Computation Biology - Final Report

(Shorty Assembler)

Bala Nagi Reddy Mudiam *(107367424)*

Chandrakantha Reddy B (*107224402*)

Anandsagar Kothapalli *(107232216)*

1. Introduction

Shorty is a short read genome assembler. It is targeted towards paired end micro read sequencing data. It uses seeds and contig distances for sequencing.

2. Project Description

Shorty is a short read de novo assembler which is particularly targeted at the new ABI SOLID sequencing technology. Currently it doesn't have test data to evaluate. So we collected the different test data sets and also different sequencing technology data such as Solexa/Illumina and Helicos Bio Sciences. By running shorty on different data sets we evaluated its performance with respect to other genome assemblers such as velvet.

3. Goal of the Project

Evaluate it on Solexa, ABI Solid data against other assemblers (velvet):

We collected various assembler's data and executed them on shorty and velvet assemblers. We analyzed the results of both the assemblers and drawn advantages and limitations of shorty.

Improve its ease of use for other platforms:

We have collected the data of different assembler's and created required scripts for transforming data from assembly-specific format script.pl <infile> <outfile> paired-end fasta format, the default format for shorty. We have documented this information so that users can use this information to run their assembly data on shorty.

4. Tasks Done

4.1. Running Shorty (fixing configuration issues in the assembler)

Previous version of shorty has lot of configuration issues and it is not executing properly. Running shorty on sample data will give lot of errors. We found out the errors and changed the respective header file declaritives and some configuration issues in the code so that it can be executed on latest C++ compilers. We have successfully executed on GCC 4.2 compiler. We created a README explaining in detail on how to setup, install and run shorty. We have created few scripts and modified some existing scripts to generate contig statistics for the assembly output produced by shorty. By following the new readme document, shorty is working properly on the test data we've collected from Charles. By running according to this and using mummer software we got the outputs in graph format.

To run the shorty assembler, we used the following command.

```
bin/shorty-assembler -o12 -r5 -s3 -l100 -L580077 -t5 shorty-
dat/ab/m_genitalium/<seed file> shorty-dat/ab/m_genitalium/<fasta file>
.mg.0.100
```

All the outputs will be stored in out directory.

We also tried running the assembler end to end. ***run-all*** script in scripts directory allows one to run assembler, geography and collect contig stats by running through Mummer.

The arguments to run-all are explained below.

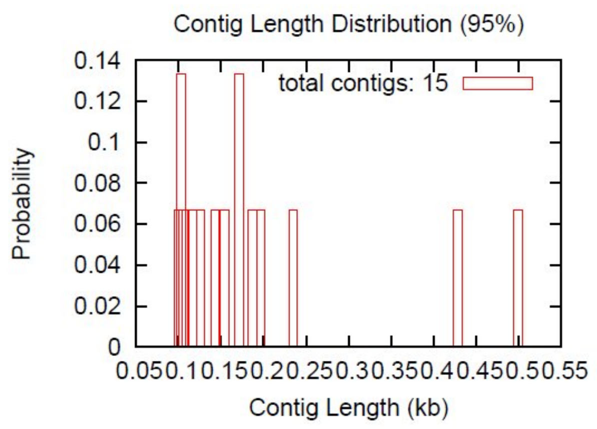
scripts/run-all 12 5 3 100 shorty-dat/ab/m_genitalium/org.seq 5 shorty-dat/ab/m_genitalium/seed.0 shorty-dat/ab/m_genitalium/pair100_2270_350.fasta .mg.0.100

