Plant Tissue Cult. & Biotech. **26**(1): 105-121, 2016 (June)

PTC&B

SeqDev: An Algorithm for Constructing Genetic Elements Using Comparative Assembly

Tasnim Rahman, Hasnain Heickal[1]*, Shamira Tabrejee[2], Md. Miraj Kobad Chowdhury[2], Sheikh Muhammad Sarwar and Mohammad Shoyaib

Institute of Information Technology, University of Dhaka, Dhaka-1000, Bangladesh

Key words: Genome sequences, Assembler, Genetic elements, Bioinformatics

Abstract

With the availability of recent next generation sequencing technologies and their low cost, genomes of different organisms are being sequenced frequently. Therefore, quick assembly of genome, transcriptome, and target contigs from the raw data generated through the sequencing technologies has become necessary for better understanding of different biological systems. This article proposes an algorithm, namely SeqDev (Sequence Developer) for constructing contigs from raw reads using reference sequences. For this, we considered a weighted frequency-based consensus mechanism named Blast*Assemb* for primary construction of a sequence with gaps. Then, we adopted suffix array and proposed a gap filling search (GFS) algorithm for searching the missing sequences in the primary construct. For evaluating our algorithm, we have chosen Pokkali (rice) raw genome and Japonica (rice) as our reference data. Experimental results demonstrated that our proposed algorithm accurately constructs promoter sequences of Pokkali from its raw genome data. These constructed promoter sequences were 93 - 100% identical with the reference and also aligned with 96 - 100% of corresponding reference sequences with eValue ranging from 0.0 - $2e^{-14}$. All these results indicated that our proposed method could be a potential algorithm to construct target contigs from raw sequences with the help of reference sequences. Further wet lab validation with specific Pokkali promoter sequence will boost this method as a robust algorithm for target contig assembly.

*Author for correspondence: <hasnain@cse.univdhaka.edu>. [1]Department of Computer Science and Engineering, University of Dhaka, Dhaka-1000, Bangladesh. [2]Department of Genetic Engineering and Biotechnology, University of Dhaka, Dhaka-1000, Bangladesh.

Introduction

Availability of sequences of entire genomes and software for their analysis has opened a new era in the field of molecular, comparative and evolutionary biology. With ongoing advancements, next generation sequencing (NGS) technologies are producing raw genome sequences at a revolutionary speed and with more accuracy than ever before. Such sequencing technologies are creating enormous data through whole-genome shotgun sequencing (WGS) method. In WGS approach, a whole genome is broken down into a large number of very small random fragments. Then, all of these fragments are sequenced, where the sequence of an individual fragment is called a "read". Raw genome sequence is a collection of all of these reads, from which the whole genome sequence is constructed. Therefore, it has become a fundamental goal to assemble genome or transcriptome from such a large number of reads. A computational approach called "genome assembly" is used for such a construction of genome from all the reads. An assembly is defined by a hierarchical data structure that maps the raw sequence to a putative construction of the genome (Miller et al. 2010). Simply, an assembler puts all the reads together for constructing the whole genome that would be practical for interpreting the functions of the organism.

Generally genome assembly uses three types of approaches and they are: (i) *de novo* assembly, (ii) comparative assembly and (iii) a combination of *de novo* and comparative assembly (Quigley 2014). *De novo* approach focuses on constructing genome sequences from a set of sequence reads without a previously sequenced reference genome of an organism. This approach uses overlap-layout-consensus (OLC), de Bruijn graph (DBG) and greedy algorithm for constructing genome sequence. *De novo* genome assembly can be difficult, which falls within a class of problems, NP-hard, for which no efficient computational solution is known (Myers 1995, Medvedev et al. 2007). This is because NGS sequencing technology is now producing very short reads, as short as 35 bp (Pop and Salzberg 2008). As a result, *de novo* assemblies of short read data are highly fragmented (Simpson et al. 2009, Farrer et al. 2009). A number of tools have been developed on the basis of *de novo* approach for genome assembly such as: SHARCGS (Dohm et al. 2007), VCAKE (Jeck et al. 2007), VELVET (Zerbino and Birney 2008) SOAPdenovo (Xie et al. 2014), MaSuRCA (Zimin et al. 2013), CABOG (Miller et al. 2008), EULERSR (Chaisson and Pavzner 2008), ABySS (Simpson et al. 2009) and ALLPATHS (Butler et al. 2008).

