that generated the hook (including both real polymorphisms and sequencing errors) (K). Step3 locates locus codding for the Hook Variants in the species genome using BLAST

and identifies potential donor sites (M). Finally, putative coding sequences are discarded using a third BLAST search against a protein-coding or protein sequences reference databases (N).

SLFinder-step0

The objective of this script is to filter a transcriptome assembled *de novo* with Trinity. The longer isoform for each gene is retrieved based on the contigs IDs (Fig. 4). Not because we expect that longer isoforms were produced by SL trans-splicing (on the contrary) but to account for incomplete transcript assembly when using Poly-A capture. Once identified, their full sequences are retrieved using “seqkit grep” and the terminal regions with “seqkit subseq”.

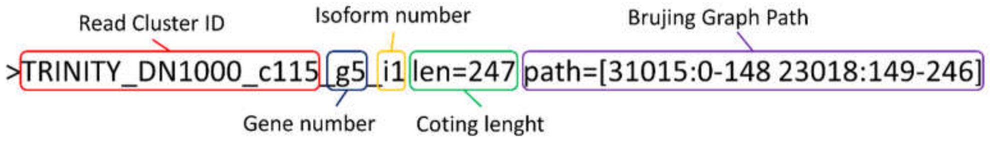


Fig. 4: Contig name convention used by Trinity.

Three output files per sample are stored on the folder set with “-F”: [Sample]_LongerIsoforms.id with the IDs of the longer isoforms, [Sample]-LongerIsoforms.fa with the complete sequence of the longer isoforms and a third with a user given extension (-f parameter) with the 5’ and 3’ ends of the longer isoforms.

The behavior of Step0 can be customized to a degree with the following parameters:

-a, --assemblies: Extension string identifying the assemblies to be analyzed. As the script is designed to work on Trinity assemblies. By default is “_Trinity.fasta”

-f, --filtered: Extension string identifying the filtered assemblies (5’ and 3’ ends of the longer isoforms). By default is “_Trinity.filtered”

-F, --fi_folder: Folder containing the filtered assemblies. By default is “Filtered_Assemblies”

-n, --idnum: This is the maximum number of contig IDs to retrieve from the assembly at a given time with seqkit grep. While efficient, this program can give an out of memory error if given too many entries to search are given. So, the [Sample]_LongerIsoforms-ID is sub divided by lines and the sequences retrieved part by part. Default value is 50,000

--s, --seqcut: Sequence length to retrieve from each contig end, default value is 50 bp.

-t, --threads: Number of threads to use when parallelization is possible (seqkit grep in this script), default value is 1.

This step is numbered zero because this part can be easily replaced by other strategies that serve the same end. For example, the sequences of both ends for all contigs can be retrieved and clustered using CD-HIT-EST or the transcriptome can be annotated previously and only one of the transcripts for each gene considered according other criteria. If for some reason a different strategy would work better on your data, you can still use the rest of this pipeline (see below).

SLFinder-step1

This script takes the filtered transcriptome and generates "Hook" sequences that are used to "fish out" common sequences on the original assembly, among which should be the SLs if present on the transcriptome using a BLAST search. In order to work, the script must receive a string identifying the original assemblies and associated filtered sequences (options "-a" and "-f" in the previous script and this one).

-a, --assemblies: Extension string identifying the assemblies to be analyzed. As the script is designed to work on Trinity assemblies. By default is "_Trinity.fasta"

-f, --filtered: Extension string identifying the filtered assemblies (5' and 3' ends of the longer isoforms). By default is "_Trinity.filtered"

-F, --fi_folder: Folder containing the filtered assemblies. By default is "Filtered_Assemblies"

First, the Kmer present on the filtered sequences are counted with "jellyfish count -C" and those with less than a user given cutoff counts discarded and converted in a Fasta file. To preserve frequency information in the next step, each Kmer sequence is duplicated as many times as it was observed before been assembled by Inchworm. The behavior of this process can be controlled with the following parameters:

-kc, --kmercount: Percentage threshold of Kmer observations required to be analyzed. It must be a float number representing (Default value is 0.0005). Increasing this number will increase the strictness of the analyses but it will reduce the ability to identify low frequency SLs. Testing shows that increasing this value above 0.001 can greatly reduce the detection power of the pipeline, even when the SL Tran-Spliced transcripts are abundant but there is biological diversity in their sequence (i.e. SL-2 in *Caenorhabditis elegans*).

-kl, --kmerlenght: Kmer length to count. Longer Kmers will greatly reduce false positives, however they impose a hard limit on the length of the SL that can be detected. Default value is 20.

-ki, --inchwormk: Kmer size used by Inchworm to conduct the assembly. This value must be between 10 and 32, and also be smaller than the value provided with “-kl”. Smaller numbers might lead to chimeric hooks that do not match the original transcriptome while longer values create a larger number of redundant hooks. Default value is 15.

-m, --hookmin: Minimal hook length to analyze. This serves to discard Hooks composed of very few Kmers and simplify results. Default value is 20.

-c, --cdhitH: Sequence identity threshold used for Hook filtration by clustering similar sequences with CD-HIT-EST. As stated before, its common that several hooks are created for the same sequence. This option allows to filter these hooks early on the pipeline and reduce running time. Default value is 1 (100% sequence identity).

For simplicity, Hooks keep the ID given by Inchworm but the “;” is replaced by “-” to prevent issues with special characters. Once assembled, the script searches the location of all hooks in the unfiltered transcriptome (necessary for Hook filtering and sequence retrieval) by a BLAST search “blastn -task blastn-short”. It’s possible to change the stringency of the BLAST with the following parameters. Since the Hook sequence might include bp that aren’t present in any of the transcripts a 100% identity match is not advised. Also, a more permissive configuration allows to detect SL variants that are not represented in the hooks but are similar in sequence (i.e SL-2 variants in *C. elegans*).

-bi, --blastnI: Identity threshold used to identify Hook matches in the unfiltered transcriptome. Default value is 90 (2 mismatches in a 20 bp Match)

-ba, --blastnA: Minimum hook Match length. Default value is 15.

The results are then filtered according to the coi and relative frequency by the Observation count threshold. These can be controlled with the parameters:

-i, --iorent: Float number representing the cutoff threshold for the ci. Testing shows that Hooks from true SLs have a value close to 1, but it’s not uncommon for them to have few instances of transcripts having matches in unexpected orientations so a value of 1 is not advised. Default value of 0.95

-me, --medianFilter hooks according to the median value of their matches count: T (True) or F (False). If this is set to "F" the observation count threshold is set to 1 instead. Default value is "T".