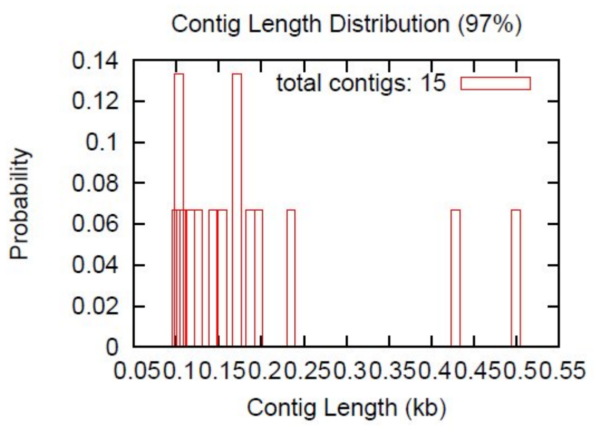
# 1 overlap of reads
# 2 max read reuse
# 3 substition error allowed in seed mapping
# 4 min output contig length
# 5 reference sequence file
# 6 desired thickness of output
# 7 seed file
# 8 reads file
# 9 output extension
(Each one of the above corresponds to the argument specified after run-all)

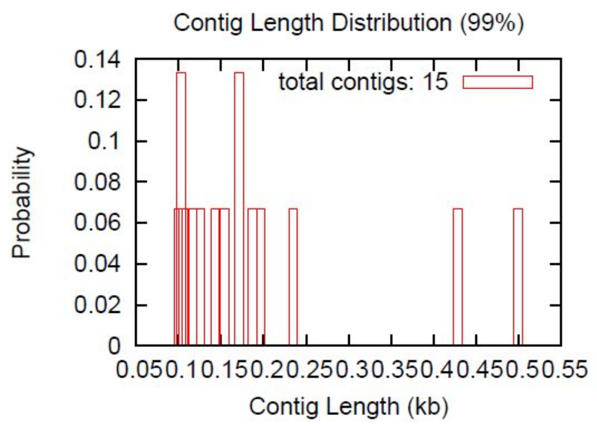
Output:
contigs[ext] : Contigs generated by shorty-assembler
contigs-enriched[ext] : Contigs enriched using shorty-geography
contig-map[ext].eps : Contig length mapping to the reference coverage
contigs-len[ext].eps : Contig length mapping to the accuracy.
contig-map-enriched[ext].eps : Enriched Contig length mapping to the reference coverage(Produced by shorty-geography)
contigs-len-enriched[ext].eps : Contig length mapping to the accuracy.(Produced by shorty-geography)

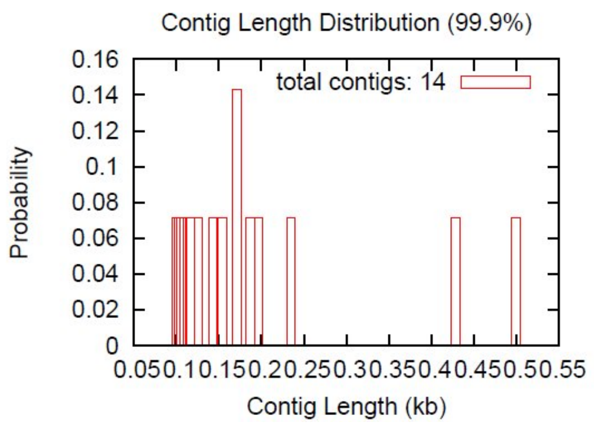
To download the new shorty with all the fixes, click on the below link.
http://www.cs.sunysb.edu/~cbabireddyga/shorty/

