algorithms

Article

DNA Paired Fragment Assembly Using Graph Theory

J. Emilio Quiroz-Ibarra [1,*], Guillermo M. Mallén-Fullerton [2] and Guillermo Fernández-Anaya [3]

[1] Physics and Mathematics, Universidad Iberoamericana Ciudad de México, Prolongación Paseo de la Reforma 880, Lomas de Santa Fe, Ciudad de México 01219, Mexico
[2] Engineering Department, Universidad Iberoamericana Ciudad de México, Prolongación Paseo de la Reforma 880, Lomas de Santa Fe, Ciudad de México 01219, Mexico; guillermo.mallen@ibero.mx
[3] Physics and Mathematics Department, Universidad Iberoamericana Ciudad de México, Prolongación Paseo de la Reforma 880, Lomas de Santa Fe, Ciudad de México 01219, Mexico; guillermo.fernandez@ibero.mx
[*] Correspondence: quirozem@yahoo.com.mx; Tel.: +52-1-55-1393-8302

Academic Editor: Andras Farago
Received: 26 January 2017; Accepted: 17 March 2017; Published: 24 March 2017

Abstract: DNA fragment assembly requirements have generated an important computational problem created by their structure and the volume of data. Therefore, it is important to develop algorithms able to produce high-quality information that use computer resources efficiently. Such an algorithm, using graph theory, is introduced in the present article. We first determine the overlaps between DNA fragments, obtaining the edges of a directed graph; with this information, the next step is to construct an adjacency list with some particularities. Using the adjacency list, it is possible to obtain the DNA *contigs* (group of assembled fragments building a contiguous element) using graph theory. We performed a set of experiments on real DNA data and compared our results to those obtained with common assemblers (*Edena* and *Velvet*). Finally, we searched the *contigs* in the original genome, in our results and in those of *Edena* and *Velvet*.

Keywords: DNA fragment assembly; Trie data structure; graph theory algorithms

1. Introduction

Each monomer comprising the DNA polymer is formed with a pentose, a phosphate group and one of four nitrogenous bases: adenine, guanine, cytosine and thymine. In 1953, scientists James Watson and Francis Crick [1] discovered the double-helix spatial structure of the DNA molecule. This double chain is coiled around a single axis, and the strands are attached by hydrogen bridges between pairs of opposite bases. The direction of the polymer chain is determined by the pentose carbon atoms $5'$ and $3'$ (the beginning and ending positions of a DNA strand are denoted as $5'$ and $3'$, respectively; a DNA strand is read in the $5'$ to $3'$ direction, and the complementary strand runs in the opposite direction). The bases in each pair are complementary. The content of adenine (**A**) is the same as the content of thymine (**T**), and the cytosine (**C**) content is the same as that of guanine (**G**) (Figure 1).

In 1975, Frederik Sanger [2] proposed a DNA sequencing technique that involved detecting small dark bands in a thin gel using electrophoresis. Sanger proposed cutting the DNA molecule at specific points in the sequence with restriction enzymes [3]. This method is slow and costly; with each digestion, the sample must be divided, and each new division must be cloned to obtain a sufficient amount of material. To reduce processing time, Sanger et al. [4,5] proposed splitting the DNA sequences at random points. The disadvantage of this method is that the order of the fragments is unknown, generating an NP-Complete (Non-deterministic polynomial time) [6] computational problem. This method is known as the *shotgun technique*.

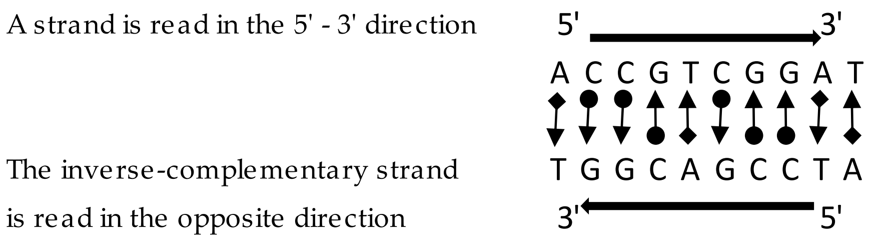

A strand is read in the 5' - 3' direction
5'
3'
A C C G T C G G A T
T G G C A G C C T A
The inverse-complementary strand
is read in the opposite direction
3'
5'

Figure 1. Reading a strand in direct and inverse-complementary.

Rodger Staden [7] proposed a method to assemble the genome using a computer. As the DNA fragments are produced from many copies of the original genome, more than one fragment comes from the same region. The DNA fragments should be processed while looking for overlaps, coincidences in the extremes of the fragments. The total number of bases in the fragments divided by the total number of bases in the complete genome is called the *coverage*. If the *coverage* is high enough, it is possible to rebuild the genome, but it is difficult to obtain high-quality results because false overlaps can be generated due to sequencing errors, sample contamination with foreign DNA and chimeras (the cloning process is carried out using host bacteria, and sometimes the DNA of the sample is concatenated with that of the host bacteria, producing what is known as a *chimera*). In our experiments, we noticed that some DNA sections might not be sampled and that complete genome reconstruction would therefore not be possible. What can be obtained is a set of *contigs* covering most of the genome, and it is the job of a molecular biologist to assemble the *contigs* using other techniques.

Later, James Weber and Eugene Myers [4] proposed the generation of paired fragments in the *shotgun* process. Traditional sequencing starts from the $5'$ end of each piece of DNA, and only a limited number of bases can be sequenced. In NGS (Next-Generation Sequencing), the number of bases is always the same, yielding fixed-size fragments; however, the DNA pieces sequenced are generally longer. The goal of the paired fragments procedure is to obtain a sequence from the $3'$ end as well as one from the $5'$ end, obtaining two fragments from the same piece of DNA. This would help to establish an interval in the DNA sequence associated with the paired fragments. Unfortunately, sequencing from the $3'$ end is difficult and produces a relatively large number of sequencing errors.

DNA sequencing technology has advanced, decreasing costs and process time. Today, there are databases with information about many genomes, including the human genome and those of many disease-causing microorganisms [8]. Steven Salzberg [9] developed a comparative study of DNA sequence assemblers; tests were carried out with fragment sets obtained using Illumina equipment (*Illumina*: http://www.illumina.com/, Illumina, Inc., San Diego, CA, USA). The fragments' lengths were found by NGS technology to be in the range of 50 to 150 bases [10]. The Salzberg study was developed in 2012; nevertheless, the paradigm remains the same today, and the Salzberg study is therefore still applicable. The primary metric in the evaluation was N50 (shortest sequence length close to the median of a set of *contigs*). Salzberg employed four organisms in the test, including a *Staphylococcus aureus* problem. These results are particularly interesting because the tests we present in this article have been obtained using a problem with the same bacterium. Salzberg did not include the *Edena assembler* (http://www.genomic.ch/edena.php, Genomic Research Institute, Geneva, Switzerland.), developed by David Hernández [11] in his research. In our comparative study, this assembler is an important reference.