Unlike the *de novo* approach, there is a prior view of genome sequence in comparative genome assembly. Comparative genome assembly constructs a genome sequence by mapping it into a sequence of a closely related organism as a guide during the assembly process and the mapped information is used for

inferring the new genome sequence. Several works have been done on the basis of comparative assembly approach and strategies belonging to this category include, AMOS (Pop et al. 2004), PGA (Zhao et al. 2008), MAQ (Li et al. 2008), Gene boosted assembly (GBA) (Salzberg et al. 2008), etc. In general, speed and accuracy are the common limitations of these methods. Specially, short reads may create problems in this regard (Homer et al. 2009).

Some assembly algorithms have been developed (Vezzi et al. 2011, Schneeberger et al. 2011, Wang et al. 2014, Nishito et al. 2010), which lie between these two models. It adopts a *de novo* approach relying on the assistance of the reference genome and loosely adopts the alignment-overlap-layout-consensus scheme.

Here, we have proposed an algorithm, named SeqDev (Sequence Developer) for constructing genetic elements such as promoter, enhancer etc., from a raw genome sequence with the help of a set of closely related reference sequences. This algorithm combines the capability of BLAST (Basic local alignment search tool) (Altschul et al. 1990, Zhao and Chu 2014, Oehmen and Baxter 2013), which is a well known tool in the bioinformatics community, as well as a gap filling search (GFS) algorithm (Rahman et al. 2014). To validate this proposed method, we have constructed a set of promoter using the raw genome sequence of Pokkali rice variety with the help of promoter sequences of Japonica rice variety as reference sequences. The resulting promoter sequences were then validated by different promoter identifier programs.

Proposed Methodology

The proposed method has two major parts: (1) Blast*Assemb* and (2) Gap Filling Search (GFS). Part 1 roughly constructs the desired sequence with the help of reference sequence. Genetic elements constructed from part 1 may have some missing characters in them. Thus, we extended it to part 2 for the construction of the complete sequence without any 'N' by adopting suffix array and proposing a GFS algorithm.

Blast*Assemb*

Blast*Assemb* follows three steps to produce the genetic elements: (1) Run BLAST and find the matches between a reference sequence and the raw genome, where we extract the matched part (subsequence) from the raw genome sequences, (2) make extracted matched subsequences and the reference sequence length the same by padding 'N' as required where 'N' represents any of A, C, G, or T, and (3) construct consensus using weighted frequency of the matched sequences. We start with retrieving sequences from FASTA-formatted raw genome, and the matched positions between each raw and reference sequence. This information is

retrieved by using 'local BLAST'. From now on the term 'local BLAST' will be used to mean the use of BLAST where only reference sequence and raw genome sequences are used. The whole process of the proposed mechanism is described in Algorithm 1.

Algorithm 1 Constructing Sequence from Raw Genome

Input: refGenome, RawGenome
Output: consGenSequence

```
1:  Begin
2:  PaddedSeqList := Ø
3:  eValueList := Ø
4:  BlasedGenomeList ←BLAST(refGenome, RawGenome)
5:  for each id € BlasedGenomeList do
6:  rawGenomeSeq ← rawGenomeid
7:  rawStart ← rawStartid
8:  rawEnd ← rawEndid
9:  refStart ← refStartid
10: refEnd ← refEndid
11: E-value ← E-value id
12: subsequence ← FindMatchedSubsequence
    (rawGenomeSeq, rawStart, rawEnd)
13: paddedSubseq ← MakePaddedSeq (subsequence, ref-Start, refEnd)
14: PaddedSeqList ← PaddedSeqList ← paddedSubseq
15: E-value List ← E-value List ← E-value
16: end for
17: consGenSequence ← ConsensusSeq(PaddedSeqList, eValueList)
18: End
```

BLAST

BLAST algorithm is used to align between query sequences with subject sequence (Altschul et al. 1990). It is a heuristic based searching algorithm, whose aim is to find the fragment of query sequence that is matched in the subject database. The output of BLAST provides a file containing information about sequence ID, positions of reference sequences and raw sequence fragments where they matched.

Finding matched subsequence

From the local BLAST output file, procedure 1 takes raw genome IDs, reference sequence IDs, *rawStart* and *rawEnd* positions and expect value (E-value). The matched subsequence fragment is then extracted from the raw sequence.