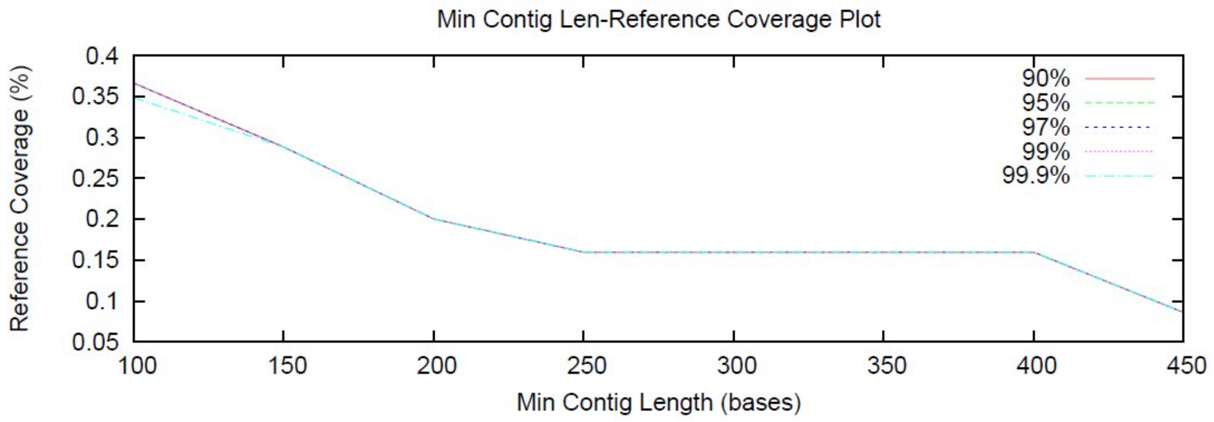
We ran Shorty on Test data after fixing all the issues and the results are shown in following pages.

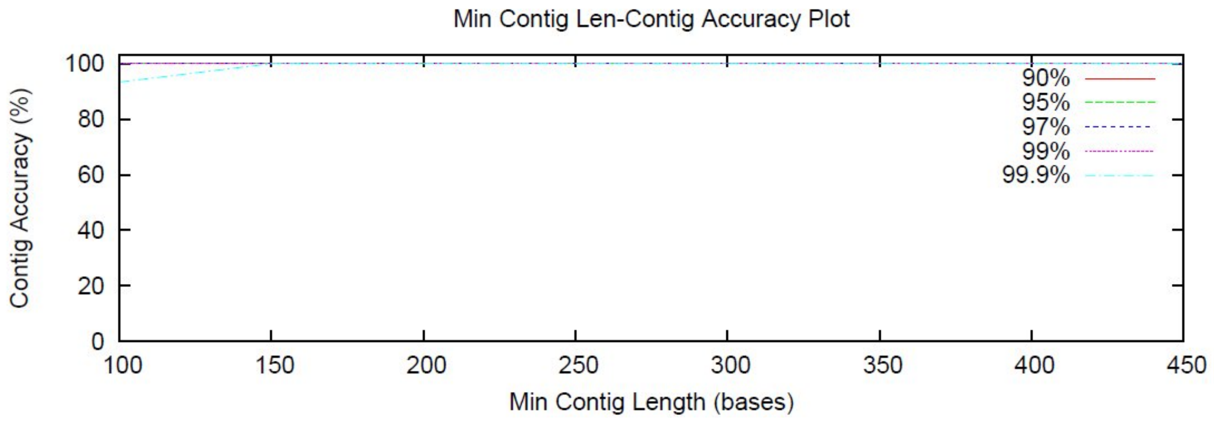

Contig Length Distribution (95%)
total contigs: 15
Probability
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.05 0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45 0.5 0.55
Contig Length (kb)

