2014 IEEE 14th International Conference on Bioinformatics and Bioengineering

Effect of Multi-K Contig Merging in *de novo* DNA Assembly

Mohammad Goodarzi
Department of Computer Science
Brock University
St. Catharines, ON Canada

Sheridan Houghten, Member, IEEE
Department of Computer Science
Brock University
St. Catharines, ON Canada

Ping Liang
Department of Biological Sciences
Brock University
St. Catharines, ON Canada

***Abstract*—DNA Assembly is among the most fundamental and challenging problems in bioinformatics. Near optimal solutions are available for bacterial and small genomes. However assembling large and complex genomes including the *human genome* using Next-Generation-Sequencing (NGS) technologies is shown to be very difficult. This paper presents an algorithm for creating contigs from NGS short read data that is capable of working with multiple *k-mer* lengths and introduces a technique to combine contigs generated from different *k* runs with results from other assemblers in order to obtain significantly better assemblies. Experimental results from 9 real datasets show an increase in *N50* value by a factor of 3, when combining newly created contigs with results from other assemblers.**

I. Introduction

DNA assembly is the process of deducing the unique single and contiguous sequence of a DNA molecule by using a set of reads containing shorter sequences from random locations of the genome. There are different general approaches to solve the DNA assembly problem including overlap-layout-consensus (OLC) and de Bruijn graph techniques. Algorithms based on de Bruijn graphs are shown to be more effective and practical for large assemblies such as the human genome [1]. In de Bruijn graph methods, all reads are first processed to find overlapping substrings of length *k*. These substrings are called *k-mers*. All *k-mers* from all reads in the dataset are extracted and each unique *k-mers* is only stored once, while it can be repeated in more than one read. The de Bruijn graph approach is the most used technique for assembling short reads [1]. Using the de Bruijn graph structure to solve the DNA assembly problem is first proposed in [2]. For more information about the de Bruijn graph DNA assembly algorithms, interested readers are referred to [1], [2], [3].

Three different assemblers including Velvet [4], Meraculous [5] and SOAPdenovo [6] are used for comparisons in this paper. SOAPdenovo and Velvet are among the most used DNA sequence assemblers and Meraculous is selected because of our inspiration from their idea on how to create contigs from short reads. Velvet introduces a set of algorithms to manipulate de Bruijn graphs and to eliminate and resolve the repeating patterns in the genome. In Velvet, first all sequences that belong to one region are detected and then a repeat solver algorithm separates paths sharing local overlaps [4]. Meraculous works by using a conservative traversal of a subgraph of the de Bruijn graph. The Meraculous contig creation algorithm only considers unique high quality extensions in the dataset and does not rely on any other error correction techniques other than the base quality scores [5]. Based on Meraculous experiments, we decided to use their technique to create contigs from the short reads and then implement our own contig merging algorithm on top of it. This is because we believe that while Meraculous contigs may be smaller in size, they are more accurate because of using a more conservative algorithm that rejects false-positive links between the *k-mers*. SOAPdenovo is another short read assembler that is based on de Bruijn graphs and is shown to be capable of solving datasets in the size of the human genome. SOAPdenovo uses similar de Bruijn graph data structures to Velvet, but handles read locations and paired-read information in a different way which is further discussed in [6]. SOAPdenovo is shown to be an aggressive assembler that creates more indels as it tries to obtain the maximum size for contigs [7].

The *k* parameter has significant influence on assembly results and due to reasons such as uneven data coverage, noisy data and varying repeat structures in different genome locations, a single value of *k* does not necessarily give the optimal result for all locations in the genome. Having a very large value for *k* may fail to detect overlaps between the reads, while a small *k* value may result in tangled assembly graphs that are impractical to solve [8].

The DNA assembly problem is usually solved by having heuristics in mind. These include de Bruijn graph simplifications or greedy-based techniques that decide on the correctness of assembly graph edges heuristically. Different heuristics result in fragmented assemblies from different locations of the genome. By applying different heuristics and simplification methods, various assemblies are generated for one genome and the problem becomes worse when it is infeasible to accurately select the best result. This is mainly because in *de novo* DNA assembly, there is no reference genome available to measure the correctness of different assemblies. Results with higher length-based metric values such as the *N50* parameter are currently considered as better assemblies because they are producing larger fragments from the genome. The *N50* value is a statistical measure of a set of numbers in which all elements of greater than or equal to the *N50* value cover at least half of the total sum [1]. However, there are experimental results [1], [9] showing that larger contigs do not necessarily mean improved results and *N50* values can be misleading when assemblies have many false-positive fragments. For instance, a new technique for evaluating genome assemblers [7] first splits the contigs/scaffolds on locations for which the left and right pieces map onto different locations in the reference genome

978-1-4799-7502-0/14 $31.00 © 2014 IEEE
DOI 10.1109/BIBE.2014.49

(a well characterized genome that can be used for evaluation purposes) and then calculate the *N50* value based on the split contigs. This technique is believed to obtain more accurate calculations by skipping false-positive links in assemblies.