Initially, *greedy* (Algorithmic technique that at each step tries to generate the optimal solution of that step of the problem without considering the rest of the problem) algorithms [12] were used to find the order of the fragments; later, the *de Bruijn* graph [13] was introduced with different values for the *k-mer* (section of k consecutive bases) [13]. Most of the assemblers available today are based on *de Bruijn* graphs. In our comparative study, we include the *Velvet assembler* (http://www.ebi.ac.uk/~zerbino/velvet/, EML-EBI, Cambridge, UK), developed by Zerbino [14], which applies *de Bruijn* graphs.

Parsons et al. [15] proposed an optimization with a genetic algorithm to maximize the sum of the fragment overlaps, but the obtained *contigs* were relatively short and processing speed was low. Later, Mallén-Fullerton and Fernández-Anaya [16] suggested a reduction of the fragments' assembly problem to the traveling salesman problem (TSP) that has been studied extensively. Solution methods with relatively good efficiency exist for the TSP. Applying heuristics and algorithms, they obtained optimal solutions for several commonly used benchmarks, and for the first time, a real-world problem was solved by using optimization. Using graph theory and setting the appropriate objective functions, Mallén et al. [17] developed a new assembly method from the perspective of a directed graph, looking for the reduction of the algorithm's complexity.

In this publication, taking Mallén et al. [17] as a starting point, we developed a new algorithm working with the *paired fragments* (two records identified by "/1" and "/2", respectively, conforming to an interval of the sequence of DNA) resulting from the sequencing results of the *Illumina equipment.* In our initial tests, we found some *contigs* that were not located in the published genome for the same organism, even though these *contigs* were properly obtained. Empirically, we found that when these *contigs* were split at certain points, all of the pieces could, in most cases, be found in the genome. Using the information from the paired fragments information, we could find the locations where a *contig* should be split to increase the quality of the assembly.

To obtain the overlaps between DNA fragments, we used a *Trie* [17]. Using the overlaps as edges, a directed graph has been obtained, and we could build a set of *contigs* improving the N50 [8] of the previous release [17]. The interval of the paired fragments was a critical factor in our success. In our experiments, we worked with the sequencing data of real-life organisms obtained from *Illumina equipment*, maximizing the lengths of the *contigs* as the objective function.

In Section 2 of this paper, we present the applied graph theory elements, data structures and algorithms. In Section 3, we reveal the developed algorithms that solve the assembly problem. Section 4 sets out the application of this new model to a real-life problem comparing our assembler with other assemblers (*Velvet* and *Edena*), and finally, in Section 5, we present our conclusions and ideas for future work.

2. Graph Theory in the DNA Fragment Assembly Problem

2.1. Generalities

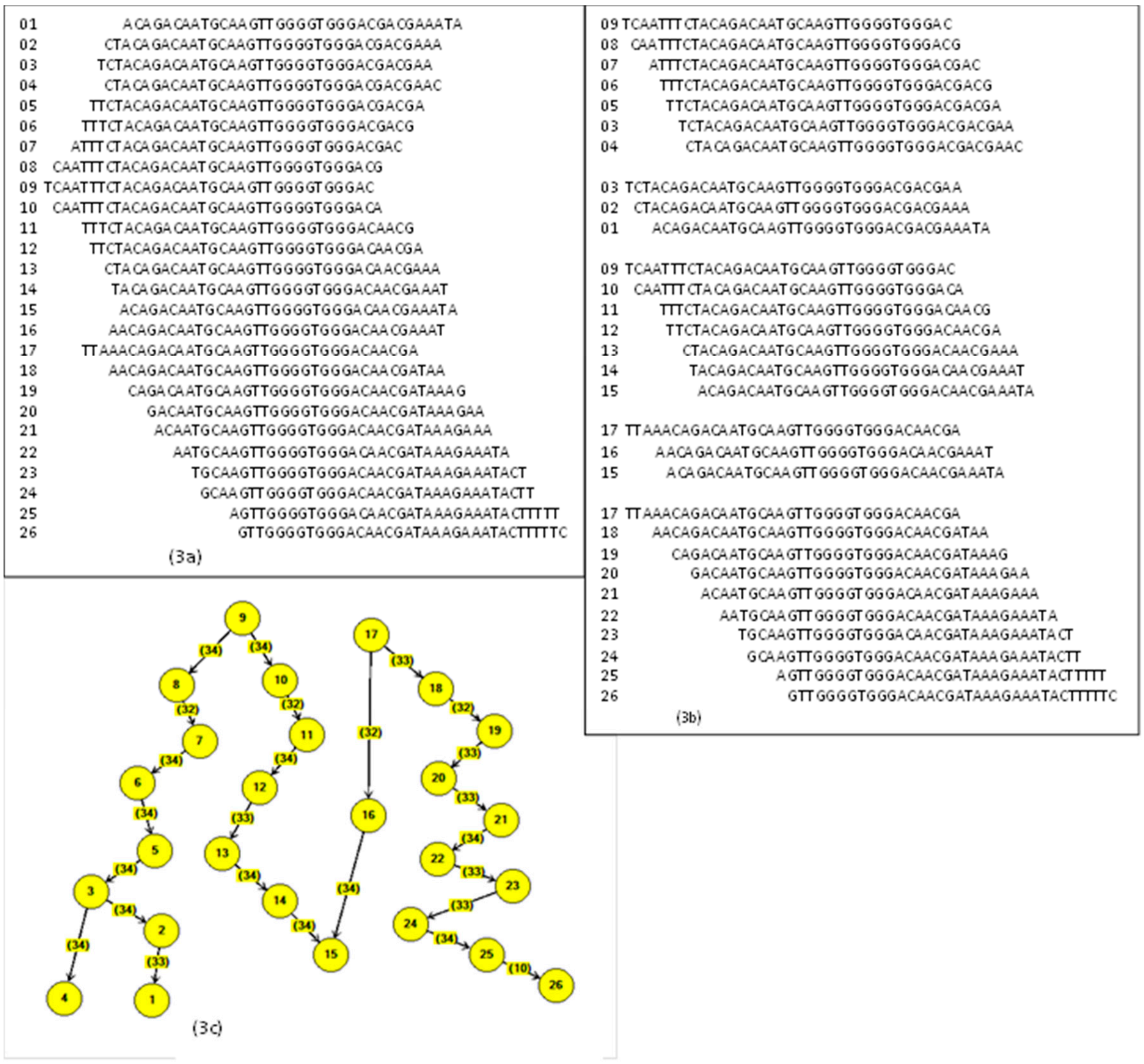
A graph $G = (V, E)$ is a set of vertices V and edges E; the vertices are linked by the edges, and, on each side of the edge, only one vertex exists. Some graphs are non-directional; nevertheless, if the system requires a direction, it will activate an ordered paired of vertices, and the result will be called a directed graph. One paradigm for DNA fragment assembly using overlapping fragments is based on a graph. Several models have been developed, and these have had variable outcomes with respect to the algorithm's complexity and efficiency and the quality of the results in the assembled DNA. The algorithms based on graph theory include *de Bruijn* graphs [13], Eulerian paths [10], Hamiltonian paths [10] and Depth-First search (DFS) [17].