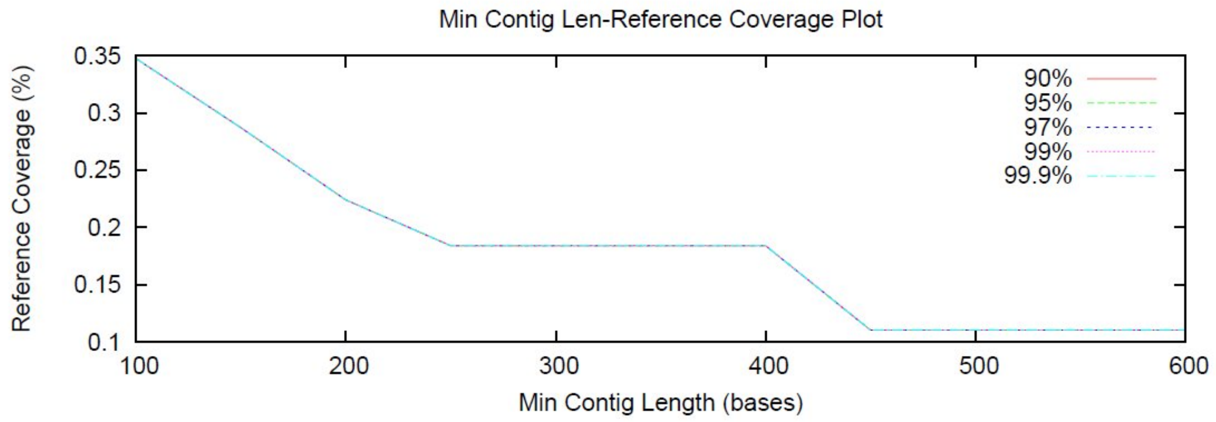

Contig Length Distribution (97%)
total contigs: 15
Probability
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.05 0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45 0.5 0.55
Contig Length (kb)

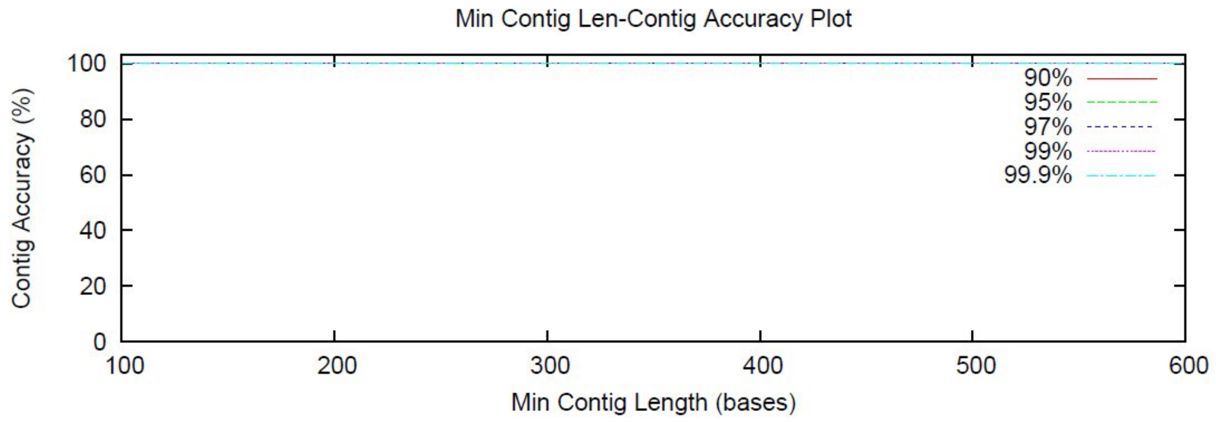

Contig Length Distribution (99%)
total contigs: 15
Probability
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.05 0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45 0.5 0.55
Contig Length (kb)

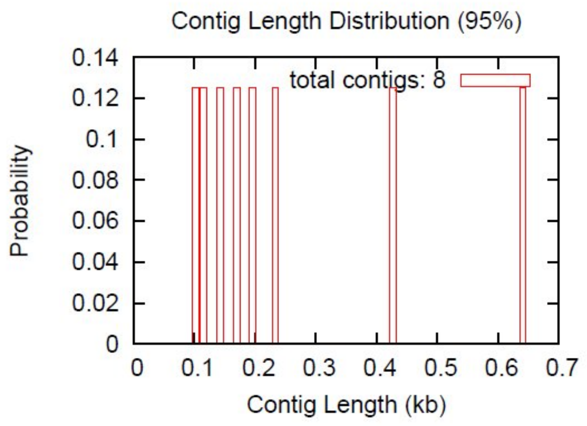

Contig Length Distribution (99.9%)
total contigs: 14
Probability
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.05 0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45 0.5 0.55
Contig Length (kb)


Min Contig Len-Reference Coverage Plot
Reference Coverage (%)
Min Contig Length (bases)
90%
95%
97%
99%
99.9%

