LPS: a strategy for the generation of longer DNA sequence fragments from short reads

Francisco Vera Voronisky[1], Ansel Y. Rodríguez-González[1], Ivan Olmos-Pineda[2], Patricia Sanchez-Alonso[3], Candelario-Vazquez-Cruz[4], and Jesus A. Gonzalez[1]

[1] Coordinación de Ciencias Computacionales, INAOE, Puebla, México
{fvera,ansel,jagonzalez}@ccc.inaoep.mx
[2] Benemérita Universidad Autónoma de Puebla
iolmos@cs.buap.mx
[3] Centro de investigaciones en Ciencias Microbiológicas, ICUAP, BUAP
maria.sanchez@correo.buap.mx
[4] Centro de investigaciones en Ciencias Microbiológicas, ICUAP, BUAP
ecobacilos@yahoo.com

Abstract. The output of some modern genome sequencing techniques consists of short length DNA fragments known as reads. A disadvantage of short length reads is that they may appear at different positions of the original genome sequence. Not recognizing the position of repetitive fragments may generate gaps in the final assembled sequence. This happens because repeated fragments would not be considered as candidates to appear at different positions of the assembly. Generating longer length reads could reduce the number of possible repeats in the genome, but this option is not always feasible due to technical restrictions. In this paper we propose a preprocessing method, called LPS, that takes short reads as input to construct longer DNA fragments. LPS uses a clustering strategy based in the overlaps between sequences to reduce the number of fragments to be fed to the assembly tool and to obtain enhanced quality assemblies.

Keywords: DNA assembling, clustering, bioinformatics, machine learning

1 Introduction

Assembly of a genome sequence consists of making a digital copy of the DNA of an organism into a string composed of characters representing nucleotide residues. In this process, the DNA of the organism is isolated, read by a sequencer, and converted into digital information that can be processed by a computer. Because sequencing technology can only read a few base pairs of DNA (in the range of hundreds of base pairs) [1], the genome sequences obtained are small, independent fragments called *reads*. These fragments are then used to reconstruct the genome sequence within a process called assembly.

Sequences assembly is the process of merging fragments of DNA (subsequences or substrings) to reconstruct the original DNA sequence or genome string. In the process

of obtaining the DNA sequence, the DNA molecule is randomly cut in different positions to get a set of fragments that can be sequenced. The overlaps between fragments are then used to construct the best string representing the DNA original molecule. This strategy is known as the shotgun sequencing.

Currently, shotgun sequencing is used in a new generation of methods called 'next-generation sequencing'. Although these technologies produce short sequences (between 25 and 500 base pairs), it is possible to obtain a large amount of reads in a short time, covering several times the whole sequence of the genome [2]. Due to the high speed and low cost of the process, it can be repeated several times over the same sequence, which increases the accuracy of the assembly by obtaining overlapped data reads.

Notwithstanding the high computational complexity of the assembly process, repeated elements, either from tracks of low complexity repeats or selfish elements can be found and they may appear at different positions of the genome. If these repeats are not identified, gaps might be introduced in the reconstructed genome sequence. With the aim of identifying repeated fragments it is possible to follow any of two strategies:

- Obtain longer length reads.
- Obtain reads with context information.

The first option consists on generating longer length reads in order to span in a read most of the possible information of the segment harboring the repetitive element that does appear at different locations of the genome. Obtaining longer length sequences decreases the probability of ambiguities derived from the virtual overlapping process of the reads, and from sequence orientation of the repeated element found a different positions in the genome. As mentioned above, methods called 'next-generation sequencing' produce shorter length sequences [2], making this strategy unviable.

In the second option we obtain reads together with context information, instead of getting each read independently of the others. The fragments obtained with this technique consist of pairs of reads separated by a distance of approximately d characters [3-5]. Obtaining pairs of sequences reduces the probability that these pairs belong to a repeat as the distance between them is larger. These methods require a heuristic to reconstruct a path between a pair of sequences. This path may contain errors that grow as the distance between the reads increases. Additionally, it is not always possible to obtain the pairwise sequencing information or this information is subjective.