2.2. The Shotgun Technique

In the sequencing process, a sample of DNA is broken into many fragments [10]. Next-Generation Sequencing (NGS) produces very short fragments, all the same size, usually between 25 and 500 bases. The cuts are made in random places, usually producing millions of fragments. Each fragment is sequenced, and the results are stored in a FASTA (plain-text file format used to represent and store genetic information) file format [18]. Figure 2 illustrates the *shotgun technique* [10]. Each fragment overlaps with other fragments; the fragments are the graph vertices, and the number of overlapped bases are the edge weights. In Figure 3, a graph model example is shown.

```
Sequenced info:         TTCACTTATTTAAAATCTGGAAGAAACCTAGG
fragment 1 left cut:    TTCACTTATTTAAAATCTGGAAGA
fragment 2 right cut:           TTTAAAATCTGGAAGAAACCTAGG
Overlap assembly:       TTCACTTATTTAAAATCTGGAAGAAACCTAGG
                                |   <= 16 overlaps =>   |
```

Figure 2. Shows the *shotgun* process for a DNA sequence and the assembly with overlaps.



```
01          ACAGACAATGCAAGTTGGGGTGGGACGACGAAATA
02        CTACAGACAATGCAAGTTGGGGTGGGACGACGAAA
03       TCTACAGACAATGCAAGTTGGGGTGGGACGACGAA
04        CTACAGACAATGCAAGTTGGGGTGGGACGACGAAC
05      TTCTACAGACAATGCAAGTTGGGGTGGGACGACGA
06     TTTCTACAGACAATGCAAGTTGGGGTGGGACGACG
07    ATTTCTACAGACAATGCAAGTTGGGGTGGGACGAC
08  CAATTTCTACAGACAATGCAAGTTGGGGTGGGACG
09 TCAATTTCTACAGACAATGCAAGTTGGGGTGGGAC
10  CAATTTCTACAGACAATGCAAGTTGGGGTGGGACA
11     TTTCTACAGACAATGCAAGTTGGGGTGGGACAACG
12      TTCTACAGACAATGCAAGTTGGGGTGGGACAACGA
13        CTACAGACAATGCAAGTTGGGGTGGGACAACGAAA
14         TACAGACAATGCAAGTTGGGGTGGGACAACGAAAT
15          ACAGACAATGCAAGTTGGGGTGGGACAACGAAATA
16         AACAGACAATGCAAGTTGGGGTGGGACAACGAAAT
17      TTAAACAGACAATGCAAGTTGGGGTGGGACAACGA
18         AACAGACAATGCAAGTTGGGGTGGGACAACGATAA
19           CAGACAATGCAAGTTGGGGTGGGACAACGATAAAG
20             GACAATGCAAGTTGGGGTGGGACAACGATAAAGAA
21              ACAATGCAAGTTGGGGTGGGACAACGATAAAGAAA
22                AATGCAAGTTGGGGTGGGACAACGATAAAGAAATA
23                  TGCAAGTTGGGGTGGGACAACGATAAAGAAATACT
24                   GCAAGTTGGGGTGGGACAACGATAAAGAAATACTT
25                      AGTTGGGGTGGGACAACGATAAAGAAATACTTTTT
26                       GTTGGGGTGGGACAACGATAAAGAAATACTTTTTC
                   (3a)
```

```
09 TCAATTTCTACAGACAATGCAAGTTGGGGTGGGAC
08  CAATTTCTACAGACAATGCAAGTTGGGGTGGGACG
07    ATTTCTACAGACAATGCAAGTTGGGGTGGGACGAC
06     TTTCTACAGACAATGCAAGTTGGGGTGGGACGACG
05      TTCTACAGACAATGCAAGTTGGGGTGGGACGACGA
03       TCTACAGACAATGCAAGTTGGGGTGGGACGACGAA
04        CTACAGACAATGCAAGTTGGGGTGGGACGACGAAC

03 TCTACAGACAATGCAAGTTGGGGTGGGACGACGAA
02  CTACAGACAATGCAAGTTGGGGTGGGACGACGAAA
01    ACAGACAATGCAAGTTGGGGTGGGACGACGAAATA

09 TCAATTTCTACAGACAATGCAAGTTGGGGTGGGAC
10  CAATTTCTACAGACAATGCAAGTTGGGGTGGGACA
11     TTTCTACAGACAATGCAAGTTGGGGTGGGACAACG
12      TTCTACAGACAATGCAAGTTGGGGTGGGACAACGA
13        CTACAGACAATGCAAGTTGGGGTGGGACAACGAAA
14         TACAGACAATGCAAGTTGGGGTGGGACAACGAAAT
15          ACAGACAATGCAAGTTGGGGTGGGACAACGAAATA

17 TTAAACAGACAATGCAAGTTGGGGTGGGACAACGA
16    AACAGACAATGCAAGTTGGGGTGGGACAACGAAAT
15     ACAGACAATGCAAGTTGGGGTGGGACAACGAAATA

17 TTAAACAGACAATGCAAGTTGGGGTGGGACAACGA
18    AACAGACAATGCAAGTTGGGGTGGGACAACGATAA
19      CAGACAATGCAAGTTGGGGTGGGACAACGATAAAG
20        GACAATGCAAGTTGGGGTGGGACAACGATAAAGAA
21         ACAATGCAAGTTGGGGTGGGACAACGATAAAGAAA
22           AATGCAAGTTGGGGTGGGACAACGATAAAGAAATA
23             TGCAAGTTGGGGTGGGACAACGATAAAGAAATACT
24              GCAAGTTGGGGTGGGACAACGATAAAGAAATACTT
25                 AGTTGGGGTGGGACAACGATAAAGAAATACTTTTT
26                  GTTGGGGTGGGACAACGATAAAGAAATACTTTTTC
         (3b)
```

Figure 3. Directed graph from a group of fragments. (**3a**) Semi assembled fragments; (**3b**) Overlapped fragments; (**3c**) Resulting graph from the overlap analysis.