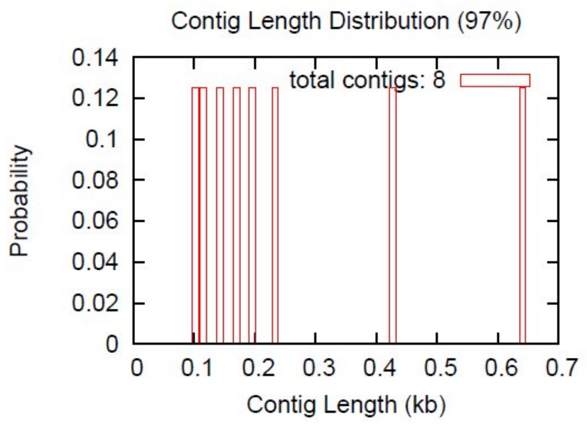

Min Contig Len-Contig Accuracy Plot
Contig Accuracy (%)
Min Contig Length (bases)
90%
95%
97%
99%
99.9%


Min Contig Len-Reference Coverage Plot
Reference Coverage (%)
Min Contig Length (bases)
0.1
0.15
0.2
0.25
0.3
0.35
100
200
300
400
500
600
90%
95%
97%
99%
99.9%

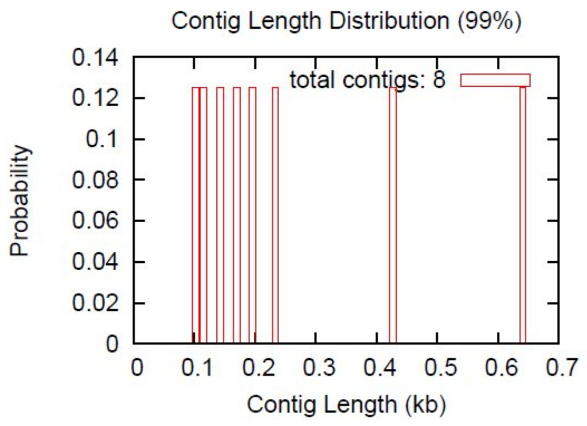

Min Contig Len-Contig Accuracy Plot
Contig Accuracy (%)
Min Contig Length (bases)
0
20
40
60
80
100
100
200
300
400
500
600
90%
95%
97%
99%
99.9%



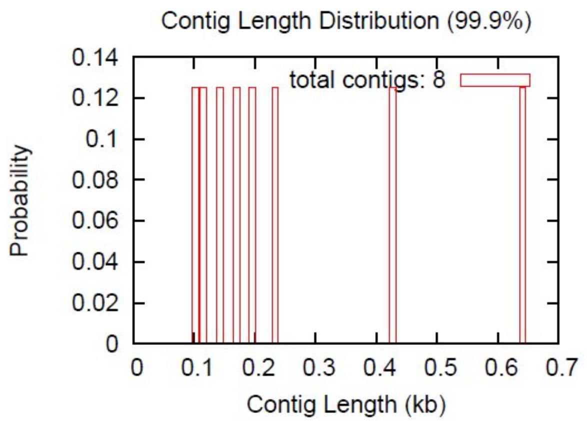
Contig Length Distribution (95%)
total contigs: 8
Probability
Contig Length (kb)



Contig Length Distribution (97%)
total contigs: 8
Probability
Contig Length (kb)



Contig Length Distribution (99%)
total contigs: 8
Probability
Contig Length (kb)



Contig Length Distribution (99.9%)
total contigs: 8
Probability
Contig Length (kb)

4.2. Data Collection - ABI Solid, Solexa Illumina, Helicos

Next task is to collect various assembly reads from the internet, which is a lot difficult than we first thought. Although there are various kinds of sequencing data is available, the data we needed such as solexa/illumina paired end read data, and helicos bio sciences data that is less than 5 GB is rarely available. We were able to find solexa/illumina and ABI solid data and tested them with shorty. The main sources of downloading ABI Solid data which we used are [1],[2],[3],[4],[5]. Solexa/Illumina data is downloaded from [6],[7],[8],[9] Where as Helicos data is downloaded from [10]. Both Solexa and ABI Solid which we used for our project are from ECOLI and M-Genitalium. We search for Helicos data and couldn't find the data which is less than 10 GB. The Helicos data which we got is in FASTQ format by default.