This paper presents a contig merging algorithm that analyzes the contigs generated by different assemblers and identifies the overlapping parts to merge the results and obtain larger contigs. We also developed our own DNA assembler based on the Meraculous assembler [5] with modifications that enable us to perform the assembly with multiple *k* values. The remainder of this paper is organized as follows: Section II discusses the related works in the field. Section III includes the multi *k-mer* assembly idea and Section IV presents the contig merging algorithm. Experimental results are shown in Section V and conclusions and future work are discussed in Section VI.

All algorithms presented in this paper are available to download at http://genomics.brocku.ca/dna_assembly/.

II. Related Work

The idea of finding the optimal value for *k* or using multiple *k* parameters has been investigated before. VelvetOptimizer [10] performs the Velvet assembly algorithm multiple times with the aim of finding the best *k* value. VelvetOptimizer decides on the best *k* value based on the *N50* value of assemblies and is an exhaustive approach to find the most appropriate *k* which is not practical in case of large datasets. Moreover, it relies on the *N50* parameter to assess the quality of the assemblies which is not always accurate [7], [11].

IDBA assembler [12] solves the DNA assembly problem by defining a valid range for the *k* value and keeps all information for all *k* values in the graph, thus benefiting from both small and large values. Their research is the closest work to our idea. However, we think that while considering different *k* values brings many advantages, importing the results from other tools and combine them with the internal contigs can also boost the results.

III. Multi *K-MER* Assembly

Our DNA assembly tool is inspired by the Meraculous assembler [5]. We selected the Meraculous assembler as the base idea for our implementation because of the level of accuracy that it provides [5]. We think that that the accuracy of the contigs is the most important factor for our contig merging algorithm to succeed. However, our implementation differs from Meraculous in several ways:

- Our implementation is capable of running multiple assemblies with different *k* values. Our tool can be set to run different *k* assembly runs either sequentially or in parallel. The sequential run demands less memory and is useful when there are memory limitations but it takes longer to perform. Conversely, the parallel run benefits from .NET framework threading techniques and uses all CPU cores to perform the algorithm more quickly, while it may require a considerable amount of memory depending on the data size. In our tool's parallel run scenario: one global set of reads is first created and is used by all k-assembly runs. Computations for each k-assembly run are set to be performed in separate threads (likely to run on separate CPU cores managed by .NET framework) that significantly improves algorithm's time. Our tool's time and memory requirements are further discussed in Section V-C.
- Our implementation offers a more compact solution to work with NGS reads. Instead of using one byte (8 bits) for each base in the reads, we only use 2 bits to represent each base (A:00, C: 01, G:10, T:11). In doing so, each byte represents 4 bases leading to a 75% decrease in the amount of memory required to load the input NGS data. It should be noted that by using the proposed bit-structure our tool is not able to handle DNA sequences with undetermined bases (N bases).
- As our tool works at the level of bits, we implemented a new set of data structures and different bitwise hashing techniques compared to [5].

Many current assembly algorithms consider a fixed value for *k* while this parameter has a significant role in obtaining the best results. There are methods to analyze the input data and find the most appropriate *k* value for the given input [13], [14]. However, to the best of our knowledge, many of the proposed methods assume an even coverage through the input data and calculate a single *k* value for the data set; this is not always correct especially for human genome data because of its size and heterogeneity in repeating patterns. Moreover, repeating patterns in the genome have different characteristics and they play the most important role in the quality of assembly results. Different *k* values result in either resolving repeat structures, or being stuck in the middle of the contig creation process, and there is not any unique *k* value that can work for all locations of the genome. Small *k* values make the de Bruijn graph very tangled and messy, thus the paths are not fully detectable and the quality of results decreases. On the other hand, large *k* values may resolve repeat patterns with length of less than *k* but may fail to detect overlaps between the reads/*k-mers*, particularly in low coverage regions, making the graph more fragmented [8].