Transcripts presenting multiple hits for the same Best Hook are excluded from further analysis and are separated to be inspected. Once completed, a blast search of hooks passing the filters against themselves (100% Identity cutoff) in order to detect anomalies as redundancy on sequence or internal repeated sequences. And sequences matching all best hooks is retrieved to be used for SLFinder-step2. To prevent issues during the next step, sequences are separated by the Match location and orientation and named: [HookID]-[contig end]-[match orientation]-sequence (e.g. a1-4907-3prima-forward-sequence). This division helps to reduce running times during the sequence alignment in Step 2 and prevents trying to align the same sequence on forward and reverse strands.

To compensate the potential partial recovery of SLs mentioned earlier, three additional bp are included toward the center of the contig. Another complication that can happen in some assemblies is the presence of non-conserved sequence toward the 5' of the assembled transcripts that likely doesn't have a biological origin, even when a SL sequence is observed. These can greatly increase running time and be detrimental during SLFinder-step2. Our solution is to trim the raw matches using "seqkit subseq", keeping XX bp counted from the more central match coordinate (the presumed 3' end the SL). The sequence length to conserve can be controlled with:

-s, --seqtrimm: Trimm Raw matches larger than X, counting from more central position of the match. Default value is 50

Other important parameters to consider are:

-o, --outdir: Directory where to store the output files, it's important that no directory with this name exists. The default name is "SL-analysis"

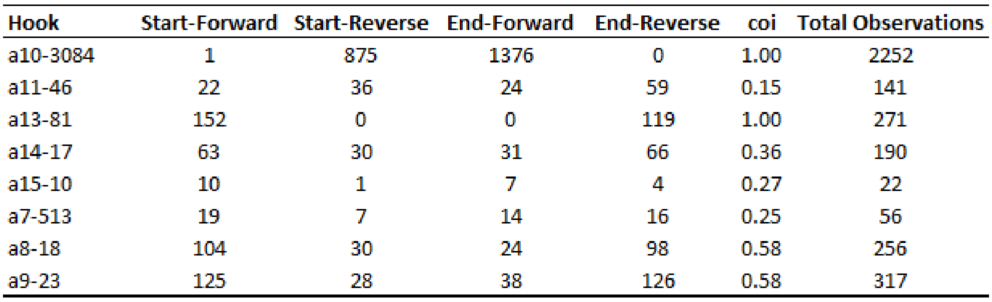
-M, --Tmaxmemory: Maximum suggested max memory to use by Trinity in Gigabytes Default value is 1

-t, --threads: Number of threads to use when parallelization is possible (BLAST, Jellyfish, and Trinity in this script), default value is 1.

Main results are provided in four files in Results:

1) Step1-All_Hooks: Table with the matches counts for all Hooks on each location and orientation. It includes the hooks, n° observations at the start or the end of the transcripts either matching as forward or reverse, Concordance Index and total

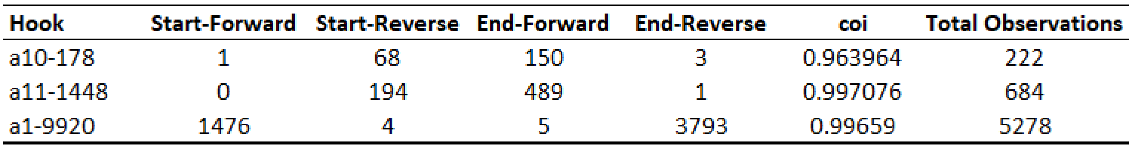
number of observations (Table 1). Since contigs with multiple matches aren't considered some hooks might be missing from the Step1-All_Hooks. This usually happens with low complexity hooks made of repetitive short sequences.



Hook	Start-Forward	Start-Reverse	End-Forward	End-Reverse	coi	Total Observations
a10-3084	1	875	1376	0	1.00	2252
a11-46	22	36	24	59	0.15	141
a13-81	152	0	0	119	1.00	271
a14-17	63	30	31	66	0.36	190
a15-10	10	1	7	4	0.27	22
a7-513	19	7	14	16	0.25	56
a8-18	104	30	24	98	0.58	256
a9-23	125	28	38	126	0.58	317

Table 1: Example of a Step1-All_Hooks table.

2) Step1-Best_Candidates: Same as Step1-All_Hooks but only with the Best Hooks, plus the Concordance Index and total count number (Table 2).



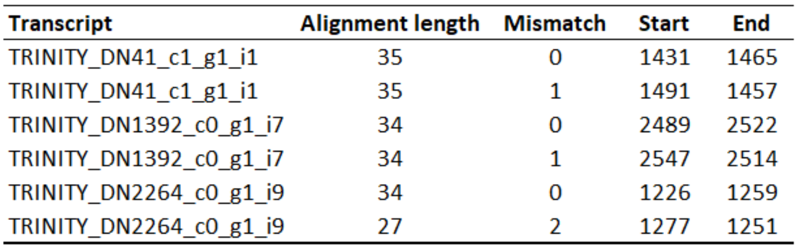
Hook	Start-Forward	Start-Reverse	End-Forward	End-Reverse	coi	Total Observations
a10-178	1	68	150	3	0.963964	222
a11-1448	0	194	489	1	0.997076	684
a1-9920	1476	4	5	3793	0.99659	5278

Table 2: Example of a Step1-Best_Candidates.

3) Best-Hook.fasta: Sequences of the Best hooks

Additionally, three folders will be added inside Results:

1) Hooks: Sequence of all the Hooks and the CD-HIT-EST cluster report
2) Raw_Matches: Best Hooks matching sequences on the transcriptome
3) Anomalous_Matches: Transcripts with multiple matches for the same Best Hook and the simplified BLAST results (Table 3), separated by Hook.



Transcript	Alignment length	Mismatch	Start	End
TRINITY_DN41_c1_g1_i1	35	0	1431	1465
TRINITY_DN41_c1_g1_i1	35	1	1491	1457
TRINITY_DN1392_c0_g1_i7	34	0	2489	2522
TRINITY_DN1392_c0_g1_i7	34	1	2547	2514
TRINITY_DN2264_c0_g1_i9	34	0	1226	1259
TRINITY_DN2264_c0_g1_i9	27	2	1277	1251

Table 3: Example of the simplified BLAST results for Transcripts with multiple hits for the same Best Hook (i.e. a23-2035)

A summary of all results is also provided on the file Step1_Summary located on the main Analysis Directory (the one provided with "-o" parameter). It includes the relevant parameter values used in the run, transcriptome files analyzed with the number of transcripts and the kmer counts filter applied to them, the tables from the files "Step1-All_Hooks table" and "Step1-Best_Candidates", and the matches number found for each hook divided by transcript end and orientation.