4.3. Conversion between data formats

After collecting data we need to change the different assembly specific formats into FASTA format, since shorty works on fasta format only. ABI solid data is only available in csfasta files. Solexa data available in fastq format only. But to run it on shorty we need to convert it to fasta format. So we found some of the scripts online and modified them to retain the paired information required for shorty. These scripts are used to convert the sequencing data formats. These scripts can be executed using bio perl tools. By using these scripts all the downloaded data is converted to fasta format with paired information required for shorty.

We followed the below procedure to convert paired end Illumina/solexa data to fasta format.

Converting paired end Illumina/Solexa data to FASTA format.

First we do the subsetting. Start by counting the lines, as above:

```
wc -l SRR001666_1.fastq SRR001666_2.fastq
```

Do the subsetting. Soon we will compare the single ended assembly to the paired-end assembly. In order for the comparison to be fair, we must use the same total number of reads. Therefore each paired end file will contain 1/4 of the reads:

```
subset_fastq.pl --input SRR001666_1_sample.fastq --output
SRR001666_1_sample.paired_subset.fastq --length 7047668 --fraction 1

subset_fastq.pl --input SRR001666_2_sample.fastq --output
SRR001666_2_sample.paired_subset.fastq --length 7047668 --fraction 1
```

Shorty expects a single fastq file for paired-end reads, so we have to shuffle (forward, reverse, forward, reverse, etc.) the two files together:

```
shuffleSequences_fastq.pl SRR001666_1_sample.paired_subset.fastq
SRR001666_2_sample.paired_subset.fastq SRR001666_sample.paired_all_subset.fastq
```

Now use the fastq-to-fasta.pl to convert from fastq format of Solexa data to fasta format.

```
./fastq-to-fasta.pl fq2fa SRR001666_sample.paired_all_subset.fastq >
SRR001666_sample_converted.paired_all_subset.fastq
```

The sequence of execution of the above steps would produce the following output.

Sample output before conversion

```
@SRR001666.1 071112_SLXA-EAS1_s_7:5:1:817:345 length=36
GGGTGATGGCCGCTGCCGATGGCGTCAAATCCCACC
+SRR001666.1 071112_SLXA-EAS1_s_7:5:1:817:345 length=36
IIIIIIIIIIIIIIIIIIIIIIIIIIIIII9IG9IC
@SRR001666.2 071112_SLXA-EAS1_s_7:5:1:801:338 length=36
GTTCAGGGATACGACGTTTGTATTTTAAGAATCTGA
+SRR001666.2 071112_SLXA-EAS1_s_7:5:1:801:338 length=36
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII6IBI
@SRR001666.3 071112_SLXA-EAS1_s_7:5:1:839:309 length=36
GAATTTCAATACGGGTGACTTTAATCCCCCACGGGT
+SRR001666.3 071112_SLXA-EAS1_s_7:5:1:839:309 length=36
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII/
```

Sample output after conversion

```
>aSRR001666.1
GGGTGATGGCCGCTGCCGATGGCGTCAAATCCCACC
>bSRR001666.1
AAGTTACCCTTAACAACTTAAGGGTTTTCAAATAGA
>aSRR001666.2
GTTCAGGGATACGACGTTTGTATTTTAAGAATCTGA
>bSRR001666.2
AGCAGAAGTCGATGATAATACGCGTCGTTTTATCAT
>aSRR001666.3
GAATTTCAATACGGGTGACTTTAATCCCCCACGGGT
>bSRR001666.3
TTATCCCGATTCTCATTTTTGTCGCGCTGGTCATTG
>aSRR001666.4
GATACCCAGAATACCAAACACCGTCTGCTGCATATC
```

The data which we collected w.r.t ABI Solid was in FASTA/CSFASTA format. We used FASTA files directly as shorty is compatible with fasta by default. For CSFASTA files, we used conversion scripts[21] to convert into fasta and we ran the same on shorty. As far as Solexa data is concerned, the data which we downloaded was in fastq format by default. We used the above strategy to convert into required fasta format so that it could be made compatible with shorty. Helicos data that we download was in fastq format.