Procedure 1 Find Matched Subsequence

```
Input: raw Genome Seq, raw Start, raw End
Output: subsequence
1:    Begin
2:    subsequence :=Ø
3:    for i = raw Start : raw End do
4:    subsequence += raw Genome Seq[i]
5:    end for
6:    End
```

Making padded sequences

All the extracted matched parts are padded with character *'N'*. Then the padded sequences are used to calculate consensus sequence. Procedure 2 takes the matched subsequences and their positions in reference sequence (*refStart* and *refEnd*) as input. The remaining parts of the sequence are padded up with *'N'*, where *'N'* is the placeholder for any nucleotides.

Procedure 2 Make Padded Sequence

```
Input: subsequence, refStart, refEnd
Output: paddedSeq
1:    Begin
2:    for i=0:n do
3:    paddedSeq += 'N'
4:    end for
5:    for i=refStart:refEnd do
6:    paddedSeq[i] = subsequence[i-refStart]
7:    end for
8:    End
```

Constructing consensus sequence

Once all the *padded sequences* are constructed, the final step of Blast*Assemb* is to construct the consensus sequence. *ConsensusSeq*, as described in Algorithm 1, takes a list of padded sequence and the corresponding E-value. The consensus is calculated by weighted sum of nucleotides. The weight is given based on the expectation value where each nucleotide of a particular position is summed over the negative logarithm of expected values, $\sum_{i=1}^{r} -\log(eValue)$, where r is the number of a specific nucleotide for calculating the consensus, to generate the weighted consensus. Thus nucleotide having the maximum E-value is taken as consensus. Furthermore, there might be a case where the weighted frequencies of two nucleotides are same. In that case we will consider both the nucleotides as consensus. The consensus sequences, generated by 'Blast*Assemb*', may have series of *'N'*s in the consensus sequences. For performing the most realistic biological analysis, it is preferred that, the sequences should be as complete as possible which is addressed in the second part of our proposal.

Gap filling search

The resulting consensus sequences from Blast*Assemb* algorithm are not fully constructed. There are a series of 'N' in the constructed consensus sequences. Our next algorithm uses the proposed "Gap filling search" which is a suffix array (Manber and Myers 1993) and a binary search-based distributed algorithm. This algorithm replaces the series of 'N's from the previously constructed consensus sequences from "Blast*Assemb*". The algorithm initiates with retrieving the FASTA formatted raw genome sequence file. These raw reads are sorted using suffix array (Shrestha et al. 2014). This suffix array uses counting sort to sort intermediate substrings. This way it achieves faster performance. Later, binary search is used to find candidate raw reads from these sorted raw reads sequences. Using a recursive backtracking technique, we are able to find appropriate raw genome sequences to fill up "N"s in the consensus sequences generated in "BlastAssemb".

Suffix array construction

Recently, suffix array and its variants of text-indexing data structures have become essential in the field of bioinformatics (Shrestha et al. 2014). Suffix array has been used for prefix and suffix matches in genome assembly (Ilie 2011).

Algorithm 2 takes all the raw reads sequences form the file and sorts them alphabetically using suffix array. First all the genomes are concatenated to produce one large string. Suffix array is applied to this string to sort the indices alphabetically, as each index represents the suffix that starts there. Only the

indices of the starting of a raw reads are saved. Also the suffix size is limited to cover only the size of each of the raw reads.

Algorithm 2 GFS (Gap filling search)

Input: Raw Genome Sequence RG
Consensus Sequence CS
Output: Consensus without "N" FCCS

```
1:  Begin
2:  RGList ← SuffixArray (RG)
3:  FCCS ← Replace Missing Characters (RGList, CS)
```

Searching for missing nucleotide characters

Since suffix array provides the alphabetical orders of all the raw reads, a typical binary search is used to search appropriate candidates for replacing the 'N's. The procedure is given below:

i. We have taken the consensus sequences generated by "Blas*Assemb*" and eliminated those sequences which have 'N' in the starting or ending of the sequences.
ii. For each of the remaining sequences, we identified the sections which contain 'N's.
iii. We took *Starting* and *Ending* position of each of those sections.
iv. L (Length of consensus sequence upstream or downstream of 'N's) characters before the *Start* position of 'N' as Prefix and L character after the 'N's *End* position as Suffix were taken as sub sequence for input in the searching algorithm. Value of L was between 5 and 10.