Important note: *Changing kmer size was removed from Trinity v2.9.1 but restored as a hidden parameter (changed from "--KMER_SIZE" to "--__KMER_SIZE") in Trinity v2.10.0, reason why SLFinder is incompatible with older versions. However, the script doesn't verify this before running. So, if the SLFinder-Step1 fails and the screen is full of "file not found" error messages this the most likely reason.*

Negative results in SLFinder-step1

Sadly, here is where the "good enough" nature of SLFinder comes into play. Step1 can fail to identify SLs for a number of technical issues. Options -kl, -m and -kc set a hard limit on how many and how long distinct SL variants need to be in order to be detected. These issues could be easily be solved by knowing the SL sequence and fine tuning the parameters, if your assemblies contain the SL trans-spliced transcripts in enough quantity. But if you do you shouldn't be using this software in the first place.

When this happens, the best option is to relax progressively reduce these parameters and set "-me" to FALSE, and then processing all Hooks with a high enough coi. But keep in mind that the lower these parameters the greater the chances of picking up any other sequence. For example, an early version of this software detected a SL candidate in Mus musculus that turned out to be Histones transcripts. If suitable hooks are obtained but very few hits are reported, try changing "-ba" and "-bi".

SLFinder-step2

This script takes the Best Hooks raw matches sequences recovered on the previous step, filters them by size and sequence similarity, aligns them and then automatically trims the results to generate SL sequences. During its run, CD-HIT-EST is used in many steps to reduce sequence redundancy and shorten running times.

The initial filtering is carried out with "seqkit seq" to filter sort sequences and then cd-hit-est. They can be controlled with:

-c, --cdhitR: Sequence identity threshold used for Raw Matches filtration by clustering sequences with CD-HIT-EST. Default value is 1 (100% sequence identity).

-m, --minimun: Minimal Raw Match length to consider. Default value is 1 (No filtering)

Then the sequences are aligned using “mafft --globalpair”. While intended for a precision alignment of a small dataset, the relative short sequences (50 bp maximum by default) allow to align several thousand sequences in minutes. Trimming is then carried out with Trimal and it can be controlled using:

-ts, --trimalsm: Similarity score used in trimAL. Default value is 0.5

-tg, --trimalgap: Gap threshold used in trimAL. Default value is 0.90

Important note: *these default parameters were set using trimAL version 1.4rev59 (version available on Bioconda). The older 1.2rev59 is far more aggressive and is likely to decimate Hook Variants with this set up. When using the older 1.2rev59 version (as used in the publication paper), set -ts to 0.001 to compensate. Always review the alignments. The pipeline is subdivided like this for a reason.*

To ensure that the trimmed sequence properly represents the original region (i.e. no central position was removed, hindering SLFinder-step3 approach). The central section is retrieved using with “seqkit subseq -r 2:-2", gaps removed and the sequence searched against the original alignment with “seqkit grep -s -r”. These bases are removed because more likely to be errors and terminal mismatches here aren’t critical on SLFinder-step3, unless “-sc” is set to 100 (see below).

If one of the sequences doesn’t match the automatic process for that file ends (we recommend manual curation of the alignment). Passing this control, gaps are removed and clustered with CD-HIT-EST again with a 100% identity threshold. First creating an individual file and then combining al sequences. In addition, a sequence logo is generated for the alignments pre and post trimming (Fig. 5) intended to aid the manual curation.

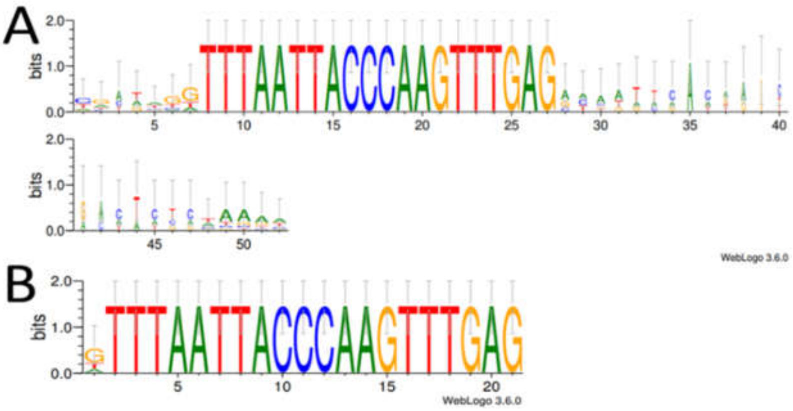


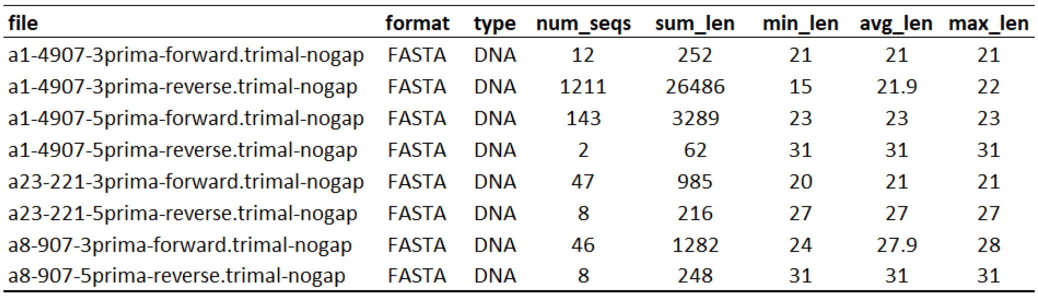
Fig. 5: Example of Sequence Logos for a hook matching SL-1 of C. elegans before (A) and after (B) the trimming process. Since the symbol "-" is not included in the Weblogos alphabet, positions with smaller letters are positions with more GAPs.

Other options to consider are:

-o, --outdir: Directory where to store the output files, must be the same as the string used on SLFinder-step1. The default name is "SL-analysis"

-t, --threads: Number of threads to use when parallelization is possible (CD-HIT-EST and MAFFT), default value is 1

A small overview of the results is provided in the file Step2_Trimal-Stats, which contains Basic stats of the trimmed sequences (seqkit stats). It includes the number of sequences, total length (irrelevant for this application), min, max and average sequence length (Table 4).



file	format	type	num_seqs	sum_len	min_len	avg_len	max_len
a1-4907-3prima-forward.trimal-nogap	FASTA	DNA	12	252	21	21	21
a1-4907-3prima-reverse.trimal-nogap	FASTA	DNA	1211	26486	15	21.9	22
a1-4907-5prima-forward.trimal-nogap	FASTA	DNA	143	3289	23	23	23
a1-4907-5prima-reverse.trimal-nogap	FASTA	DNA	2	62	31	31	31
a23-221-3prima-forward.trimal-nogap	FASTA	DNA	47	985	20	21	21
a23-221-5prima-reverse.trimal-nogap	FASTA	DNA	8	216	27	27	27
a8-907-3prima-forward.trimal-nogap	FASTA	DNA	46	1282	24	27.9	28
a8-907-5prima-reverse.trimal-nogap	FASTA	DNA	8	248	31	31	31

Table 4: Example of a Step2_Trimal-Stats file.