This paper proposes a method that uses short length reads to generate larger sequences using the information provided by overlapped fragments in order to generate longer fragments. The method, called LPS (Leader – Prefix – Suffix), is able to detect repeats using clustering techniques. Thus, this proposal would be equivalent to obtain longer length reads.

2 Related work

The idea of using the overlaps between prefixes and suffixes of strings has already been used by some assemblers such as SSAKE [6]. This approach uses the overlapping information between suffixes and prefixes of reads (considering a determined length and threshold value) in order to match pairs of fragments.

Prefixes and suffixes have also been used together with suffix trees to compute the similarity between pairs of DNA fragments. An example of such a tool is MUMer [7], which is used to compare pairs of sequences. Despite of not being a DNA fragment assembling algorithm, it gives us the idea of using the prefix and suffix as a comparison criterion in a clustering algorithm. This idea will be used in LPS.

The Celera Assembler [8] introduces the concept of unique overlapping, known as 'unitigs'. Unitigs are subsequences obtained by merging smaller sequences, with the property of having high level of reliability of belonging to the original sequence. Then, unitigs are used to construct longer sequences.

MaSuRCA [9] reduces the set of short length reads by creating a new set of longer length reads. This new set of longer length reads is known as super-reads. In this way, we have fewer reads to be assembled and the assembly process is less expensive.

The purpose of this paper is to provide a clustering based method that uses the information provided by overlaps between suffixes and prefixes of the sequences in order to generate longer sequences with a reliability level similar to unitigs.

3 The LPS Algorithm

In this research we propose the LPS method, which merges short length reads (25 to 500 base pairs) with a prefix or suffix overlap to create longer sequences. In order to merge the sequences, we perform a two-step clustering process. In the first step we use a macro clustering criterion in which sequences with overlapped prefix – suffix are grouped in the same cluster. In the second step we apply a micro clustering criterion to the sequences in a cluster. In this step, sequences sharing the same suffix or prefix are assigned to the same cluster, while sequences that do not adjust to this criterion are assigned to different clusters. Finally, sequences in the same cluster are merged to create longer sequences. Since all prefixes and suffixes are considered, when a longer sequence is created, it is possible to recognize repeated sequences. This allows disambiguating the position in the genome in which they may appear by using the prefix/suffix information in the longer sequence.

3.1 Clustering using the macro criterion

In the macro criterion clustering step we use the Leader cluster algorithm [10]. In the first step of this algorithm we select a sequence fragment as the leader of the cluster. For the rest of the fragments, we count the number of characters of the Leader prefix that overlap with the suffix of the fragment currently being analyzed. Similarly, we count the number of characters of the Leader suffix that overlap with the prefix of the

sequence. If any of these overlaps exceeds a threshold value, then the sequence is assigned to the cluster represented by the leader. In other case, a new cluster will be created with that sequence as its leader. **Fig. 1** shows (in gray color) the characters of the leader suffix that overlap with the sequence prefix being analyzed.

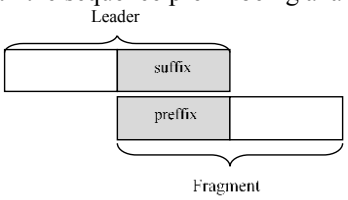

Fig. 1. Overlapping characters between the cluster leader suffix and a sequence fragment prefix

Let F denote the set of sequences fragments f obtained from a sequencer, C the set of clusters fragments, u a threshold value and $|s'|$ the cardinality of a sequence s'. The macro criterion clustering algorithm can be seen in Fig. 2:

```
macro_criterion_cluster(F, u)
input: F -> set of sequences fragments, u -> threshold value
output: C -> set of clusters fragments
for each fragment f ∈ F
  if C = ∅ then
    create a new cluster c
    add f to c and select f as leader
    add c to C
  else
    for each cluster c ∈ C
      l ← c.leader
      s' ← longest prefix of f that is suffix of l
      s'' ← longest suffix of f that is prefix of l
      k ← max(|s'|, |s''|)
      if k > u then
        add f to c
      else
        create a new cluster c'
        add f to c' and select f as leader
        add c' to C
```

Fig. 2. Macro criterion clustering algorithm

Table 1 shows the clusters obtained with the macro criterion algorithm. The first row shows the sequence AAAA, which is the leader of the cluster. The sequences AAAG, AAGG, AATG, and CAAA are fragments that belong to the cluster. The prefix of the fragments in rows 2, 3, and 4 has an overlap with the suffix of the leader of at least two characters. The substrings of these fragments that do not overlap are G, GG, and TG respectively and are shown in gray color. The last row shows a fragment for which its suffix overlaps with the prefix of the leader by 3 characters. Substring C does not overlap with any other subsequence and is shown in gray.