There have been attempts in assemblers such as [4] to run the algorithms for multiple *ks* but the assemblers themselves do not try to further improve the overall results based on the outputs from multiple *k* runs.

By running the algorithm for different *k* values, it is more likely that the best contigs from all locations of the genome are created but in different runs. While most of the contigs from different runs express on the same locations, there are new regions in each assembly as well. This makes it feasible to obtain larger contigs by analyzing the results from different runs and trying to merge the overlapping parts.

IV. Contig Merging Algorithm

Contigs are contiguous portions of the genome that the assembler successfully constructs. Because there is not any information regarding the strand to which the base reads belong, contigs are created on both strands which brings two versions of each contig (the contig itself and its reverse-complement) to the contig set. However, contigs do not have

```
1 : p <= L1 - 1
2 : while p >= CONTIGS_MIN_OVERLAP do
3 :    match <= true
4 :    for i = 0 to p - 1 do
5 :       if cntg1[L1 - p + i] <> cntg2[i] then
6 :          match <= false
7 :          break
8 :       end if
9 :    end for
10:   if match then
11:      return cntg1 + cntg2.substr(p)
12:   end if
13:   p <= p - 1
14: end while
15: return null
```

Fig. 1. Contigs left link check algorithm

```
1 : p <= 0
2 : while p + L1 >= L2 do
3 :    match <= false
4 :    for i = 0 to L1 - 1 do
5 :       if cntg1[i] <> cntg2[i + p] then
6 :          match <= true
7 :          break
8 :       end if
9 :    end for
10:   if match then
11:      return cntg2
12:   end if
13:   p <= p + 1
14: end while
16: return null
```

Fig. 2. Contigs substring check algorithm

any overlap of length more than k with each other, because if they did this overlap would be detected in the previous steps of the assembly algorithm, unless they come from different k runs. Therefore the attempt to merge contigs all generated from one fixed k value does not improve the results. However, the merging idea works when the contigs are generated from different k assemblies.

The first step to merge contigs is to find overlaps between them. As contigs can belong to either of the genome strands, reverse-complements are generated for all of them in the first step. The reverse-complement contigs double the size of the dataset but we can ensure finding overlaps between the contigs that construct the same location in the genome but from different strands. In order to find extensions for the contigs, an algorithm is needed to check if there is any overlap between any two inputs. There are three situations in which two contigs can be linked together:

1) The first contig's ending bases are matched with the second contig's starting bases, thus the first contig can be linked to the second contig from the left. The algorithm to check this condition is presented in Figure 1.
2) The first contig is completely repeated in the second contig, thus the second contig expresses the merging result. The algorithm to check this condition is presented in Figure 2.
3) The first contig's starting bases are matched with the second contig's ending bases, thus the first contig can be linked to the second contig from the right. The algorithm to check this condition is presented in Figure 3.

```
1 : p <= L1 - 1
2 : while p >= CONTIGS_MIN_OVERLAP do
3 :    match <- true
4 :    for i = 0 to p - 1 do
5 :       if cntg1[i] <> cntg2[L1 - p + i] then
6 :          match <= false
7 :          break
8 :       end if
9 :    end for
10:   if match then
11:      return cntg2 + cnt1.substr(p)
12:   end if
13:   p <= p - 1
14: end while
15: return null
```

Fig. 3. Contigs right link check algorithm

```
1 : Contig cntg1;//cntg 1 is always the smaller contig
2 : Contig cntg2;
3 : L1 <= length(cntg1)
4 : L2 <= length(cntg2)
5 : Consensus <= RightLinkCheck(cntg1, cntg2)
6 : if consensus <> null then
7 :    return consensus
8 : end if
9 : consensus <= LeftLinkCheck(cntg1, cntg2)
10: if consensus <> null then
11:    return consensus
12: end if
13: consensus <= SubStringCheck(cntg1, cntg2)
14: if consensus <> null then
15:    return consensus
16: end if
17: return null
```

Fig. 4. Finding contigs overlap algorithm

In all of the algorithms introduced in Figures 1, 2 and 3, variable `p` defines the overlap length between the two contigs and a loop is used to check for equality between the two contigs with regards to the `p` value. For the left-link-check and right-link-check algorithms (Figure 1 and Figure 3) the procedure starts with the maximum possible value for `p` and decreases it until finding a match between the two contigs. For the contig-substring-check algorithm in Figure 2, the procedure checks to see if the first contig is entirely seen in the other contig or not.