2.3. Pair Generation

The overlaps between fragments are obtained using a *Trie* data structure [19]. With this data structure, it is possible to calculate only the real overlaps without checking all of the possible pairs or the overlaps that are smaller than a predefined length. In the worst-case scenario, the search complexity of the *Trie* is *O(l)*, where *l* is the fragment length. If *n* is the number of fragments in the problem, the complexity to obtain all the overlaps is *O(nl)*. In addition, the *Trie* has the advantage of storing no duplicate data [20].

2.4. Adjacency List

A proper way to manage sparse graphs, as was applied in our case, is the *adjacency list* data structure [21]. We found the use of a linked list appropriate for our problem, with a header section as a dictionary. Each header value is a starting point for the adjacency elements stored in the same data structure, which really contains a set of *adjacency lists*. The input and output degrees for each vertex are calculated while the overlaps are loaded. Each overlap has a preceding and a subsequent fragment that determine the direction of the edges of the directed graph. The vertices with zero input degree value are the starting point for a path. The path with a larger sum of weights is a *contig* in our algorithms.

Because of the fragment overlaps, each base in a *contig* can appear in more than one fragment, as shown in Figure 4. The *consensus* is the number of times that a nucleotide appears in the same position of a *contig*. We considered a minimum *consensus* to accept a base in a *contig*. The base found most frequently in each position of a *contig* is accepted if its frequency is over the minimum *consensus* previously defined. In the example presented in Figure 4, a minimum *consensus* of four is accepted.

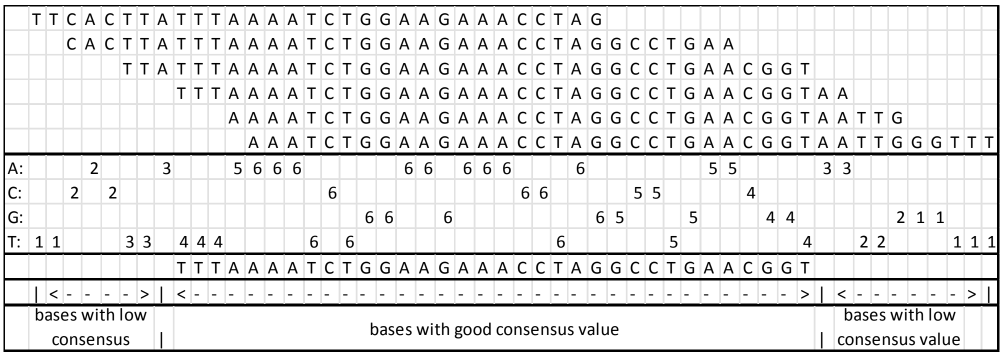

```
   T T C A C T T A T T T A A A A T C T G G A A G A A A C C T A G
       C A C T T A T T T A A A A T C T G G A A G A A A C C T A G G C C T G A A
             T T A T T T A A A A T C T G G A A G A A A C C T A G G C C T G A A C G G T
                   T T T A A A A T C T G G A A G A A A C C T A G G C C T G A A C G G T A A
                         A A A A T C T G G A A G A A A C C T A G G C C T G A A C G G T A A T T G
                           A A A T C T G G A A G A A A C C T A G G C C T G A A C G G T A A T T G G G T T T
A:       2       3       5 6 6 6           6 6   6 6 6       6             5 5         3 3
C:     2   2                       6                   6 6         5 5         4
G:                                     6 6     6               6 5       5       4 4           2 1 1
T: 1 1       3 3   4 4 4         6   6                     6           5             4     2 2       1 1 1
                   T T T A A A A T C T G G A A G A A A C C T A G G C C T G A A C G G T
   | < - - - - > | < - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - > | < - - - - - - > |
   bases with low                        bases with good consensus value                  bases with low
      consensus  |                                                                      | consensus value
```

Figure 4. *Consensus* effect in the *contig* assembly.

2.5. Specific Characteristics

Our objective is to obtain the paths that maximize the sum from the weights of the edges contained in the path [17]. Each path starts at a node with zero-input degree and ends at a node with zero-output degree. Notice that several starting nodes can connect to a shared group of nodes. In this case, we keep only the path with the maximum edge sum. From the selected paths, it is possible to assemble a set of *contigs*.

In the assembly process for the *contig*, the *consensus* value [17] must be selected. In the experiments presented in this article, we used a *consensus* of 4. Bases with a *consensus* lower than the specified value are discarded. A *consensus* value greater than or equal to the specified value indicates a good-quality base, as shown in Figure 4. Usually, the lowest *consensus* values are found in the extremes of the *contig*.

To verify the quality of the *contigs*, we searched for them in the original genome. Those not located were separated into several segments, searching for each one in the genome. Some of these fragments were located, while a few were not, indicating that the prediction of the split point was not always accurate. The existence of paired fragments in the *contig* is a guarantee that the content of the interval is correct, and this gave us the criterion for the split point with respect to the interval of the paired fragments.

Figure 5 shows two intervals; the split point of the *contig* is outside the intervals. All of the elements outside of the intervals can be discarded because there is no confirmation with the paired fragments ("/1", "/2"). It should be noted that the N50 [9] value will be reduced; meanwhile, the new *contigs* are of better quality.

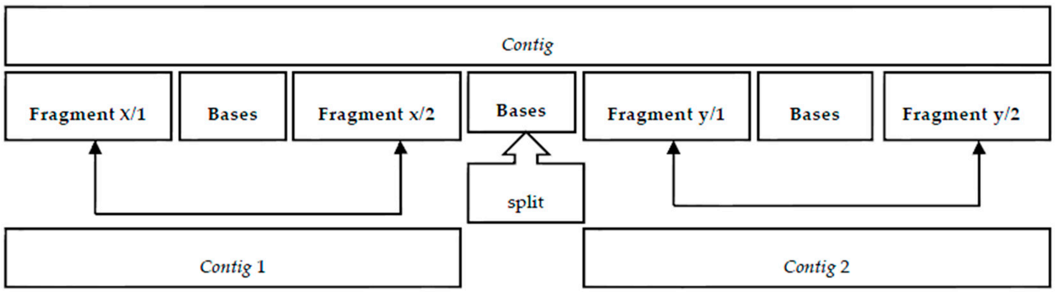


Figure 5. Assembly ensured with the intervals of paired fragments.