Table 1. Sequences cluster created with the macro criterion and a threshold of 2 characters

-	A	A	A	A	-	-
-	-	A	A	A	G	-
-	-	-	A	A	G	G
-	-	-	A	A	T	G
C	A	A	A	-	-	-

3.2 Clustering using the micro criterion

As can be noticed from **Table 1**, the sequences of the cluster can be merged with the leader, creating the new sequences: AAAAG, AAAAGG, AAAATG, and CAAAA. We can also note that fragment AAAAG is a substring of sequence AAAAGG. So, instead of generating a new sequence from merging the leader with another sequence of the cluster, it is possible to reduce the number of generated sequences by analyzing if one of them is substring of another one and keeping the longest one. Moreover, we only need to verify those substrings that are part of the fragments that do not overlap with the leader. **Table 1** shows these fragments in gray color. Some of these substrings are GG and TG. These ideas will be used to create a micro criterion clustering step.

In this clustering step, sequences belonging to a cluster can be divided in three sets: set L only containing the leader sequence, set P that contains the suffix of those sequences that overlap with the prefix of the leader, and set S that contains the prefix of the sequences that overlap with a suffix of the leader. As an example, in **Table 1** $L = \{AAAA\}$, $P = \{CAAAA\}$, and $S = \{AAAAG, AAAAGG, AAAATG\}$. Other interesting sets are: $\bar{P}$ which is the set of prefix fragments of P that do not overlap with L, i.e. $\bar{P} = \{p \in P - L\}$, and set $\bar{S}$ of suffix fragments of S that do not overlap with L, i.e. $\bar{S} = \{s \in S - L\}$. For the cluster shown in **Table 1**, $\bar{P} = \{C\}$ and $\bar{S} = \{G, GG, TG\}$.

The micro clustering process focuses on the strings of sets $\bar{P}$ and $\bar{S}$. For each of these sets, we identify which of the strings is a substring of another string in the same set. The strings that meet this condition can be merged as the longest length string containing those substrings. The string sets created using this criterion will be denoted as $\bar{P}'$ and $\bar{S}'$ and are named as reduced prefix and suffix sets, respectively.

For the strings in set $\bar{S} = \{G, GG, TG\}$, obtained from the fragments shown in **Table 1**, it is possible to join string G with string GG, so the new reduced suffix set will be $\bar{S}' = \{GG, TG\}$. In the case of set $\bar{P}$, which only contains one element, cannot be reduced, so $\bar{P}' = \bar{P} = \{C\}$.

Once we obtained the reduced prefix and suffix sets, it is possible to merge the fragments of the cluster in order to build new longer length fragments. The new fragments will belong to set $\bar{P}' \times L = \{pl | p \in \bar{P}' \wedge l \in L\}$ and $L \times \bar{S}' = \{ls | l \in L \wedge s \in \bar{S}'\}$, where the $\times$ operator is the Cartesian product of two sets. As the sequences belonging to $\bar{P}' \times L$ have L as suffix (which is the prefix of the sequences in the set $L \times \bar{S}'$) then, the fragment resulting from applying the micro criterion clustering process will yield to set $\bar{P}' \times L \times \bar{S}' = \{pls | p \in \bar{P}' \wedge l \in L \wedge s \in \bar{S}'\}$.

For the cluster shown in **Table 1**, the result of applying the micro criterion clustering process yields the sequences contained in $\bar{P}' \times L \times \bar{S}' = \{CAAAAGG, CAAAATG\}$.