Figure 4 shows the procedure for finding the overlap between two input contigs (consensus sequence). It calls other procedures presented in Figure 1, Figure 2 and Figure 3 to check for all conditions in which two contigs can generate a consensus sequence. The minimum overlap length between the contigs can be set in the assembly's configuration file and usually equals to the minimum k value considered.

Being able to merge any two input contigs, an iterative procedure can be devised to merge and extend contigs until no more extension is found. This procedure is shown in Figure 5.

A. Importing External Contigs

While merging the results from different runs of our own assembly algorithm helps creating better contigs, importing and mixing the contigs from other tools can also be very beneficial. Each assembler has a unique way of creating contigs with various heuristics and assumptions involved, which leads to different results for one input dataset. We argue that while most parts of the genome may be created correctly by all assemblers, there are some regions that are only built by

```
1 : while contigs > 1 do
2 :    baseContig <= contigs[0]
3 :    remove baseContig from contigs
4 :    overlapFound <= false
5 :    List<Contig> newlyAddedContigs
6 :    for all cntg in contigs do
7 :       consensus <= ContigsOverlaped(baseContig, cntg)
8 :       if consensus <> null then
9 :          remove cntg from contigs
10:          add consensus to newlyAddedContigs
11:          overlapFound <= true
12:          if consensus == cntg then
13:             break
14:          end if
15:       end if
16:    end for
17:    add newlyAddedContigs to contigs
18:    if overlapFound == false then
19:       add baseContig to finalContigs
20:    end if
21: end while
22: return finalContigs
```

Fig. 5. Contigs expansion algorithm

some techniques and left over by others. We believe that building contigs with different initial k values and merging them with results from other tools should lead to better assembly results. Experimental results in Section V show the improvements obtained from merging external contigs to the outputs generated from our own algorithm.

V. Experimental Results

All of the datasets considered in this paper are from NCBI SRA database for human sample ID, NA12878. These sequences were generated using Illumina HiSeq 2000 with pair-end sequencing at $100bp \times 2$. We created 9 datasets with different genome lengths for our experiments in this paper that are presented in Table I. All assemblers ran to the point at which the contigs are produced. The rest of the assembly process (including contigs orientation, scaffolding using mate-pairs and etc.) are skipped as our focus in this paper is sorely on the contigs.

We use the human reference genome (hg19) to estimate the accuracy of the contigs and detect the false links between the final contigs. In order to detect the false links, we split each contig from all locations for which the left and right fragments are aligned to distant locations in the reference genome, meaning the contigs are not built in a correct way and should split. In other words, we consider alignment blocks from the BLAT [15] tool as the correctly mapped fragments and the maximum allowable gap between the alignment blocks is set to 50 bases. Among all possible alignments for each contig, the alignments that are not in the selected region are filtered out first and then the best scoring alignment is selected for each contig.

From now on, when we refer to the *N50* value, we mean the calculated value based on the fragments generated by splitting contigs in described locations, and not the base contigs which are the actual outputs of the assemblers. Note that this approach focuses more on true positive results (which is our focus in the merging algorithm), while it may neglect the effect of false positives in the final results of algorithms (including our own). It should also be noted that in order to compare different assemblies based on the *N50* statistics, we use a modified version of the *N50* in which the genome length is used as the reference total sum.

TABLE I. Experimental Datasets

Dataset	Genome Length	Location	Chrom.	# Reads
1	1Kb	100k-101k	1	190
2	10Kb	100k-110k	1	3452
3	10Kb	60k-70k	10	1296
4	100Kb	100k-200k	1	19246
5	100Kb	60k-70k	10	17178
6	1Mb	100k-1100k	1	190030
7	1Mb	60k-1060k	10	182370
8	10Mb	100k-10100k	1	1766556
9	10Mb	60k-10060k	10	1825054

There are two main criteria for the experimental results:

- ***N50* comparisons:** Measuring the quality of results based on contigs' length. Before calculating *N50* values, contigs split into several fragments as described in section V.
- **External contigs expansion results:** Investigating the quality of results when external contigs are added to our generated contigs and the contig merging algorithm is performed on the dataset.