Additionally, two folders will be added to Results:

1) Hool_Variants: Trimmed sequences in Fasta format for each original alignment file (folder Hook_Individual), combined Hook_all_variants.fa and their respective cluster reports
2) Weblogos: Sequence logos for the raw and trimmed alignments (folders alignments and trimed_aligments respectively).

A summary is provided on the file Step2_Summary located on the main Analysis Directory. It includes the relevant parameter values used in the run, hook variants generated and the subsets of sequences that were discarded.

Negative results in SLFinder-step2

Unlike the previous step, a negative result at this stage respond mostly to technical issues. Assuming enough data was retrieved in Step1 (more than one sequence in at least 1 file), or the trimming parameters were too restrictive for your dataset. Try relaxing

"-ts" and "-tg" and setting an appropriate "-m". It might be a good idea to copy MAFFT alignments in a different folder and test trimal parameters first. Important to note: "-ts" in SLFinder is "-st" in trimal and "-tg" is "-gt".

SLFinder-step3

The final steps starts by making a BLAST search of all Hook variants (file: "Analysis directory"/Results/Hook_Variants/Hook_all_variants.fa by default) against the Reference Genome. Once identified, the script determines loci orientation by checking how many times its sequence is present, both in forward and reverse, on the raw sequences using "grep -c". Twenty percent of the loci length bp from both ends to account for incomplete sequencing. Since these are separated by transcript end and hook orientation, finding more hits in a file indicates orientation (only the hook sequences that generated the best Hook Variant are checked). According to the results, the pipeline continues by searching for the donor site in the 3' end of the Loci ("GT" at the end of the Loci if it is encoded in Forward or "AC" at the beginning if in Reverse). If for some reason the Loci's sequence is found an equal number of times present on Forward and Reverse (or it isn't found on the Raw Sequences) the orientation is reported as "undetermined". Finally, the region surrounding the loci is annotated using a Protein/cDNA reference.

The script requires two inputs: A Reference Genome and an external reference with known proteins or cDNAs (e.g. Swiss-Prot from Uniprot). They can specify with these options:

-g, --genomeblast: Reference Genome. Must be an uncompressed Fasta file.

-br, --blast_ref: Protein or Reference database to be used to annotate the pSL genomic Loci. Must be an uncompressed Fasta file.

The pSL vs Reference Genome BLAST search is done with "blastn -task blastn-short", settled as ungapped and with a 100% identity threshold. Query coverage can be changed to account for non-matching bp on the terminal regions (likely caused because sequencing errors or problems during the trimming process). A higher value reduces false positives, however if set at 100 some true SL sequences may not be detected due to problems during the automatic trimming.

-gc, --gen_cov: Coverage threshold of the pSL hits on the genome. Default value is 90 (up to two not matching bp in a 20 bp Hook Variant).

Hits are then sorted by starting coordinate and the individual pSL each Loci identify by checking which pSL hits overlap and if there are anomalies. If the pSL variants are in fact SL sequences they should overlap almost on their entirety. The pipeline check this by taking the first hit according to its coordinates and checking if the following matches overlap with it. If it does and the End coordinate is further away the loci coordinates are extended. Depending on the how big is the last loci compared with the longest hit the scripts registers an alert number: 0 (no problem) if its extended less than 2 bp, 1 (acceptor site determination might not reliable for some of the pSL variants) if its extended between 2 and five bp, 2 (acceptor site determination is not reliable and it is not analyzed) if between 2 and 5 bp and 3 (probably a microsatellite) if more than five. These configurations are illustrated on Fig 6.

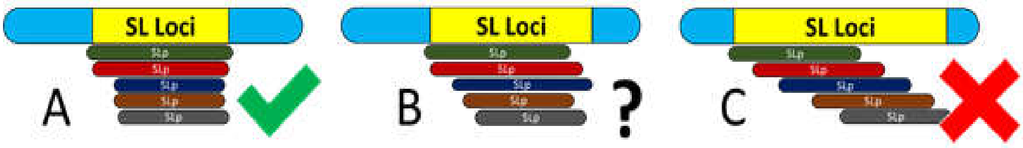
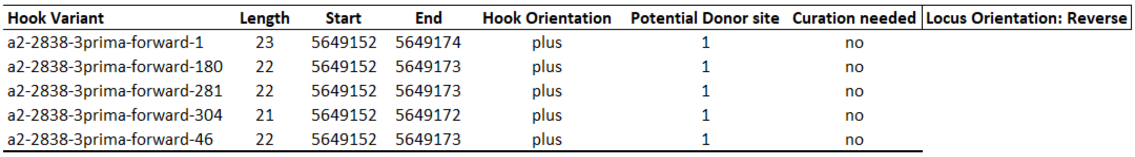


Fig. 6: Possible ungapped alignment configuration between pSL variants and their codding loci. A) represents the ideal situation where all pSL matching the loci overlap almost entirely (Alert 0) while C) is likely false positive in a low complexity region (Alert 3). Configuration B (Alerts 1 and 2) are given where some pSL extend slightly more than the others is more likely to be caused by retention of some bp from the transcripts that also match with the genome.

As mentioned above, Donor sites are identified by simply counting either “GT” or “AC” in the 3’ end of the pSL according to its orientation. Testing however shows that is common the inclusion of a limited number of base pairs at both the 5’ and 3’ ends of the Hook Variants that can then be incorporated in the reported pSLs when they match the Genomic sequence. These often are “G” or a “GT” matching the real Donor site.

Because of this, the script first scan 4 bp surrounding the pSL 3’ coordinate to detect the donor site, and then verifying if it is or isn’t included in the hook variants. Also, Loci with Alerts 2 or 3 are not analyzed and require manual inspection. The information needed will be stored in the folder “Genome_locis” and divided by chromosome (or contig/scaffold name): A table with all pSL variants matching the loci and an alignment produced with MAFFT (Table 5 and Fig 7). Another situation that can happen is that only some of the Hook Variants matching the loci have a potential donor site in the expected



Hook Variant	Length	Start	End	Hook Orientation	Potential Donor site	Curation needed	Locus Orientation: Reverse
a2-2838-3prima-forward-1	23	5649152	5649174	plus	1	no	
a2-2838-3prima-forward-180	22	5649152	5649173	plus	1	no	
a2-2838-3prima-forward-281	22	5649152	5649173	plus	1	no	
a2-2838-3prima-forward-304	21	5649152	5649172	plus	1	no	
a2-2838-3prima-forward-46	22	5649152	5649173	plus	1	no	

region. If more than 20% have this issue the loci is reported as “Unclear” and it will be treated separately (see below).