Fig. 3 shows the micro criterion algorithm where C is the set clusters obtained using the macro criterion and C' the set created with the micro criterion algorithm:

```
micro_criterion_cluster(C)
input: C -> set of clusters fragments
output: C' -> set of sequences fragments
C' = ∅
for each cluster c ∈ C:
  L ← c.leader
  Compute P̄
  Compute S̄
  P̄' ← P̄
  S̄' ← S̄
  for each p ∈ P̄:
    if p is substring of t such us t ∈ P̄ − {p} then:
      P̄' ← P̄' − {p}
  for each s ∈ S̄:
    if s is substring of t such us t ∈ S̄ − {s} then:
      S̄' ← S̄' − {s}
  c' ← P̄' × L × S̄' = {pls|p ∈ P̄' ∧ l ∈ L ∧ s ∈ S̄'}
  C' ← c' ∪ C'
```

Fig. 3. Micro criterion clustering algorithm

In this phase, we use the information obtained from the sequences prefixes and suffixes to merge them and create longer sequences and avoiding the repeats. For example, given the fragments taken from the sequence $CAAAAGGTTTCAAAATG$, the sequence fragment $AAAA$ appears in two different positions (it is a repeat). For the sequences fragments shown in **Table 1**, using the macro and micro criterion clustering algorithm, the repeated versions of $AAAA$ are recognized as different ones because we generate fragments $CAAAAGG$ and $CAAAATG$ as we previously showed in the example. We can notice that these fragments are not repeats anymore.

4 Experiments and results

In our experiments we tested LPS as a preprocessing step to create larger sequences that fed an assembler system. In order to measure LPS's performance we compared the result obtained by the assembler when using it, with that obtained without using it. We used LPS with a plasmid sequence known as Bacillus cereus "ATCC 14579" (available at (Accesion NC_004721.2), which was synthetically sequenced. The experiments simulate ten independent *in silico* sequencing processes with coverage of 10 times the entire molecule. For each of these experiments, we performed the assembly process and reported results in different quality measures (the result of a quality measure is reported as the mean of the ten results obtained for that measure).

Our simulation of the sequencing process creates fragments with a uniform length of 50 characters, considering only one direction (5' to 3'), error free, and all characters from the sequences of the plasmid are contained at least in one fragment. The number of fragments obtained in each sequencing process oscillates from 6000 to 6500.

Common or standardized quality measures to evaluate the performance of an assembler do not exist [11], but we use the following quality metrics:

- *Contigs count*: An assembler should minimize the amount of fragments resulting from the assembling process. The best case would be that in which only one fragment is obtained, and this fragment corresponds to the original sequence before it was fragmented.
- *Sum of contigs length*: The sum of the fragments constructed by an assembler should approximate the sum of the contigs length of the original sequence. The closest the sum of the length of the constructed fragment to the size of the original sequence, the better the assembler.
- *Error percentage*: Because the initial fragments are error free, a contig is considered to be correct if it is a substring of the original sequence. In any other case, it will be considered as an erroneous contig. For this, the quality measure used is the contig error percentage from the total assembled contigs. The lower the error percentage, the better is the assembler. An assembly will be accepted as valid if its error percentage is lower than 1%.

LPS is considered a sequence fragment preprocessing algorithm because it only takes into account the fragments prefix and suffix information. When larger sequences are created, we could find sequences that are contained in the middle of another fragment, and LPS would not be able to merge such sequences. That is why we need to call a DNA sequence assemble algorithm after preprocessing the fragments with LPS. In our experiments we use a sequence assembly tool known as PadeNA [12]. PadeNA is an open source bioinformatics framework for.NET. LPS was implemented in the C# programming language, just as PadeNA. The PadeNA assembly algorithm relies in the use of *de Bruijn graphs* [13]. These graphs represent DNA sequences in their edges. Then, the sequence assembling problem could be translated into the problem of finding a path that passes through all the graph edges. This problem is knows as finding an Eulerian path in the graph [14]. Representing whole fragments in the graph edges yields a very complex graph. In order to optimize the graph size, our fragments are split into substrings of length k, known as k-mers. In our experiments, we look for a value of k that yields suitable quality values (described above) for the assembly generated by PadeNA.