Three assemblers including Meraculous [5], SOAPdenovo [6] and Velvet [4] were selected to run on the given datasets. Our tool is capable of running the assembly process for multiple k values in parallel with any k value set provided. Other tools either do not have this feature or have it implemented in a way that cannot accept all k combinations in one run; therefore we ran each assembler for each k value individually and obtain the average between the runs. The k values that are used in our experiments are fixed for all datasets, and cover a range of small and large values. These values are k = 19, 31 and 41. While it is not guaranteed that the selected k values produce the best assembly results, we argue that values of smaller than 19 and larger than 41 are not reliable enough to do our experiments. However, we have left the investigation on finding the most appropriate k values for each dataset for the future works.

A. N50 Results

Figure 6 demonstrates the *N50* comparisons between our tool and the Meraculous assembler. We perform this comparison because of the fairly similar contig creation algorithm that two assemblers have. Thus we can clearly demonstrate the efficiency of our own specific modifications. Results show that our tool has better performance in all of the experimented datasets. The most important factor for the better performance is using multiple k values in the assembly process. Different k values produce reasonably large contigs from different locations in the genome and merging the results from various k runs considerably boosts the performance. This helps obtaining significantly better results in some cases.

In two of the datasets the Meraculous assembler has an *N50* value of zero which means the total length of all fragments is not more than half of the targeted genome's length, while our tool obtains *N50* values of 86 and 573 respectively. Considering only the remaining 7 datasets, our tool creates contigs that are 6.15 times larger than those produced by Meraculous. It is worth noting that all of the input short reads

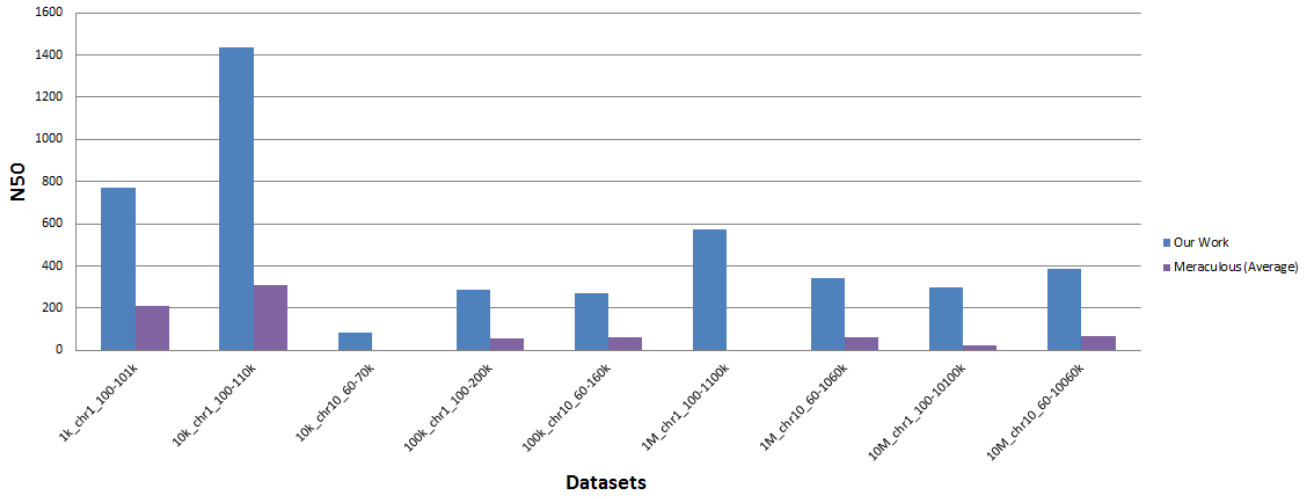

Fig. 6. *N50* results. Comparing our tool's performance to Meraculous on 9 datasets

are 100 bps in length but the minimum acceptable length for contigs is set to the minimum k value. Therefore, in some of the datasets we obtain *N50* values of less than 100.

B. External Contigs Merging Results

This section presents results for performing the contig merging algorithm described in Figure 5 when external contigs from other tools are imported into the system. For each assembler, the best run with the highest *N50* value is selected.

Figure 7 (top chart) shows experimental results for integration of our tool with the Velvet assembler. Results show that combining our contigs to contigs generated by Velvet significantly increases the quality of results leading to larger fragments from the genome and better *N50* values. All datasets show improvements in results and the *N50* value is increased by on average a factor of 3.2.