Algorithm 3 performs a binary search with the prefix as input on the sorted raw reads. If there are raw reads whose prefix matches with the input prefix, the whole raw reads followed by the match is taken. The procedure is iterated until the given *Start*, *End* range is totally filled up. For each iteration, *Start* position and input prefixes are changed. The changes are made in such a way that the end position of last raw reads becomes the new *Start* position and the last L characters from the resulting string is taken as new prefix unless the *Start*, *End* range is filled up. Once the range is filled up, the input suffixes are used to check the matching rate with the resulting sequence. This process is performed for each of the matches in each file.

Algorithm 3 Replacement of Missing Characters

Input: Suffix Array, Prefix, START, END
Output: Sequence

```
Begin
Sequence ← Ø
if START≤ END then
StringList ← Binary Search (Suffix Array, Prefix)
for each each String S € String list do do
Concat S with Sequence
newPrefix = Prefix.length long Suffix of S
new Start = START + S.legnth - prefix. length
new S = Replace Missing Characters (Suffix Array, new Prefix, new Start, END)
Concat new S with Sequence
end for
end if
End
```

Reconstructing consensus sequences

Once the binary search provides all possible characters regarding the *Start*, *End* position of each sequence, the final step is to reconstruct consensus sequence from them. The 'N's of each previously constructed consensus sequences are replaced with the searched nucleotide characters. Then, the consensus sequences are calculated by counting the frequencies of each nucleotide in each position to obtain the newly constructed consensus sequence without any 'N's.

Experimental Setup

In this section we first describe the data explanation, secondly validation process of the constructed genetic elements and finally the results with discussion will be presented in this section.

Data Description

Cotsaftis et al. had done an analysis of root gene expression of salt-tolerant genotypes FL478, Pokkali and IR63, and salt-sensitive genotype IR29 under control and salinity-stressed conditions during vegetative growth. They provided a data set, from which they wanted to identify those genes associated with salt tolerance. We took the Probset ids from the data sets and converted the ID's into gene names from the affymetrix website (http://www.affymetrix.

com/estore/). Later, the gene names were used for downloading the corresponding promoter sequences from Rice Annotation Project Database (http://rapdb.dna.affrc.go.jp), which archives sequences of *O. sativa* Japonica variety. Each sequence was 1000 bp upstream from the transcription initiation site. These sequences were taken as reference promoter sequences. We also collected the raw data of Pokkali, which is a salt tolerant variety, from GigaDb, 3000 Rice Genome Project (http://gigadb.org/dataset/200001). The raw genome sequence contained about 40 million reads and each of the read was of 83 base pairs in length.

Validation Protocols

The newly assembled sequence using our method is confirmed for its rice origin by performing BLAST. Here, the term 'BLAST' suggested any alignment between a sequence and all tested sequences deposited to NCBI (http://www.ncbi.nlm.nih.gov/) unless otherwise specified. From the BLAST result, the mostly matched sequence was considered. The corresponding sequence ID, description, match score, coverage and eValue were recorded for each considered match. Multiple sequences were considered in cases where both *Oryza sativa* Indica as well as *O. sativa* Japonica scored closely or if there was a considerable match with any rice genes. If the newly assembled sequences showed more than 65% coverage, they were considered as a constructed genetic element (a promoter in this case) of rice.

To test whether a sequence is promoter or not there exist several softwares (Ma et al. 2013). Here, we considered two well known programs developed to identify eukaryotic promoters: 1) neural network promoter prediction or NNPP (http://www.fruitfly.org/seq_tools/promoter.html) and 2) Promoter 2.0 (http://www.cbs.dtu.dk/services/Promoter/). NNPP uses neural network which is trained to identify promoter. The cut-off value for a predicted promoter was considered as 0.8 in this case. Promoter 2.0 combines neural network and genetic algorithm for identifying the promoter elements with 0.5 as a cut-off value. A sequence was considered to be promoter only if both of these programs successfully identified putative promoter regions. If only one program was successful to find such elements, then we went for a third level test.

The third level test uses PlantPan (plantpan.mbc.nctu.edu.tw/) which determines the existence of transcription factor binding site. The sequence was considered as a promoter if there were more than 8 transcription factor binding sites. All the constructed promoters were validated using aforementioned methods.

Results and Discussion

We have divided our test into two parts. In the first part, using the 15 reference promoter sequences of Japonica, we successfully assembled 15 new promoter sequences from the Pokkali raw genome sequences using our Algorithm 1.