Table 5: Example of an individual loci table detailing each matching pSL, its length, coordinates in the chromosome, orientation, potential donor sites and if curation is advised: there was no potential donor site, or the Hook Variant overlaps with it. Since the program takes four bases surrounding the 3’ region coordinates these values can be 0, 1 or 2. The locus orientation is included on the Header.

```
                                          10        20
                                 ....|....|....|....|....|
SL-1_Celegans                    GGTTTAATTACCCAAGTTTGAG---
SL_Coding_locus                  GGTTTAATTACCCAAGTTTGAGGTA
a11-1448-3prima-forward-1        GGTTTAATTACCCAAGTTTGAGA
a11-1448-3prima-forward-189      GGTTTAATTACCCAAGTTTGAGCT
a11-1448-3prima-forward-214      tGTTTAATTACCCAAGTTTGAGG
a11-1448-3prima-forward-242      GGTTTAATTACCCAAGTTTGAG
a11-1448-3prima-forward-25       GGTTTAATTACCCAAGTTTGAGA
a11-1448-3prima-forward-27       GGTTTAATTACCCAAGTTTGAGC
```

Fig 7: Alignment between the Loci and each matching Hook variant, obtained with MAFFT. The potential donor site is highlighted.

The next step is a BLAST search against the external reference for annotation. Since most false positives with a biological origin at this stage are likely 5’ UTR sequences a cDNA reference is preferable, but often this is not available. Also, if one of the cDNAs includes the SL sequences by error the SL will be wrongly discarded. This search can be customized with the following parameters:

-bt, --blast_ty: Reference database type: “nucl” or “prot” for nucleotides and protein, depending on the selection, the script runs blastn or blastx. By default is “prot”.

-be, --blast_ev: Order of magnitude for the evalue cutoff used in the BLAST annotation, the program reads it as '1e-'[YOUR VALUE]. Default value is 10, for a 1e-10 cutoff.

-gr, --rangeG: Region surrounding the pSL hit to annotate. Default value is 100.

For a pSL to be considered validated three points need to be met

1) There must be a matching locus in the Reference Genome
2) A potential donor site must be present
3) No hit must be found in the annotation BLAST.

Sequences are retrieved according to their orientation (Leaving de donor side "GT"), clustered with CD-HIT-EST (100% identity threshold) and reported in "SL.fa". The same is done with Loci with an "Unclear" donor site and are reported in the file "Unclear_SL.fa". In both files each pSL heather will include the Loci number and the hook ID that identified it and had an identifiable donor site (referred as "Best SL ID" in Step3_Loci.tab, see below). Lastly, the transcripts IDs with those hooks are retrieved from the temp_files folder, counted and organized in the folder "Spliced_leader_transcripts". One important note: these are all transcripts with hits for the original hook, not the ones bearing the pSL variant found in the genome. They are provided as a quick overview of how well supported is the pSL and facilitate to discard false positives. See "Working with the pSL sequences" for details along with SLFinder-Genes.

Other options to consider are:

-o, --outdir: Directory where to store the output files, must be the same as the string used on SLFinder-step1. The default name is "SL-analysis".

-s, --pslfile: File name with trimming information needed to retrieve the pSL sequences (Default: "Analysis directory"/Results/Hook_variants/Hook_all_variants.fa)

-t, --threads: Number of threads to use when parallelization is possible (BLAST, CD-HIT-EST and MAFFT), default value is 1

Main results are provided following files:

1) Step3_Loci.tab: Table summarizing the SL Locus. It includes their location, longest and best SL hits, donor sites present and the annotation Blast search results (Table 6).
2) pSL_Loci.fa/pSL_Loci_Unclear.fa: Fasta file with the validated SLs loci sequences found with a clear/unclear donor site.
3) SLCLases.fa/pSL_Loci-Unclear.fa: Fasta file with the clustered SL sequences by cd-hit-est, with a clear/unclear donor site.
4) Step3_Spliced_transcripts: Matching transcript counts for each hook with at least one "Best pSL ID" in a Loci, across all analyzed transcripts (Table 7).

Additionally, two folders will be added inside Results:

1) SL_BLAST: Results of the SL Blast search against the reference genome
2) Genome_locis: Each loci hits summary (table with matching pSL variants to the loci and alignment file) separated in folder per chromosome, and results of the annotation BLAST (Blast_annotation folder).

3) Spliced_leader_transcripts: Folder with the Transcripts IDs of the Hooks that identified the SLs on each assembly, separated on different folders.

A summary is also provided on the file Step3_Summary located on the main Analysis Directory. It includes the relevant parameter values used in the run, the total number of pSL Loci detected and donor site detection count. It also includes tables for the files Step3_Spliced_transcripts and Step3_Loci.tab, however Step3_Loci.tab is subdivided according to the donor site detection and the clusters generated by CD-HIT-EST to identify representative pSL sequences. First are Loci with a clear donor site and each Cluster (with the Loci representative of the Loci and it's sequence), the Unclear Loci and their clusters, and the discarded Loci (Loci close to a CDS sequence, No donor site detected and Loci not analyzed).

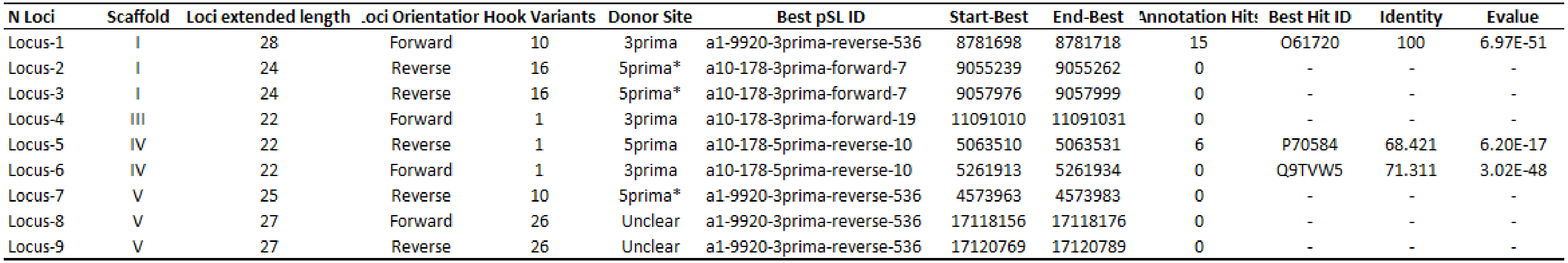
Warning files

Depending on your data and the references used several problems can arise at different steps of the pipeline. Reports of these are saved on the Warnings folder.

1) Step1-Hook_Anomalies: Results of the self-BLAST search among the Hooks. It only contains Query Hook, Subject Hook and alignment Start and End positions in the query (Table 8).
2) Step2_Manual-curation: Report of the files that require to be manually curated (Trimal brought no results, sequence was too short, or the center was modified)
3) Step3_Sequence-retrieval-warnings: Report with the Loci were not enough sequence could be recovered for the annotation BLAST search or even the Donor site determination. This happens if the pSL is located too close to the end of a chromosome/coting in the reference genome.
4) Step3_Problematic-Loci: Reports loci whose apparent length when recovering the genomic loci is larger than the largest individual hit. This can happen in low complexity regions.