Figure 7 (middle chart) presents results for integration of our tool with the SOAPdenovo assembler. Results show that combining our tool's contigs with contigs generated by the SOAPdenovo assembler also generates better results, having larger fragments and *N50* values. The *N50* value is increased by on average a factor of 3.06.

Figure 7 (bottom chart) presents results for integration of our tool with the Meraculous assembler. Results show significant improvements in *N50* values by merging our tool's contigs with contigs generated by the Meraculous assembler. The *N50* value is increased by on average a factor of 3.5. In two datasets Meraculous has an *N50* value of zero that are excluded when calculating the average.

Results obtained in this section support the idea that it is possible to obtain improved assembly results by merging external contigs to our contigs that are created with multiple k values. This in fact shows that some of contigs generated by our tool are from the locations that are left over by other assemblers, therefore overlaps can be found between the results. However, there are also false positive links between the merged contigs, thus creating incorrect fragments in the results. Currently, we avoid the influence of the false contigs in our results by splitting the contigs from different locations using the BLAT alignments on the human reference genome.

C. Time and Memory Requirements

1) DNA Assembly Algorithm: Our implementation for the DNA assembly algorithm uses considerably less amount of memory compared to other assemblers as it only uses 2 bits to represent each base in NGS reads. This effectively reduces the memory requirements of the algorithm. Once the *k-mers* are built during the assembly process, the tool does not need to acquire any additional memory to perform the algorithms, meaning that the memory requirement is dependent on the number of *k-mers* which increases with the size of the input data. The largest dataset experimented in this paper covers a region of 10 million base pairs in human genome and has about 2 million short reads. On this largest dataset, the algorithm takes about 1.3 GBs to complete the process for three k values in batch sequential mode and about 2.6 GBs to complete the process in parallel mode. As the memory requirement grows linearly with the number of *k-mers* (and consequently reads), we expect the tool to be able to perform assembly runs for datasets of about 10 million reads on currently available personal computers (less than 12GBs of memory).

In terms of time requirements, the largest dataset in our experiments takes less than 30 minutes to complete, and we expect the time requirements for larger datasets to increase linearly to the dataset's size (number of reads). Because our tool only supports the contig creation phase of the assembly process and does not implement the rest of algorithms in assembly pipeline (including scaffolding) yet, we are not able to directly compare our time and memory requirements to other assemblers. However we believe that our algorithm's time and memory complexity is comparable and in the same level to most of other assemblers. However, supporting this claim in a more concrete and scientific way is left for our future work when our assembly pipeline is completed.

2) Contig Merging Algorithm: The contig merging algorithm accepts a set of generated contigs from different assembly tools. As the number of contigs are very small compared