3. Algorithms

To resolve the assembly problem, we hereby propose the following sequence (Figure 6):

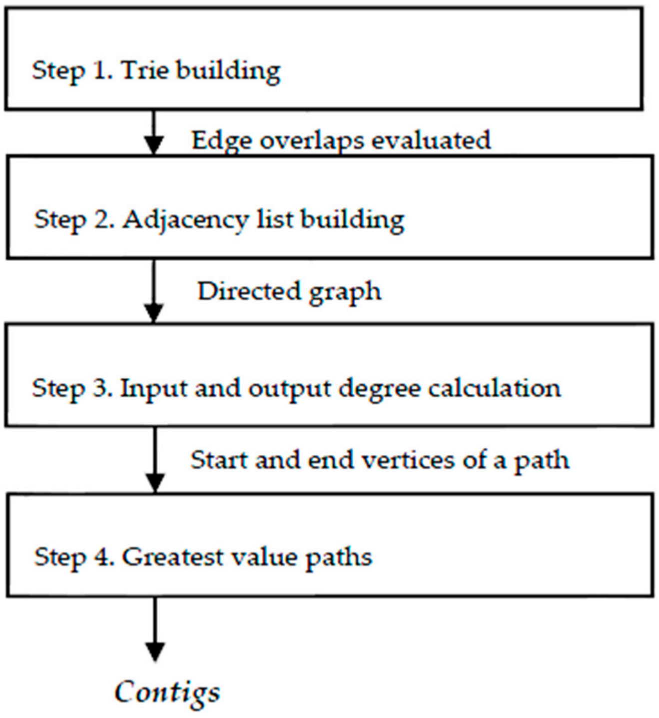

Step 1. Trie building
Edge overlaps evaluated
Step 2. Adjacency list building
Directed graph
Step 3. Input and output degree calculation
Start and end vertices of a path
Step 4. Greatest value paths
Contigs

Figure 6. *Contigs* build sequence.

Step 1. Build a *Trie* (prefix tree) [19] to identify the overlapped fragments, trying to get the greatest value from the overlapping while resolving duplicates, fragments without overlap and repetition bases with the same element.

Step 2. From the list of overlapping fragments that comprise a directed graph, an adjacency list is built [21], with which we can calculate the input and output degrees of each vertex.

Step 3. Using the degree input and output information of the vertices, we could detect those that are a head of a path, with zero input degree, and build the path to those with zero output degree, accumulating the overlapping values of each edge. This accumulated value can change if, when on a new path, there is a value greater than the previous intersection of an intermediate vertex of the path.

Step 4. Finally, walking in reverse, beginning with the last vertex, rebuild the route marked with the greatest overlapping values until a vertex of zero input degree is reached. This path is a *contig*.

3.1. Trie

The *Trie* data structure was developed by Fredking [22]. It is a tree structure that forms prefix texts that are converted into a search index for the new texts. The new texts fit with the prefix and complement the branch of the tree with a suffix. In our problem, the prefixes and suffixes are letters from the alphabet: A, C, G, T.

In the building process, the first node with four locations is empty and ready to receive the four elements A, C, G, T. The first fragment arrives and accommodates depending on the letter value. For example, the first string is AGGTCGA, and it goes on creating new blank nodes. The next one arrives with AGGTTTC, and it accommodates from AGGT; the next fragment is AGGCCTC, and now it accommodates from AGG. Figure 7 shows the resulting tree.

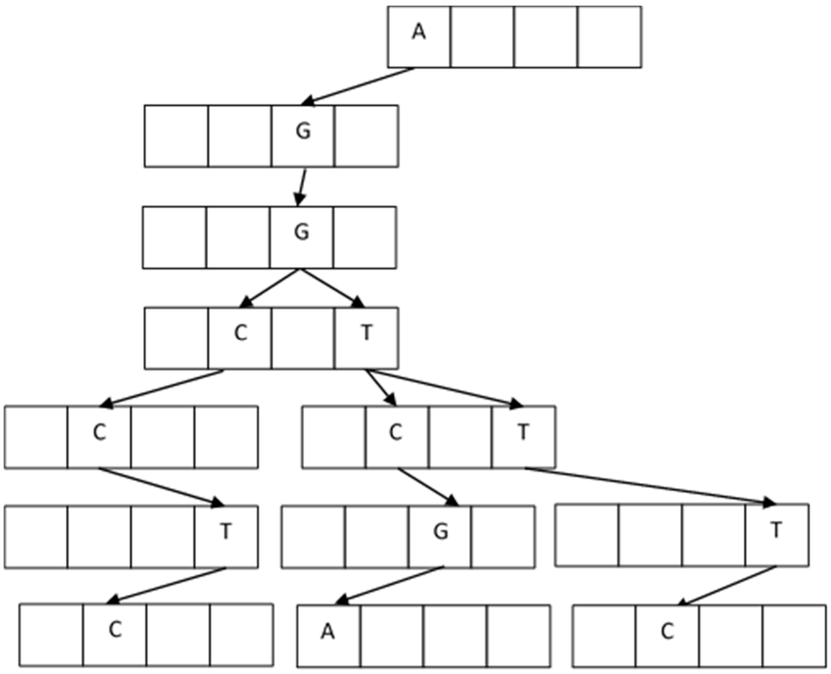


Figure 7. Inserted sequences in the Trie.

The duplicate fragments are easily handled because they do not provide new values to the branches and are eliminated. The construction of the *Trie* is, in the worst case, *O(ln)* [19] when there are no coincidences between the fragments (l is the fragment length and n is the number of fragments). In our data problem, the number of coincidences is large. Algorithm 1 describes the process of building the *Trie*.

Algorithm 1. Trie construction.

```
1.      For each fragment
        1.1     For each letter x from the fragment
                1.1.1   If box node(x) is free:
                        1.1.1.1 Take up the box and create new empty node
                1.1.2   Else
                        1.1.2.1 Go to next node
```

Algorithm 2 shows the search for the overlapped fragments. In the worst case, it is *O(l)* for a single fragment, where *l* is the length of the fragment, because, at that point, it has completely walked through the branch of the tree.

Algorithm 2. Searching a fragment in the Trie.

```
1.    For each fragment (l)
      1.1    For each fragment ($l \leftarrow l$-1 until overlap limit value)
             1.1.1    For each fragment-letter vs. Trie-node-letter
                      1.1.1.a If equal: continue to next fragment-letter and node
                      1.1.1.b If empty box: finish cycle 1.1
      1.2    If end of fragment:
             1.2.1    Drain the branch
             1.2.2    Identify both fragments and create an edge (from, to, overlap)
```

3.2. Adjacency List

This data structure is built by taking the list of edges. The list is a conventional data structure for each fragment.