4.1 Finding the size of the k-mers for PadeNA

In this section, we look for the value of k for the length of k-mers that yields a good assemble using PadeNA (without preprocessing the fragments with LPS). The quality values obtained from this assembly will be used as our baseline to compare the performance of PadeNA when working with and without LPS. We tested the values of k from 2 to 31 which are the admitted values for PadeNA.

Fig. 4 shows the mean of the sum of the contigs length for the fragments of the ten sequencing experiments. The horizontal line with value 15,274 shows the length of the original sequence. A value of k, producing a sum of contigs closer to 15,274, generates a better assemble. The range of values for k that better approximates the horizontal line

ranges from 13 to 24. In order to reduce this interval to obtain some candidate values of k, we analyze the contigs count and the error percentage.

Fig. 5 shows the mean of the number of contigs created for the fragments of the ten sequencing processes. The lower this value, the better the assembly. Looking at values of k from 13 to 24, the number of contigs increases from 180 to 250. The preferred value of k to select would be the smallest one with lowest error in this range.

Fig. 6 shows the error percentage. Since for all values of k the error percentage is less than 1, the assemblies obtained for the different values of k in the interval from 13 to 24 are accepted. The lowest value of k with 0% of error is $k = 17$.

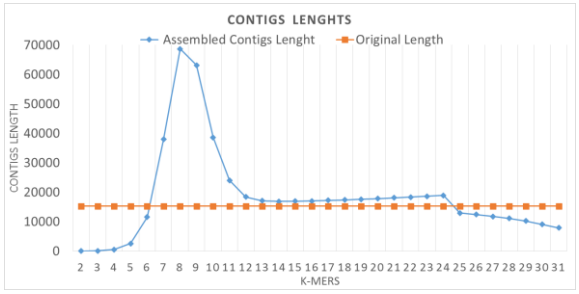

Fig. 4. Sum of the contigs length

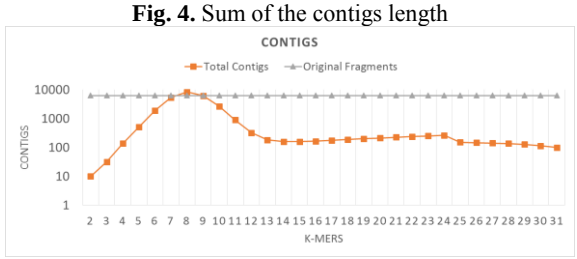

Fig. 5. Contigs count

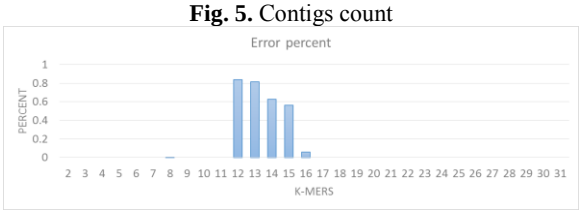

Fig. 6. Contigs error percentage

For our 17-mers, the quality values obtained are: 17,199 for the sum of the contigs length, 176 contigs, and 0% error. These values will be used as baseline to compare the performance of LPS + PadeNA.

4.2 Result obtained with LPS + PadeNA

The LPS preprocessing algorithm has a parameter "n" to control the number of characters required to overlap a prefix with a suffix. LPS is applied two times with different values of n. The first run is applied to the fragments obtained from the sequencing process and the second is applied to the fragments obtained from the first run. A third run was tested, but there were not changes in the fragments set so it was not considered in

our results. We refer to the first and second run of LPS as LPS-1 and LPS-2 respectively. The scheme of values of n used were:

1. n characters for LPS-1 - n for LPS-2: For this experiment we set n to a fix number of characters for the first and second run of LPS. We tested values of $n \in \{5, 10, 15, 20, 25, 30, 35, 40, 45\}$. These values are shown as a dotted line in **Fig. 7** and **Fig. 8**. In **Fig. 9**, this experiment is shown in the first column for each value of n.
2. $n\%$ for LPS-1 - $n\%$ for LPS-2: In this case we set n to a fix percentage of the length of the characters of the sequence in the first and second run of LPS. The values of n tested were $n \in \{10\%, 20\%, 30\%, 40\%, 50\%, 60\%, 70\%, 80\%, 90\%\}$. The result of this experiment is shown as a discontinuous line in **Fig. 7** and **Fig. 8**, and in **Fig. 9** it corresponds to the middle column for each value of n.
3. n characters for LPS-1 - 20% for LPS-2: In this experiment we set n to a fix number of characters in the first run of LPS and 20% of the length of the characters of the sequence in the second one. The values of n tested for the first run were $n \in \{5, 10, 15, 20, 25, 30, 35, 40, 45\}$. These results are shown as dots in **Fig. 7** and **Fig. 8**, and in **Fig. 9** they correspond to the third column for each value of n. The election of 20% for the second run will be explained below.

Fig. 7 to **Fig. 9** show the evaluation of the three LPS schemes through the described quality measures. The measured values were obtained as the mean of the quality measure computed from the ten sequencing processes performed. **Fig. 7** shows the number of contigs mean from the fragments of the ten sequencing processes. The horizontal continuous line with a value of 176 corresponds to the contigs obtained with PadeNA (without LPS). It is expected that LPS-PadeNA improves PadeNA's performance when results appear below this line. **Fig. 8** shows the mean of the sum of the contigs length for the ten sequencing processes. The horizontal continuous line with a value of 17,199 corresponds to the sum of the contigs length obtained with PaneNA. The horizontal continuous line with a value of 15,274 corresponds to the length of the original sequence. We expect that LPS-PadeNA enhances PadeNA's result when obtaining a value lower than 17,199 and better results are found for a value close to 15,274. **Fig. 9** shows the mean of the error percentage for the ten sequencing processes. If this value is higher than 1%, the assembly is rejected, otherwise it is accepted.

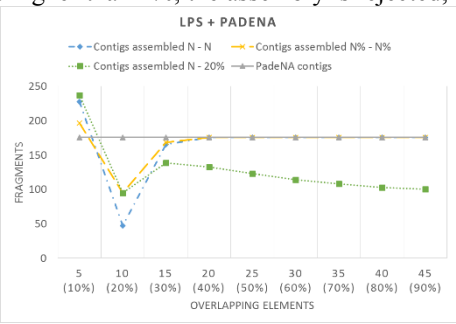

Fig. 7. Contigs generated

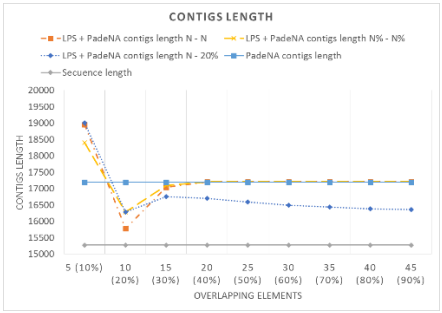

Fig. 8. Sum of contigs length

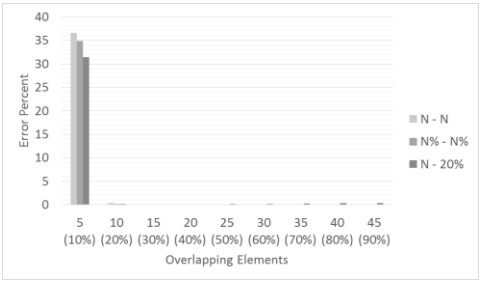

Fig. 9. Contigs error percentage

As shown in **Fig. 7**, **Fig. 8**, and **Fig. 9**, if we use a low value for n, such as 5 characters or 10 percent of the length of the sequences, the assembled fragments have an error above 1%, so these are rejected. As we reduce the number of characters to consider an overlap between two sequences, it is more common to create new sequences that do not match with a substring of the original sequence. We obtain the best results when we require n = 10 or 20% of the fragment's characters overlap in order to consider a match. In this case, we obtain the lowest contigs count and even less contigs than those obtained using PadeNA. The sum of the contigs length is lower than that obtained with PadeNA and these values are closer to the length of the original sequence. This result was obtained because the number of required overlapping characters is enough to create new sequences with a low rate of errors and the length of the new sequence is larger enough to be a non-repeat sequence. The percentage error for these sequences is less than 1%, so the assembly is considered as valid. For this reason, we chose a value of 20% of the sequence length for a second run of LPS-PadeNA in the third experimental scheme proposed above.