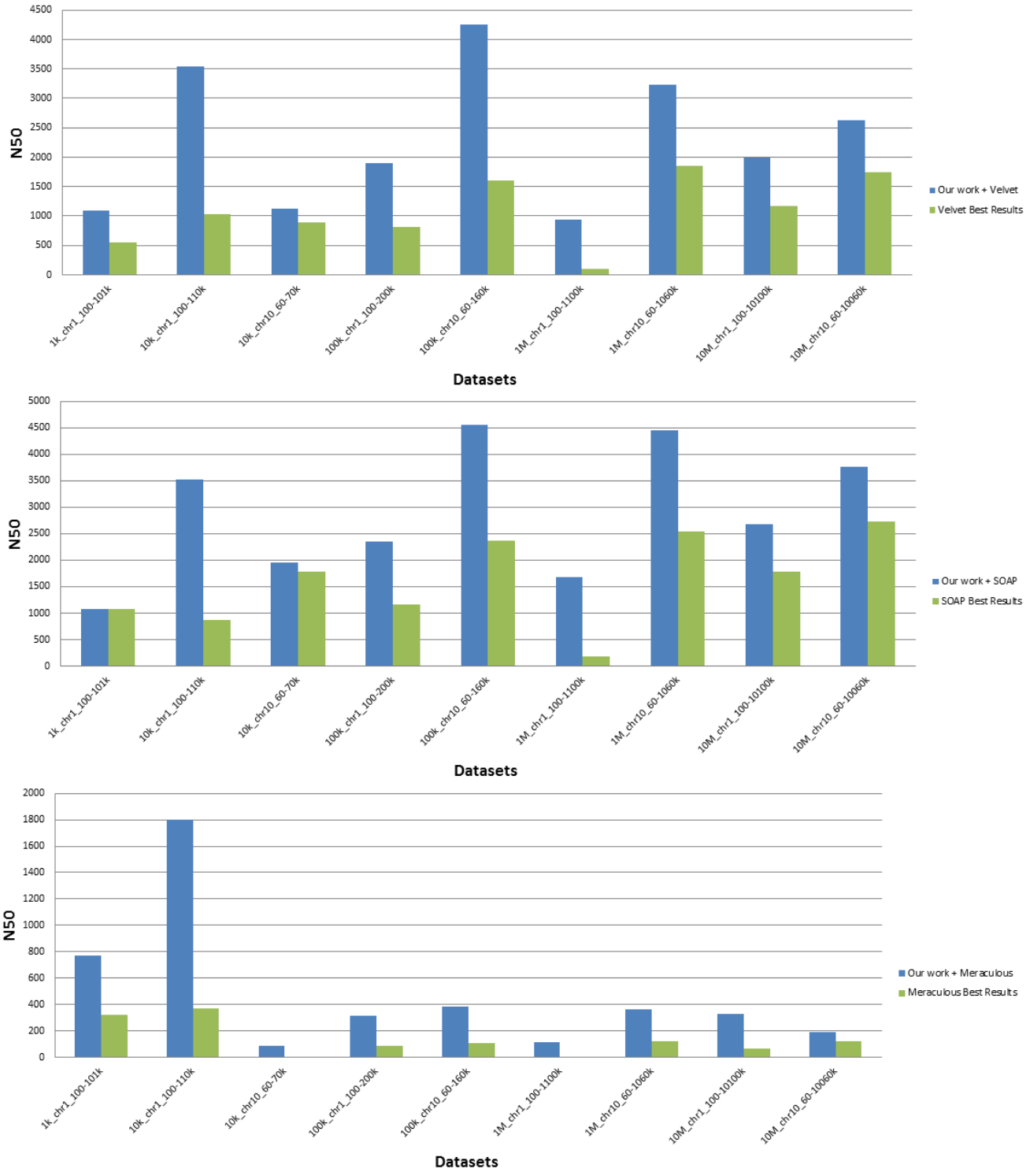

Fig. 7. Improvements made to other assemblers results with Contig Merging algorithm. Top: Merging our tool contigs to Velvet results increased the *N50* value by average factor of 3.2. Middle: Merging our tool contigs to SOAPdenovo results increased the *N50* value by average factor of 3.06. Bottom: Merging our tool contigs to Meraculous results increased the *N50* value by average factor of 3.5.

to the number of reads in NGS data, the algorithm does not demand much memory to perform and it barely requires more than 2 GBs of memory even for very large inputs. However, the algorithm has an exponential time complexity as it compares each and every contig with other contigs in the assembly runs and tries to find overlaps between them. For small inputs, the algorithm completes in only a couple of minutes. However, for large datasets containing more than 4000 contigs in which the longest contigs are more than 80Kb in length, the algorithm takes about 10 hours to complete.

VI. Conclusion and Future Work

The *de novo* DNA assembly problem is still an open problem to solve, specifically for large genomes, such as the human genome, which have a variable but a high level of repetitive sequences. This paper focuses on creating contigs from short reads generated by Next-Generation-Sequencing technologies and merging other assemblers' contigs with those generated by our tool in order to obtain improved results. Our contig creation algorithm is capable of running the assembly process with several k values in parallel and merging the results from different runs at the end. Experimental results show considerable improvements in results when using multiple k values and importing external contigs to the assembly process.

Usually the heuristics that are breaking the contigs in different assemblers are to ensure the correctness and quality of the results. Thus, merging them back together without much consideration may cause false positive overlaps and decreases the quality of the results. The most important future direction for our research is to investigate the exact false positive rate when joining contigs from different k assemblies and different assemblers and try to only select the true positive results among all generated overlaps.

Measuring the quality of the assemblies in the *de novo* DNA assembly problem is challenging as there is no reference genome. We would like to investigate using other quality measurement techniques instead of the *N50* value to better decide on the quality of the contigs.

The *de novo* DNA assembly problem is a large problem consisting of several parts. Pruning input datasets in order to remove noisy parts, creating contigs based on the short reads, orienting contigs by using mate-pair information and creating scaffolds based on contigs are all different stages of a DNA assembly process. This paper focuses only on creating contigs from short reads, while completing other stages of the DNA assembly is in our future work. For now we have avoided using the wrong contigs to influence the *N50* value in our experiments by discarding the contigs with their left and right alignments mapping to different locations in the human reference genome.