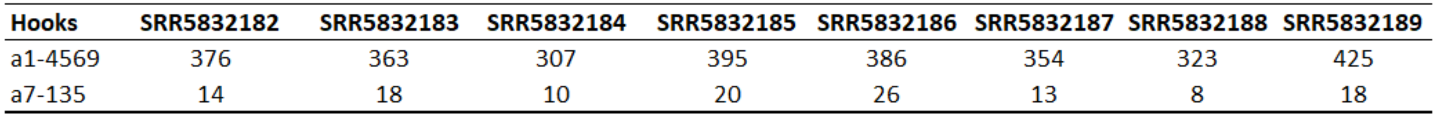
Negative results in SLFinder-step3

Total negative results (i.e. no pSL locus found) means that the BLAST couldn't locate any of the Hook Variants in the reference genome. Assuming the sequences are truly SLs, there are two likely causes: 1) The pSLs are absent from the Reference Genome assembly (in which case there is little to do beyond sequencing and/or assembling a new one) or 2) The Hook Variants require further trimming. Either re-run Step2 with more restrictive parameters or consider to manually curate the alignments. Same if many pSL are identified with an unclear or no donor site.



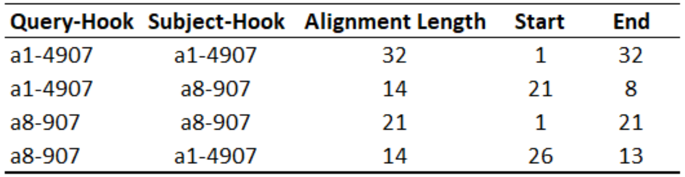
N Loci	Scaffold	Loci extended length	.oci Orientatior	Hook Variants	Donor Site	Best pSL ID	Start-Best	End-Best	\nnotation Hit:	Best Hit ID	Identity	Evalue
Locus-1	I	28	Forward	10	3prima	a1-9920-3prima-reverse-536	8781698	8781718	15	O61720	100	6.97E-51
Locus-2	I	24	Reverse	16	5prima*	a10-178-3prima-forward-7	9055239	9055262	0	-	-	-
Locus-3	I	24	Reverse	16	5prima*	a10-178-3prima-forward-7	9057976	9057999	0	-	-	-
Locus-4	III	22	Forward	1	3prima	a10-178-3prima-forward-19	11091010	11091031	0	-	-	-
Locus-5	IV	22	Reverse	1	5prima	a10-178-5prima-reverse-10	5063510	5063531	6	P70584	68.421	6.20E-17
Locus-6	IV	22	Forward	1	3prima	a10-178-5prima-reverse-10	5261913	5261934	0	Q9TVW5	71.311	3.02E-48
Locus-7	V	25	Reverse	10	5prima*	a1-9920-3prima-reverse-536	4573963	4573983	0	-	-	-
Locus-8	V	27	Forward	26	Unclear	a1-9920-3prima-reverse-536	17118156	17118176	0	-	-	-
Locus-9	V	27	Reverse	26	Unclear	a1-9920-3prima-reverse-536	17120769	17120789	0	-	-	-

Table 6: example of a Step3_Loci.tab table. It summarizes the Loci location, length, matching SL, donor site and annotation results. Best pSL ID is the longest matching Hook variant with an identified donor site if present, the longest one if doesn't. The asterisk "*" in some Loci reported either 3 or 5prima Donor indicates that the best Hook variant overlaps with the potential donor site.



Hooks	SRR5832182	SRR5832183	SRR5832184	SRR5832185	SRR5832186	SRR5832187	SRR5832188	SRR5832189
a1-4569	376	363	307	395	386	354	323	425
a7-135	14	18	10	20	26	13	8	18

Table 7: example of Step3_Spliced_transcripts file.



Query-Hook	Subject-Hook	Alignment Length	Start	End
a1-4907	a1-4907	32	1	32
a1-4907	a8-907	14	21	8
a8-907	a8-907	21	1	21
a8-907	a1-4907	14	26	13

Table 8: Example of a Step1-Hook_Anomalies. Notice that includes the line of each hook with itself, this redundancy helps when the hook has a repetitive sequence and matches with itself multiple times.

7) Manual curation, what to look for?

As detailed in (Calvelo XXXX), despite the strategies implemented to minimize to recover and pinpoint how long the detected SLs are, SLFinder can't guarantee a clean SL sequences in all cases since relies on alignment quality and differences between the transcripts and the slRNA genes. A reasonable assumption that doesn't on with a single base precision. Here we describe the known artifacts that need to be verified and curated before using SLFinder results on further analysis.

1) Overextension toward the 3' end

The most common example is the presence of an extra "GT" that coincides with the Donor Site in the slRNA. Since the chance of having these 2 bp in a transcript after SL incorporation are rather high and a single Hook Variant featuring them is enough to (a consequence of the clustering process in Step2 and the selection of the longest matching Hook Variant to process the pSL Loci in Step3. However, in some cases this can extend for several bases (Fig. 8). This can lead misreporting of several pSL-Loci, especially when very few Hook Variants align with the Loci.

```
                           10        20
                  ....|....|....|....|....|..
SL-1_Celegans     GGTTTAATTACCCAAGTTTGAG-----
Locus-15          GGTTTAATTACCCAAGTTTGAGGTAAA
Hook_Variant-1    GGTTTAATTACCCAAGTTTGAGGTAAt
Hook_Variant-2    GGTTTAATTACCCAAGTTTGAGGTAgg
Hook_Variant-3    tGTTTAATTACCCAAGTTTGAGGTAAA
Hook_Variant-4    GGTTTAATTACCCAAGTTTGAGGTAct
Hook_Variant-5    GGTTTAATTACCCAAGTTTGAGGTAAA
Hook_Variant-6    GGTTTAATTACCCAAGTTTGAGGTAAc
Hook_Variant-7    GGTTTAATTACCCAAGTTTGAGGTAAg
```

Fig. 8: Alignment between the known SL-1 in C. elegans, the detected slRNA loci (Locus-15 in Calvelo XXXX) and some of the Matching hook Variants. The true Donor Site is displayed in blond letter and mismatches in lower caps. Under this circumstances SLFinder would select the highlighted Hook_Variant-5, provided there is a GT afterwards.