When we run LPS-PadeNA with a value of n = 15 characters or 30% of the characters of the fragments length required to overlap, the quality of the assembly is very similar to that obtained by PadeNA with a small improvement in quality. This occurs because in this case we require more overlapping characters in order to generate new fragments. Then, we obtain a smaller amount of shorter length merged sequences and consequently, we obtain more sequences with repeats. Because of this, for the first and second experiments using values greater than or equal to 20 characters or 30% of the fragments length required to overlap, the resulting assemblies are similar to those obtained with PadeNA. This suggests that if LPS-PadeNA is used with a high number of characters required to overlap, the resulting assembly will be similar to that of PadeNA. Moreover, the number of input fragments is reduced by LPS to be used as input to PadeNA (LPS-PadeNA), and the assembly process will be less complex.

On the other hand, for the experimental scheme of n characters for LPS-1 and - 20% for LPS-2, the contigs count and the sum of contigs length decreases when using 15 characters (30%) in the first run and 20% on the second one. The error percentage is less than 1%, so these assemblies are accepted. These results could be obtained because in the first run of LPS we required a high number of characters to overlap, so the erroneous fragments created in the first run of LPS were just a few. In the second run of LPS, the amount of characters required to overlap are less than those required in first

run, so the fragments created at this phase have longer length and the number of erroneous contigs generated is very small. Then, it was possible to avoid repeats in the new generated fragments. The best result found with LPS-PadeNA for *Bacillus cereus* reduced the number of contigs in 25%, the sum of contigs length in 9% with an error of 0.42% compared to the result obtained with PadeNA.

5 Conclusions

LPS is a preprocessing strategy for short length DNA fragments which allows creating longer length fragments using clustering techniques. This technique allows reducing repeats and gaps in contigs in the assembling process. It is also able to reduce the number of contigs and sum of contigs length of the resulting assembly.

In the experiments we showed that with two runs of LPS and the use of an assembling algorithm such as PadeNA, we improve the resulting assembly compared to the one obtained when using the assembling algorithm by itself (without LPS).

A good configuration of threshold values for LPS can be achieved by using a small number of required characters to overlap, i.e. 20% of the fragment length for the two runs of LPS, in order to obtain a small number of contigs and sum of the length of these contigs close to that of the length of the original sequence.

When we require a high number of characters to overlap in LPS, the assembly obtained is similar to the result obtained when applying an assembly algorithm by itself (i.e. PadeNA). LPS in this case is useful for reducing the number of fragments to be passed to the assembler. It is also possible to obtain a good assembly with a small number of contigs count when a high number of characters is required to overlap in a first run of LPS and then decrease this number for the second run of LPS.

LPS is not an assembly algorithm because it only merges sequence fragments using a prefix and suffix criterion, but it does not consider the fragments that are completely contained within a larger fragment. Then, it is necessary to apply an assembly algorithm after LPS is executed.

6 Future work

Experiments were made on error-free sequences. In the next extension for LPS we will use an error correction strategy to then apply LPS and measure the quality of the assembly. We will also implement a strategy that allows the processing of complement sequences. Besides, on the second micro grouping phase of LPS (when merging fragments), we could generate sequences that are not substring of the original sequence, requiring to add a heuristic to discard these fragments. Finally, in this research we used PadeNA as the assembly algorithm to evaluate the performance of our algorithm, we will evaluate LPS with other assembling algorithms.