Apart from the above strategy, we also worked on some other scripts for converstion between formats. To view those scripts, please check the link below.
http://www.cs.sunysb.edu/~cbabireddyga/shorty/utilities.html

4.4. Shorty and Velvet – Comparison

Velvet is another short read genome assembler which uses debruijn graphs while sequencing data. We downloaded velvet assembler from the website. In the velvet manual executing various kinds of sequencing data on velvet is explained. We did accordingly and executed it on the test data. But all those results were in .txt format and hardly understandable. So we once again used mummer software for velvet also to generate contig statistics. One advantage with this approach was that Mummer will serve as platform for evaluating shorty and velvet. Now comparing both shorty and velvet will be very easy, we used these results and duly developed a comparison chart.

Velvet by default supports formats such as fasta, fastq, eland, Gerald. Running the data in velvet assembler is basically divided into two steps

```
./velveth output_directory hash_length [[-file_format][-read_type] filename]
```

```
./velvetg output_directory/ -min_contig_lgth 100
```

Velvetg command is the core of the assembler where de Brujin Graph is built, whereas velveth command is used to construct the dataset for velvetg and gives an indication to it on what each sequence file represents.

We downloaded ABI solid data and used our scripts to convert from csfasta to fasta format and ran it against shorty and velvet. We downloaded solexa data from NCBI website and converted from fastq format files to fasta format using our scripts.

To view the results of running velvet and shorty on ABI solid and Solexa sequencing data, Check the link below. From our results it is clear that velvet works better than shorty for this data.

Results are shown here: http://www.cs.sunysb.edu/~cbabireddyga/shorty/graphs.html

Comparison of Shorty with Velvet

Assembler	**N50**	**Max Contig Length**	**Number of contigs**
Shorty-90	550	2550	12122
Shorty with Geography-90	4500	7900	385
Velvet-90	3306	15229	558
Shorty-100	650	2650	11446
Shorty with Geography-100	7350	12350	207
Velvet-100	6721	55874	323

4.6. Performance improvements

We used gprof, open source gnu c profiler for profiling shorty assembler and finding bottlenecks in the code. Bottle necks from gmon output are shown below.

%	self	total	calls	name
11.81	196.60	196.60		myList::checkString(std::stringconst&, int, int)
10.12	364.96	168.36		missMatchNumber(std::string,std::string, int, int)
9.88	529.33	164.36	138559832	fini
7.56	655.20	125.88	3937446119	std::vector<readPointer,std::allocator<readPointer> ::_M_fill_insert()
6.83	768.90	113.70		std::vector<triple<int, int, mar_types>, std::allocator<triple<int, int, mar_types> >>::_M_insert_aux()
5.79	865.33	96.43	1103210846	std::vector<OverlapGraph::ContigInfo, std::allocator<OverlapGraph::ContigInfo> >::_M_insert_aux()
4.71	943.64	78.31	3693798105	ShortyString::ShortyString(std::string const&, int)
3.96	1009.49	65.84	529397792	std::vector<OverlapGraph::ContigInfo, std::allocator<OverlapGraph::ContigInfo>
2.57	1052.33	42.84	1397173708	std::vector<int,std::allocator<int> >::push_back(int const&)
2.08	1086.89	34.57	1198094656	gnu_cxx::new_allocator<triple<int, int, int>>
1.89	1118.40	31.51	1663943608	_M_check_len(unsigned int, char const*) const
1.61	1145.26	26.86		TrieNode::addToTrie(readPointer, int)
1.55	1171.13	25.87	3634312314	UtilsMath::MedianOfIntervals()
1.18	1190.78	19.64		readPointer::stringData()
1.13	1209.57	18.79		getIndex(char)