2) Loss information toward the 5'

The main drawback of working with poly-A enriched data is the loss of coverage toward the 5' end of the transcripts where the SL are located. While SLFinder has shown to be able to recover the close to full sequence of long SLs in Step1 during the Hook Assembly, the great majority of transcripts only bears a partial sequence of the SL, resulting on shorter sequence (Fig. 9).

```
                         10        20        30
                 ....|....|....|....|....|....|....|....
SL_Smansoni      AACCGTCACGGTTTTACTCTTGTGATTTGTTGCATG---
a23-2035_R       --CCGTCACGGTTTTACTCTTGTGATTTGTTGCATGata
Smansoni_pSL-1   ----------GTTTTACTCTTGTGATTTGTTGCATG-
Smansoni_pSL-2   -----------TTTTACTCTTGTGATTTGTTGCACG-
Smansoni_pSL-3   -----------CTTTACTCTTGTGATTTGTTGCATG-
Smansoni_pSL-4   ------------TTTACTCTTGTATTTTGTTGAATGg
```

Fig. 9: Alignment between the known SL sequence for *S. mansoni*, the generated Hook in Step1 and the generated pSLs in step 3 (see for details Calvelo XXXX).

The sequence loss materializes on Step2 during the trimming process. If detected, the problem can be alleviated by re-running Step2 with different -ts and/or -tf values, or by manually process the Hook Variants (see below). Note however that submitting longer Hook Variants in Step3 will likely result in fewer results than shorter ones as Step3 seeks hits alignments with 100% identity with BLAST. Hallowing mismatches on the terminal regions by changing the query coverage.

3) Potentially non-functional slRNA coding loci

Rather than an artifact of SLFinder per se, a shortcoming of making so limited assumptions about the SL sequences is the impossibility to identify SL pseudogenes. SLFinder will report as a pSL locus any region that matches Step3 thresholds. This includes some closely clustered SL exonic sequences (less than 10 bp) that can be easily filtered by checking their location. Non straightforward so loci are some of the anomalies reported in our paper Calvelo et al. XXXX: Potential Donor Sites that imply 1pb shorter than already reported donor sites and loci with partial repeats of the 3' region (Fig. 10).

```
a)                     10        20
                  ....|....|....|....|....
SL-1_Celegans     GGTTTAATTACCCAAGTTTGAG--
Locus-1           GGTTTAATTACCCAAGTTTGAGGT
Locus-12          GGTTTAATTACCCAAGTTTGAGTT
```

```
b)                       10        20        30        40
                 ....|....|....|....|....|....|....|....|...
SL-B1_Hvulgaris  AAACTTTTTAGTCCCTGTGTAATAAG-----------------
SL-B1_Hvulgaris  **************************TCCCTGTGTAATAAG
Locus-6           AAACTTTTTAGTCCCTGTGTAATAAGTCCCTGTGTAATAAGTT
                                          ***************

Locus-29          AAACTTTTTAGTCCCTGTGTAATAAGTCCCTGTGTAATAAGGT
                                          ***************

Locus-36          AAACTTTTTAGTCCCTGTGTAATAAGTCCCTGTGTAATAAGTT
                                          ***************
```

Fig. 10: Examples of potential pseudogene copies of different SLs found during the validation of this software a) SL locus that implies an SL with one less bp than the reported sequence as the last "G" forms part of the potential donor site. b) SL locus

that present a repetition of the 3’ end of the SL or other SL, likely due to transposition events. Often a closer inspection of the first case proves to be the second, but not always.

8) Working with the pSL sequences

If there are no surprises with your data and the sear was fruitful, you should have a manageable number of potential pSL. Next it is needed to discard false positives as Histone genes or adapters used during sequencing that for some reason made their way into the transcriptome assembly. Retrieving the contigs matching each hook with an pSL and annotating them is probably the most straight forward approach. After identifying and discarding those we recommend use another pipeline specialized on identifying the trans-spliced genes themselves given the known sequence of the SLs, like SL-Quant (https://github.com/cyaguesa/SL-quant). Note however that, in its current implementation (last checked in 31/08/2019) SL-quant requires the full sequence of the true SL sequence to work properly. Something that SLFinder cannot guarantee with 100% certainty for all possible datasets, though it is possible to get a longer sequence manually comparing the untrimmed alignments produced in Step2 with the pSL locis found in Step3. As an alternative we offer the script SLFinder-Genes (see below).

Remember that our pipeline is prone to report several artificial variants of the same SL by including extra bp at the sequence ends. Particularly “G” or “GT” in the 3’ end (that aligns with the donor site in the genome). So manual trimming of the pSLs might be required. For the same reason do not discard Loci with “Unclear”, “Not detected” or “Not Analyzed” donor sites without verifying the alignments stored on Genome_locis”

9) How to…

1) Use a transcriptome assembled with a program other than Trinity or filtered following a custom strategy.

As stated before, this can be done by properly using “-a” and “-f” in Step1. As long there is one sequence for every true gen (or close to) the pipeline should work without issue. The only thing important to consider is the transcripts headers. Make sure that each one has a distinct identifier and that they do not include special characters in regular expressions like points. Formats like Ensembl (e.g. ENSMUST00000021802.15) are ok because the point is always the character N°19, but if the position isn’t constant across transcripts Step1 is likely to fail. If your assembler does this replace the point for “_” or other non-special character before running the pipeline.

2) Filter hooks that are known to not be SL

If too many Best Hooks are generated running SLFinder-step1 it can save time to make a BLAST search against a cDNA reference to identify false positives (e.g. Histones) and/or remove anomalous hooks reported on "Step1-Hook_Anomalies" file. To do this just remove the hook sequence files located on: "Analysis directory"/Results/Raw_Matches

3) Use manually curated alignments on SLFinder-step 3

If needed, you can find the unaligned sequences on "$OUT_DIR/Results/Raw_Matches" and the alignments on "Analysis directory"/Results/pSL_variants/pSL_Individual, checking the sequence logos for the alignment might be also a good starting point located on "Analysis directory"/Results/weblogos/aligments to diagnose the problem. Once curated, the alignment can either be trimmed with Trimal and processed:

```
trimal -gt [0.90] -st [0.001] -fasta -in [input_file] | awk '/^>/ {printf("\n%s\n",$0);next; } {
printf("%s",$0);}   END {printf("\n");}' | awk 'NF' | grep -v ">" | awk -v data=[Identifier]
'{print ">"data"-" NR "\n" $s}' >> output_file
```

Red parameters need to be changed according to your file, blue ones are the default values used by SLFinder-step2. The rest of the command line linearizes the fasta sequence and renames each sequence as "Identifier"-Number. In the script this identifier is the one used for the files ([HookID]-[contig end]-[match orientation]) but there is no need to maintain it.

If you decide to also trim the alignment manually export it in Fasta format and run command for reformat and renaming:

```
awk '/^>/ {printf("\n%s\n",$0);next; } { printf("%s",$0);}     END {printf("\n");}' | awk
'NF' | grep -v ">" | awk -v data=[Identifier] '{print ">"data"-" NR "\n" $s}' >> [output_file]
```

Once completed, concatenate all sequence files and remove gaps running:

```
cat [your trimmed files] | seqkit seq -g >> [concatenated_file]
```

And finally, cluster sequences by sequence similarity:

```
cd-hit-est -c 1 -i [concatenated_file] -o [pSL_file]
```

Use the "-s" parameter in SLFinder-step3 to use this new pSL_file.