7 References

1. Margulies, M., Egholm, M., Altman, W.E., Attiya, S., Bader, J.S., Bemben, L.A., Berka, J., Braverman, M.S., Chen, Y.-J., Chen, Z., Dewell, S.B., Du, L., Fierro, J.M., Gomes, X.V., Godwin, B.C., He, W., Helgesen, S., Ho, C.H., Irzyk, G.P., Jando, S.C., Alenquer, M.L.I., Jarvie, T.P., Jirage, K.B., Kim, J.-B., Knight, J.R., Lanza, J.R., Leamon, J.H., Lefkowitz, S.M., Lei, M., Li, J., Lohman, K.L., Lu, H., Makhijani, V.B., McDade, K.E., McKenna, M.P., Myers, E.W., Nickerson, E., Nobile, J.R., Plant, R., Puc, B.P., Ronan, M.T., Roth, G.T., Sarkis, G.J., Simons, J.F., Simpson, J.W., Srinivasan, M., Tartaro, K.R., Tomasz, A., Vogt, K.A., Volkmer, G.A., Wang, S.H., Wang, Y., Weiner, M.P., Yu, P., Begley, R.F., Rothberg, J.M.: Genome sequencing in microfabricated high-density picolitre reactors. Nature 437, 376-380 (2005)
2. Voelkerding, K.V., Dames, S.A., Durtschi, J.D.: Next-Generation Sequencing: From Basic Research to Diagnostics. Clinical Chemistry 55, 641-658 (2009)
3. Pham, S., Antipov, D., Sirotkin, A., Tesler, G., Pevzner, P., Alekseyev, M.: Pathset Graphs: A Novel Approach for Comprehensive Utilization of Paired Reads in Genome Assembly. In: Chor, B. (ed.) Research in Computational Molecular Biology, vol. 7262, pp. 200-212. Springer Berlin Heidelberg (2012)
4. Pevzner, P.A., Tang, H.: Fragment assembly with double-barreled data. Bioinformatics 17, S225-S233 (2001)
5. Bankevich, A., Nurk, S., Antipov, D., Gurevich, A.A., Dvorkin, M., Kulikov, A.S., Lesin, V.M., Nikolenko, S.I., Pham, S.K., Prjibelski, A.D., Pyshkin, A., Sirotkin, A., Vyahhi, N., Tesler, G., Alekseyev, M.A., Pevzner, P.A.: SPAdes: A New Genome Assembly Algorithm and Its Applications to Single-Cell Sequencing. Journal of Computational Biology 19, 455-477 (2012)
6. Warren, R.L., Sutton, G.G., Jones, S.J.M., Holt, R.A.: Assembling millions of short DNA sequences using SSAKE. Bioinformatics 23, 500-501 (2007)
7. Kurtz, S., Phillippy, A., Delcher, A., Smoot, M., Shumway, M., Antonescu, C., Salzberg, S.: Versatile and open software for comparing large genomes. Genome Biol 5, 1-9 (2004)
8. Myers, E., Sutton, G., Delcher, A., Dew, I., Fasulo, D., Flanigan, M., Kravitz, S., Mobarry, C., Reinert, K., Remington, K., Anson, E., Bolanos, R., Chou, H.-H., Jordan, C., Halpern, A., Lonardi, S., Beasley, E., Brandon, R., Chen, L., Dunn, P., Lai, Z., Liang, Y., Nusskern, D., Zhan, M., Zhang, Q., Zheng, X., Rubin, G., Adams, M., Venter, C.: A Whole-Genome Assembly of Drosophila. Science 287, 2196-2204 (2000)
9. Zimin, A., Marçais, G., Puiu, D., Roberts, M., Salzberg, S., Yorke, J.: The MaSuRCA genome assembler. Bioinformatics 29, 2669-2677 (2013)
10. Hartigan, J.: Clustering Algorithms. Books on Demand (1975)
11. Narzisi, G., Mishra, B.: Comparing De Novo Genome Assembly: The Long and Short of It. PLoS ONE 6, e19175 (2011)
12. Thareja, G., Kumar, V., Zyskowski, M., Mercer, S., Davidson, B.: PadeNA: A Parallel De Novo Assembler. In: Pellegrini, M., Fred, A.L.N., Filipe, J., Gamboa, H. (eds.) BIOINFORMATICS, pp. 196-203. SciTePress (2011)
13. De Bruijn, N.G.: A combinatorial problem. Koninklijke Nederlandse Akademie v. Wetenschappen 49, 758-764 (1946)
14. Skiena, S.S.: The Algorithm Design Manual: Text. TELOS, The Electronic Library of Science (1998)