3.3. Contig Assembly

In this step, the longest path will be sought, starting with all the vertices with zero input degree, these being the potential starting points of a path that could eventually become a *contig*. The walkthrough is carried out until a vertex with zero output degree is found, while accumulating the overlapping values. Once a path is finished, the walkthrough shifts to the next vertex with zero input degree and the process is repeated. If a node that has been used in another path is detected, the process will evaluate the greatest value and will leave the greater value as the result. Once the complete graph has been processed, each starting node is a *contig*. Algorithm 3 shows the process used to obtain the greatest weight of a path.

Algorithm 3. *Contigs* assembly.

```
1.    For all elements with Din = 0
      1.1    Dout minus 1
      1.2    For each adjacency element until Dout = 0
             1.2.1    Accumulate weight
      1.3    If accumulated weight > previous weight
             1.3.1    Label maximum path
```

While assembling the branch, a content counter is generated for the column; at the end of the assembly process, the consensus is reviewed with the counter, and it is possible to remove the sections not in compliance with the predefined value.

In the FASTA file containing the original fragments [18], the paired values are identified as "/1" and "/2"; these identifiers will be used to confirm the *contig* by searching the interval inside the *contig*. Algorithm 4 describes how the paired fragments confirm intervals.

Algorithm 4. Confirm intervals.

```
1.    Interval-counter ← 0 (counts the intervals into a contig).
2.    For each fragment in the contig
      2.1    If type "/1", search type "/2" in the contig
             2.1.1    If Found type "/2" in the contig
                      2.1.1.1 Yes: interval counter = interval counter+1
3.    For each fragment in the contig
      3.1    If interval counter = 0
             3.1.1    If fragment in in-between, cut of the contig
             3.1.2    If fragment in extremes, drop the section
```

Figure 8a shows a case of a *contig* that demonstrates continuity between the first confirmation interval and the subsequent interval. If there is no continuity between the confirmation intervals, then there is a split point, as shown in Figure 8b. This split point will produce two *contigs*. The elements at the extremes are also removed because there is no confirmation interval.

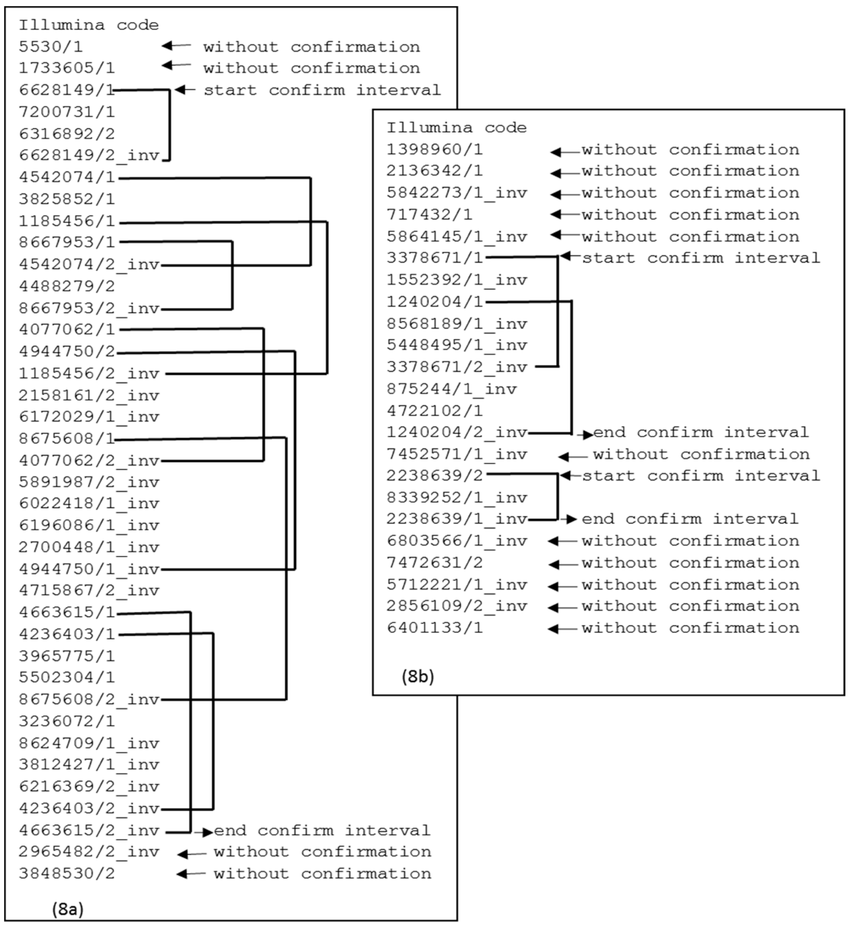

Illumina code
5530/1 without confirmation
1733605/1 without confirmation
6628149/1 start confirm interval
7200731/1
6316892/2
6628149/2_inv
4542074/1
3825852/1
1185456/1
8667953/1
4542074/2_inv
4488279/2
8667953/2_inv
4077062/1
4944750/2
1185456/2_inv
2158161/2_inv
6172029/1_inv
8675608/1
4077062/2_inv
5891987/2_inv
6022418/1_inv
6196086/1_inv
2700448/1_inv
4944750/1_inv
4715867/2_inv
4663615/1
4236403/1
3965775/1
5502304/1
8675608/2_inv
3236072/1
8624709/1_inv
3812427/1_inv
6216369/2_inv
4236403/2_inv
4663615/2_inv end confirm interval
2965482/2_inv without confirmation
3848530/2 without confirmation
(8a)
Illumina code
1398960/1 without confirmation
2136342/1 without confirmation
5842273/1_inv without confirmation
717432/1 without confirmation
5864145/1_inv without confirmation
3378671/1 start confirm interval
1552392/1_inv
1240204/1
8568189/1_inv
5448495/1_inv
3378671/2_inv
875244/1_inv
4722102/1
1240204/2_inv end confirm interval
7452571/1_inv without confirmation
2238639/2 start confirm interval
8339252/1_inv
2238639/1_inv end confirm interval
6803566/1_inv without confirmation
7472631/2 without confirmation
5712221/1_inv without confirmation
2856109/2_inv without confirmation
6401133/1 without confirmation
(8b)

Figure 8. Examples of confirm intervals. (**8a**) *Contig* with confirm interval joined; (**8b**) *Contig* with confirm intervals separated.