We have optimized the performance of the above methods by in-lining the functions called from these functions. One such example is returning the base of the corresponding alphabet. Other step where program is lagging is printing the contigs to standard I/O and additional information produced during the assembly like bambus-contig file. We have added debug statements to the code to print these values only when debug option is set to on. This value can be configured in the config file. At first shorty took almost 140 minutes of which assembly took 120 minutes and mummer took 20 minutes to generate the contig statistics. With our improvements, we reduced it to around 80 minutes.

In addition to the above optimizations, there is scope for more optimizations. We also observed that most of the time is spent in the below steps.
1. Writing the contig output and the geography information of the contig.
2. Currently the code takes only one seed; it can be further improved by providing more seeds to the assembler.

5. Documentation - creation of new Read me File

As said earlier we created a new Read Me file. This document is up-to-dated. This document clearly demonstrates how to run shorty on the test data and collected data. This file along with the shorty project with modifications can be downloaded from the website.

6. Future Improvements

We have modified shorty so that it can work on helicos configuration. But we are unavailable to test it on helicos data since all the helicos data samples are single ended reads. But we provided the required steps on how to run shorty on helicos data.

7. Conclusion

As per the goals stated, we have fixed the issues in previous version of shorty and made the working version available at {URL}. we have collected ABI SOLID and Solexa sequencing data and successfully ran them on Shorty. We also evaluated shorty with velvet and analyzed the results. We also improved the performance of shorty from its previous version by significant margin. We created extensive documentation and provided required scripts for future usage of shorty.

8. References according to category

ABI Solid Data:

1. http://www.algorithm.cs.sunysb.edu/charles/
2. http://www3.appliedbiosystems.com/AB_Home/applicationstechnologies/SOLiDSystemSequencing/SoftwareCommunityDataAnalysisResourcesforScientistsDevelopers/index.htm
3. http://solidsoftwaretools.com/gf/?
4. http://www.ncbi.nlm.nih.gov/sites/entrez
5. http://solidsoftwaretools.com/gf/project/

Solexa Data:

6. http://www.ncbi.nlm.nih.gov/sites/entrez?db=sra&term=solexa
7. ftp://ftp.ncbi.nlm.nih.gov/sra/static/SRX003/SRX003935/
8. http://www.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?cmd=show&f=faspftp_seqsamples&m=downloads&s=download_fastq
9. ftp://ftp.ncbi.nlm.nih.gov/sra/SeqSamples/ERS000/ERS000013/ERX000013/

Helicos Data;

10. ftp://ftp.ncbi.nlm.nih.gov/sra/SeqSamples/ERS000/ERS000013/ERX000013/
11. http://open.helicosbio.com/mwiki/index.php/Datasets

General

12. http://www.cs.sunysb.edu/~skiena/shorty/paper1.pdf
13. http://www.cs.sunysb.edu/~skiena/shorty/paper.pdf
14. http://seqanswers.com/forums/index.php
15. ftp://ftp.ncbi.nih.gov/genbank/genomes/Bacteria/
16. http://www.bioperl.org/wiki/Installing_Bioperl_for_Unix
17. http://genome.cshlp.org/content/19/6/1117.full#ref-31
18. http://code.google.com/p/standardized-velvet-assembly-report/
19. http://packages.debian.org/unstable/science/velvet

Conversion scripts

20. http://www.cs.sunysb.edu/~cbabireddyga/shorty/all2y.pl
21. http://www.cs.sunysb.edu/~cbabireddyga/shorty/fastq2fasta.pl
22. http://www.cs.sunysb.edu/~cbabireddyga/shorty/mycsfasta2fasta.pl
23. http://www.cs.sunysb.edu/~cbabireddyga/shorty/script.pl