We would also like to investigate using pair-read information during the contig creation algorithm. By having the estimated distance between the pair-reads in the genome, we want to investigate new ways to create contigs that have more correct links and fewer false links, leading to more accurate results.

Other future directions for our work include automatically finding the minimum overlap length parameter in the contig merging algorithm and extending our assembly algorithm to do scaffolding.

Acknowledgment

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada.

References

[1] J. R. Miller, S. Koren, and G. Sutton, "Assembly algorithms for next-generation sequencing data," *Genomics*, vol. 95, no. 6, pp. 315–327, 2010.

[2] P. A. Pevzner, H. Tang, and M. S. Waterman, "An eulerian path approach to dna fragment assembly," *Proceedings of the National Academy of Sciences*, vol. 98, no. 17, pp. 9748–9753, 2001.

[3] M. Pop, "Genome assembly reborn: recent computational challenges," *Briefings in bioinformatics*, vol. 10, no. 4, pp. 354–366, 2009.

[4] D. R. Zerbino and E. Birney, "Velvet: algorithms for de novo short read assembly using de bruijn graphs," *Genome research*, vol. 18, no. 5, pp. 821–829, 2008.

[5] J. A. Chapman, I. Ho, S. Sunkara, S. Luo, G. P. Schroth, and D. S. Rokhsar, "Meraculous: de novo genome assembly with short paired-end reads," *PloS one*, vol. 6, no. 8, p. e23501, 2011.

[6] R. Li, H. Zhu, J. Ruan, W. Qian, X. Fang, Z. Shi, Y. Li, S. Li, G. Shan, K. Kristiansen *et al.*, "De novo assembly of human genomes with massively parallel short read sequencing," *Genome research*, vol. 20, no. 2, pp. 265–272, 2010.

[7] S. L. Salzberg, A. M. Phillippy, A. Zimin, D. Puiu, T. Magoc, S. Koren, T. J. Treangen, M. C. Schatz, A. L. Delcher, M. Roberts *et al.*, "Gage: A critical evaluation of genome assemblies and assembly algorithms," *Genome research*, vol. 22, no. 3, pp. 557–567, 2012.

[8] A. Bankevich, S. Nurk, D. Antipov, A. A. Gurevich, M. Dvorkin, A. S. Kulikov, V. M. Lesin, S. I. Nikolenko, S. Pham, A. D. Prjibelski *et al.*, "Spades: a new genome assembly algorithm and its applications to single-cell sequencing," *Journal of Computational Biology*, vol. 19, no. 5, pp. 455–477, 2012.

[9] K. R. Bradnam, J. N. Fass, A. Alexandrov, P. Baranay, M. Bechner, I. Birol, S. Boisvert, J. A. Chapman, G. Chapuis, R. Chikhi *et al.*, "Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species," *GigaScience*, vol. 2, no. 1, pp. 1–31, 2013.

[10] (2014) Velvetoptimizer. [Online]. Available: http://bioinformatics.net.au/software.velvetoptimiser.shtml

[11] G. Narzisi and B. Mishra, "Comparing de novo genome assembly: the long and short of it," *PloS one*, vol. 6, no. 4, p. e19175, 2011.

[12] Y. Peng, H. C. Leung, S.-M. Yiu, and F. Y. Chin, "Idba–a practical iterative de bruijn graph de novo assembler," in *Research in Computational Molecular Biology*. Springer, 2010, pp. 426–440.

[13] J. T. Simpson, K. Wong, S. D. Jackman, J. E. Schein, S. J. Jones, and I. Birol, "Abyss: a parallel assembler for short read sequence data," *Genome research*, vol. 19, no. 6, pp. 1117–1123, 2009.

[14] J. Butler, I. MacCallum, M. Kleber, I. A. Shlyakhter, M. K. Belmonte, E. S. Lander, C. Nusbaum, and D. B. Jaffe, "Allpaths: de novo assembly of whole-genome shotgun microreads," *Genome research*, vol. 18, no. 5, pp. 810–820, 2008.

[15] W. J. Kent, "Blatthe blast-like alignment tool," *Genome research*, vol. 12, no. 4, pp. 656–664, 2002.