4. Experiments

The application of the algorithms was carried out with the *Staphylococcus aureus* problem, taken from GAGE (Genome Assembly Gold-Standard Evaluations, 2011, [9]). The information consisted of 647,062 fixed-length fragments of 101 bases. The bacterium had a genomic chromosome with 2,903,081 bases, a first plasmid with 27,428 bases and a second plasmid with 3170 bases.

The *Illumina equipment* generates three types of DNA sequences: single record, two records paired in a lineal sample and two records paired in a circular sample. In the paper of Mallén et al. [17], the work was applied to a single record; in this new approach, we have worked with lineal paired records. These paired records contained the information necessary to confirm intervals in the assembled DNA.

The reverse-complementary fragments were generated (inverted sequence exchanging AxT and CxG in reverse order) for both types of records, "/1" and "/2". The correspondence of the pairs is:

Type "/1" paired with the reverse complement of "/2",

Type "/2" paired with the reverse complement of "/1".

The generation of these reverse-complementary pairs generated duplicated *contigs*; they were eliminated from the result.

The *Velvet* release we used was 1.2.10, and we searched for the execution with the best results of N50 [9], varying the value of the *k-mer* [13]. Ultimately, the best value obtained was 31. The *Eden* release we used was 3.13, and, in the same way, we took the best result with different overlap values to obtain the best N50 [9]. The best value was 30 overlapping bases.

Illumina equipment delivers two fragment data files. The first contains the record type "/1" and the second contains the record type "/2". In both *Edena* and *Velvet*, it is possible to define a parameter for how to use these files. During the tests, this was considered.

With our programs (in the C/C++ programming language), we also had different parameter values with respect to overlapping. The best result of N50 [9] was found with 50 bases. In the results, we produced two versions, A and B. In our programs, we merge both data files because, during the initial tests, we found these records in the same file. We deleted the records with "N" values for the bases; this value means that the base is undetermined. In addition, we removed all characters except for the A, C, G, and T values. During the *Trie* process, we removed duplicate fragments, and, finally, we eliminated repetitions of the same base value with more than 15 occurrences. This decision was based on an analysis of the data as a special case and would not be applicable to other organisms.

The A version of our program was designed for paired records; the B version also included both records, but the confirmation interval was applied to the *contigs*. In this version, we also eliminated the *contigs* with a length shorter than 1.5 times the fragment size. *Edena* also carries out this process, but this decision cannot be modified by the user; it is a consistency requirement. This action resulted in a modest improvement in N50 [9]. Table 1 shows the results of the executions in the "Generated" section. To determine how close we were to obtaining total genome reconstruction, we did a search of every *contig* in the original genome with our results, with those of *Edena* and with those of *Velvet*.

The results presented in Table 1 have been obtained from QUAST (v 4.4. CAB: Center for Algorithmic Biotechnology. St. Petersburg, Russia) [23]. The search was also done with Mummer [24]. The last column of Table 1 shows the result from Mummer, representing the *contigs* that have been found completely in the original genome. This result was applied to *Edena*, *Velvet* and our programs.

We performed all program executions two consecutive times to guarantee an execution time independent of computer dependencies such as those from the cache memory, the hard disk or the processor. With the Linux command "time" in terminal mode, we obtained the real time and the user time. We took the real time in all the proofs and the lower of the two times from the first and the second executions.

All of the runs were made on the same computer, employing 64 bit Linux with Ubuntu 14.4 Long Term Support (London, UK), 16 gigabytes of RAM, an Intel (Santa Clara, CA, USA) i5-4460 processor @ 3.2 GHz $\times$ 4 and a hard disk with an EXT4 partition type. The programming language of our programs (versions A and B) was C/C++, with the standard compilation C++ ISO (-std = C++11), and without any optimization. Table 2 shows the execution times for each of the cases tested.

Table 1. Experimental results.

Organism:		***Staphylococcus aureus***					**Genome: 2,903,081 with Two Plasmids**			
		Generated (Statistics without Reference)					**Found in Genome (Statistics with Genome Reference)**			
Program	**Parameter**	***Contigs* Generated**	***Contigs* in the Assembly**	**Total Bases in the Assembly**	**Total Bases in the *Contigs***	**N50 from *Contigs***	**NG50 from *Contigs* Found**	**Total Number of Aligned Bases**	**Genome Fraction %**	***Contigs* Found by Mummer in Genome at 100%**
Velvet	k-mer = 31	1059	693	2,733,950	2,816,990	**6666**	**6216**	2,733,853	93.942%	**67.3%**
Edena	overlap = 30	3038	1670	2,017,901	2,419,142	**1380**	**938**	2,017,901	69.39%	**86.6%**
Version A	overlap = 50	2985	1701	2,234,717	2,619,179	**1539**	**1183**	2,233,545	76.326%	**84.3%**
Version B	overlap = 50 w/confirm interval	20,439	2090	1,455,722	6,016,455	**675**	**500**	1,437,787	25.986%	**93.4%**

Table 2. Execution times.

Program	Step	Execution Time (s)
Velvet	Velveth	23.085
	Velvetg	6.397
Edena	Edena–DR	363.676
	Edena-e	2.051
Version A	FragAPares	69.140
	ListAdy	5.002
Version B	FragAPares	69.140
	ListAdyU	5.834

5. Conclusions

The advantages of our algorithms are described below.

Before applying the confirmation intervals, our N50 [9] was superior to *Edena* but not to *Velvet*. It is clear that *Velvet* uses *de Bruijn* graphs [13], and it contains a rebuilding *contigs* step. The resultant N50 of *Velvet* is better than those of *Edena* and our programs (see Table 3).

Table 3. N50 comparison.

N50 from Generated *Contigs*	
Velvet	6666
Edena	1380
Version A	1539
Version B	675

After applying the confirmation intervals, our N50 [9] for the paired fragments was smaller, but the quality of the *contigs* was greater than those of the other programs, as seen in the percentage of Mummer-located *contigs* (see Table 4).

Table 4. *Contigs* found by Mummer.

Contigs Found by MUMMER in the Genome at 100%	
Velvet	67.3%
Edena	86.6%
Version A	84.3%
Version B	93.4%

The values found by Mummer (see Table 1) do not help *Velvet* (67.3%), and our program yielded the best value (93.4%). If Mummer locates the *contig* completely in the genome, it appears as a 100% value; if Mummer cannot locate the *contig*, it adjusts the difference, splitting or tolerating certain errors and trying to relocate the pieces, reporting these as a percent value less than 100%. These values are not included in Table 1.