It's important that you do not change the name of the Hook Variants. This is used in Step3 to verify Loci Orientation.

10) SLFinder-Genes

1) Necessary input data and software

SLFinder-Genes is a script designed to identify SL acceptor genes based on the read data and the reference genome annotation. Even when the full sequence of the SL exon region is not known, the likely situation after using SLFinder.

The following software and packages need to be installed and added to the path:

- Bowtie2: http://bowtie-bio.sourceforge.net/bowtie2/index.shtml
- Cutadapt: https://cutadapt.readthedocs.io/en/stable/installation.html
- CD-HT-EST: https://github.com/weizhongli/cdhit
- Htseq-count: https://bioweb.pasteur.fr/docs/modules/HTSeq/0.5.3p9/count.html
- Samtools: http://samtools.sourceforge.net
- Seqkit: https://github.com/shenwei356/seqkit

Before continuing, a disclaimer: at its core, the script is blunt and uncomplicated. It is not efficient in run time and you will need to verify that the input files are properly recognized by seqkit.

These are the necessary input files are the following. In its current version SLFinder-Genes only verifies the existence of these files and not their format. Caution is advised.

- Read files in Fastq format. Both single end and paired end are supported but they must be located on the working directory. Since only Seqkit works directly with these files, files can be compressed with gzip.
- Fasta file with the SL tags (3' region of the SL to search for) and the full SL length. Tag sequences need to be at least 10 pb long.
- Optional: Reference genome sequence in fasta format (like the one used on Step3)
- Optional: Annotation of the reference genome in GTF format.

From here, the provided read files need to be checked. When working with paired end data "seqkit grep" needs to properly recognize reads id from their heather (e.g. "@READ1::XXX::XXX 1:N:XXXXXX"). Alternatively, read pairs should be identified with and ends with either "/1 or /2 (e.g. "@READ1::XXXX::XXXX/1"). Cutadapt on the other hand can be… "picky". Try running it before this script and address whatever problem is reported by it.

Another source of troubles is the SL tags file since (likely) will be made manually by you. It should look like the one shown in Figure 11. SLFinder-Genes will use these sequences to recover the reads with SL sequences (exact matches to the sequences provided) located near enough the reads ends according to the known entire SL length (that must be provided in the SL tag heather) and the -m option (see below).

```
>SL-A 33
CTAACTTGTATG
>SL-B 30
TTACCTTGGATG
```

Fig. 11: Example Fasta file with the SL tags

Each SL tag must have a unique ID, the SL sequence should be submitted in the sense orientation and separated by a white space the known/estimated/observed SL sequence length included on the header. To prevent redundancy in the results, the script first runs cd-hit-est with a 100% identity threshold on the tags. If any sequence is clustered with another the analysis is aborted. If in doubt, you can verify if it works as intended run the following commands:

- `grep ">" [SL_tags.fa] | awk '{print $1} | sed "s/>//"`

This should print only the headers, and the following only the SLs expected length:

- `grep ">" [SL_tags.fa] | awk '{print $2}`

2) Script options

Input options:

-g, --genome: Reference genome sequences in FASTA format (If none is provided SLFinder-genes will stop after separating the reads with SL hits)

-gt, --gtf: Reference genome annotations in GTF format (Must be included along with the reference genome.)

-r1, --read1: String used to identify left paired end reads files (must be on the working directory) (Default: _1.fastq)

-r2, --read2: String used to identify right paired end reads files (must be on the working directory) (Default: _2.fastq)

-rs, --readS: String used to identify single end reads files (must be on the working directory, ignored if -p is set to TRUE) (Default: _1.fastq)

-ro, --readør: Paired end read orientation used for bowtie alignment against te Genome Reference Possible values (f forward, r reverse): fr/rf/ff (Default: fr)

-p, --paired: Paired data? TRUE or FALSE (Default: TRUE)

-s, --slfile: Fasta file with the SL sequences (Default: SL_tags.fa). Multiple variants/SL classes can be analyced toguether by specifing the same ID in the header, and the specific sequence with .(something) (ej: SLXX.1)

Filtering options:

-m, --missing: Maximun expected missing data from the true SL sequence (Crucial if you are using SLFinder results) (Default: 5 pb)

-M, --minread: Minimal read lenght to keep after removing the SL (set to 1 for no trimming) (Default: 25 pb)

Other options:

-o, --outdir: Directory where to store the output and temporary files (Default: SL-analysis)

-t, --threads: Number of processors to use when possible (Bowtie2, Samtols, Seqkit) (Default=1)

3) Script pipeline overview

The first objective of SLFinder genes is to identify, filter and retrieve reads containing the SL tags. This is done first by running "seqkit locate" to identify the reads containing each SL tag and "seqkit fx2tab" to register each read length. This information is combined and reads selected according to: a) Presence of one and only one SL tag hit per ; b) the coordinates of the 3' end of the SL is no farther away than the estimated length of the SL + the "-m" value from the read's closet end; and c) the remaining sequence has to be larger than the "-M" value.

Processing each read one by one can prove to be an inefficient computing process. Fortunately, this is what cutadapt was designed to do. On by one, cutadapt is used to detect, trim and separate reads containing each tag with -g [tag sequence], -a [reverse tag sequence] (and -G plus -A with paired end data), -O [tag length] and -e 0. And the selected reads, or read pairs, retrieved with "seqkit grep".

If no reference genome is provided, the script ends here with the reads stored on "Out_dir/Results/read_SL" so they can be utilized in more appropriate pipelines for your study. If included however, the script then proceeds to make a Bowtie2 index and then mapping the reads to the reference genome with the options "--end-to-end" in the case of single end data, and "--end-to-end --no-mixed --no-discordant" for paired end. Counts per gene are assessed by htseq-counts (--type=exon and --mode=union) and the script verifies the potential acceptor gene that precedes the read mapping location by simply verifying the site is a “AG”. This can take several minutes depending on the data. Note that SLFinder-Genes does not searches for sm sites.

Raw counts are stored on the folder Results/Raw_counts, divided by dataset and SL tag ($DATASET"_"$SL_TAG"_raw_htseq.counts"). Results regarding potential donor sites is stored on the folder Results/Potential_aceptor_sites, detailed per read ($DATASET"_"$SL_TAG"_aceptor_sites.tab", table XX) and a per gene summary ($DATASET"_"$SL_TAG"_aceptor_sites.summary", table XX). Bear in mind, reported raw read counts do not filter out reads without expected acceptor donor sites “AG”.