Regarding the execution times (see Table 2), in the first step, *Edena* took 6 min to calculate the overlap, and it took the longest time for the samples. In our version, an equivalent process took approximately 1 min, so we concluded that our process is much faster than *Edena*, though that is not the case when compared to *Velvet*. Our program requested approximately 4 gigabytes of RAM to manage the *Trie* [22]. Meanwhile, as the *Trie* grows, it becomes asymptotic; the construction of the *Trie* is rapid, with times on the order of five seconds.

A potential future study could search the "/2" type records outside the *contig* with a "/1" type record without its matching pair. These might generate a *contig* with a greater confirmation interval and, consequently, a better N50 value [9].

To obtain an improvement in the execution time, it would be advisable to eliminate the transitive edges of the graph. With this action, the data volume would be reduced significantly.

The identification of the sequences of the *contigs* could represent valuable information for molecular biologists, and these could probably be obtained by looking for paired fragments that span multiple *contigs*. The records with undetermined values ("N") could probably also contribute some information about the *contig* ordering.

Finally, the scaffolding process (*contigs* assembly) could generate important results for biologists, including a likely new step of detecting overlapping within *contigs* and giving rise to some adjustments in the quality assurance of the assembly.

Acknowledgments: This work has been developed with the support of Universidad Iberoamericana.

Author Contributions: J. Emilio Quiroz-Ibarra developed the algorithms and programs and conceived the idea for the study. Guillermo M. Mallén-Fullerton originated the idea and developed the algorithms. Guillermo Fernández-Anaya provided the formal verification of the algorithms, the complexity analysis and observations to improve them. All authors approved the manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Watson, J.D.; Crick, F.H.C. A Structure for Deoxyribose Nucleic Acid. *Nature* **1953**, *171*, 737–738. [CrossRef] [PubMed]
2. Sanger, F.; Coulson, F. DNA sequencing with chain-terminating inhibitors. *Proc. Natl. Acad. Sci. USA* **1977**, *74*, 5463–5467. [CrossRef] [PubMed]
3. Gingold, E. An introduction to genetic engineering. In *Molecular Biology and Biotechnology*; Royal Society of Chemistry: London, UK, 1988; pp. 25–46.
4. Myers, E.W., Jr. A history of DNA sequence assembly. *IT Inf. Technol.* **2016**, *58*, 126–132. [CrossRef]
5. Sanger, F.; Coulson, A.R.; Hong, G.F.; Hill, D.F.; Petersen, G.B. Nucleotide sequence of bacteriophage λ DNA. *J. Mol. Boil.* **1982**, *162*, 729–773. [CrossRef]
6. Myers, E. Toward Simplifying and Accurately. *J. Comput. Biol.* **1995**, *2*, 275–290. [CrossRef] [PubMed]
7. Staden, R. A strategy of DNA sequencing employing computer programs. *Nucleic Acids Res.* **1979**, *6*, 2601–2610. [CrossRef] [PubMed]
8. NCBI Genome&Maps, National Center for Biotechnology Information. Available online: https://www.ncbi.nlm.nih.gov/guide/genomes-maps/ (accessed on 21 November 2016).
9. Salzberg, S.L.; Phillippy, A.M.; Zimin, A.; Puiu, D.; Magoc, T.; Koren, S.; Treangen, T.J.; Schatz, M.C.; Delcher, A.L.; Roberts, M.; et al. A critical evaluation of genome assemblies and assembly algorithms. *Genome Res.* **2012**, *22*, 557–567. [CrossRef] [PubMed]
10. Tammi, M. *The Principles of Shotgun Sequencing and Automated Fragment Assembly*; Center for Genomics and Bioinformatics: Stockholm, Sweden, 2003.
11. Hernandez, D.; Tewhey, R.; Veyrieras, J.B.; Farinelli, L.; Østerås, M.; François, P.; Schrenzel, J. De novo finished 2.8 Mbp *Staphylococcus aureus* genome assembly from 100 bp short and long range paired-end reads. *Bioinformatics* **2014**, *30*, 40–49. [CrossRef] [PubMed]
12. Bang-Jensen, J.; Gutin, G.; Yeo, A. When the greedy algorithm fails. *Discret. Optim.* **2004**, *1*, 121–127. [CrossRef]
13. Compeau, P.E.; Pevzner, P.A.; Tesler, G. How to apply de Bruijn graphs to genome assembly. *Nat. Biotechnol.* **2011**, *29*, 987–991. [CrossRef] [PubMed]
14. Zerbino, D.R.; Birney, A.E. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res.* **2008**, *18*, 821–829. [CrossRef] [PubMed]
15. Burks, C.; Forrest, S.; Parsons, R. Genetic Algorithms for DNA Sequence Assembly. *ISMB* **1993**, *1*, 310–318.
16. Mallén-Fullerton, G.M.; Fernandez-Anaya, G. DNA fragment assembly using optimization. In Proceedings of the IEEE Congress on Evolutionary Computation, Cancun, Mexico, 20–23 June 2013; pp. 1570–1577.
17. Mallén-Fullerton, G.M.; Quiroz-Ibarra, J.E.; Miranda, A.; Fernández-Anaya, G. Modified Classical Graph Algorithms for the DNA Fragment Assembly Problem. *Algorithms* **2015**, *8*, 754–773. [CrossRef]

18. Fasta_Formats, DNA Sequence Formats, Genomatix GmbH, 2016. Available online: https://www.genomatix.de/online_help/help/sequence_formats.html#FASTA (accessed on 30 October 2016).
19. Ukkonen, E. On-line construction of suffix trees. *Algorithmica* **1995**, *14*, 249–260. [CrossRef]
20. Gusfield, D. Linear-Time Construction on sufixx trees. In *Algorithms on Strings, Trees and Sequences*; Cambridge University Press: Melbourn, UK, 1997; pp. 94–119.
21. Black, P. Adjacency List, Dictionary of Algorithms and Data Structures, 2008. Available online: https://xlinux.nist.gov/dads/HTML/adjacencyListRep.html (accessed on 4 November 2016).
22. Fredkin, E. Trie memory. *Commun. ACM* **1960**, *3*, 490–499. [CrossRef]
23. Gurevich, A.; Saveliev, V.; Vyahhi, N.; Tesler, G. QUAST: Quality assessment tool for genome assemblies. *Bioinformatics* **2013**, *29*, 1072–1075. [CrossRef] [PubMed]
24. Kurtz, S.; Phillippy, A.; Delcher, A.L.; Smoot, M.; Shumway, M.; Antonescu, C.; Salzberg, S.L. Versatile and open software for comparing large genomes. *Genome Biol.* **2004**, *5*, R12. [CrossRef] [PubMed]