Next, these 15 newly assembled sequences were then used as a query sequence to perform BLAST to find out their best matches. Remarkably, all the 15 new assembly aligned the best with *O. sativa* Japonica variety. At least one of them matched with a known Pokkali sequence and 5 of them matched best with Indica variety (Table 1). The alignment covered 53% - 100% of the assembled sequences with eValue ranging from 0.0 - $7e^{-62}$. These findings indicated that the newly assembled sequences were comparable with reference rice *O. sativa* Japonica genome sequences and homologous to this known genome. Thereby, it can be concluded that we have moderately assembled 15 different kilomers as a part of Pokkali genome from the partial raw sequences.

We then verified these sequences using different promoter prediction software as described in section 3B. We compared the results of these two programs and found that 8 assembled sequences were identified as promoters by both program. Rests of the 7 sequences were identified as a promoter by either one of the programs (Table 2). These data suggested that all the assembled sequences are possibly promoter sequences.

The results of transcription factor binding sites (TFBS) are summarized in Table 3. The assembled sequences showed roughly 9 -17 TFBS, except for one promoter. In cases where one promoter identifier program fails to identify a promoter region (for sequences P1, P3, P4, P6, P8, P10 and P14), we found at least 9 TFBS. This suggested that the assembled sequences were in fact promoters like their corresponding reference sequences.

For reconstructing the consensus sequences without 'N', various prefix lengths can be used for searching missing characters. Here we used L= 10 to 5 for searching missing characters, where 'L' indicates the length of consensus sequence in base pairs upstream or downstream of 'N'. We took the previously assembled 15 promoter sequences containing 'N's in them. We prepared the sequences and identified 11 promoters for searching the missing characters. We successfully reconstructed 8 promoter sequences without any 'N' by this method.

We identified the reconstructed promoter ID's and did an analysis to match the ID to most of the prefix lengths. This helped to identify whether the reconstructed complete sequences are promoter sequences more accurately. The analysis showed that 8 promoter sequences were reconstructed from most of the lengths. The results are given in Table 4.

Table 1. List of the best matched sequences from BLAST with individual construct as a query sequence.

Query promoter	Subject sequence	Sequence ID	Query location	Subject location	Similarity (%)
P1	Indica (Pokkali) HKT1.5 (HKT1.5) gene	gbjJQ695812.1j	74-1000	1766-2694	92.00%
	Indica chromosome 1 clone K0125H04	dbjjAP012695.1j	74-1000	152306-151378	92.00%
	Indica (Pokkali) H^+/K^+ transporter gene, promoter region and 5′ UTR	gbjFJ716716.1j	74-1000	698-1626	92.00%
P2	Indica chromosome 1 clone K0282G05	dbjjAP012790.1j	143-826	66122-65439	68.00%
	Japonica chromosome 1, PAC clone P0410E01	dbjjAP002866.1j	143-826	96218-95535	68.00%
P3	Indica chromosome 1 clone K0475H03	dbjjAP012877.1j	459-995	56424-56961	53.00%
	Japonica chromosome 1, PAC clone P0439E11	dbjjAP003315.1j	459-995	89200-88936	53.00%
P4	Indica chromosome 1 clone K0238C07	dbjjAP012619.1j	218-621	17683-18118	40.00%
	Japonica chromosome 1, PAC clone P0491F11	dbjjAP004669.3j	6-88; 97-122; 218-825	28293-28375; 28385-28410; 28505-29112	71.00%
P5	Indica chromosome 1 clone K0187C03	dbjjAP012940.1j	15-959	61404-62354	94.00%
	Japonica OsPCL1 gene	dbjjAB206578.1j	15-959	708-1652	94.00%
	Japonica chromosome 1, PAC clone P0518C01	dbjjAP003277.2j	15-959	61649-62593	94.00%
P6	Japonica chromosome 2, BAC clone OJ1717-A09	dbjjAP004071.3j	1-73; 133-926	146527-146599; 145674-146467	86.00%
P7	Japonica chromosome 3 clone OSJNBa0042J17	gbjAC125471.3j	1-1000	2618-3617	100.00%
	Japonica (Nipponbare) chromosome 3 clone OSJNBa0032G08	gbjAC079633.9j	1-1000	29172-28172	100.00%

(Contd.)

Table Contd.

Query promoter	Subject sequence	Sequence ID	Query location	Subject location	Similarity (%)
P7	Japonica chromosome 3 clone OSJNBa0042J17	gbjAC125471.3j	1-1000	2618-3617	100.00%
	Japonica (Nipponbare) chromosome 3 clone OSJNBa0032G08	gbjAC079633.9j	1-1000	29172-28172	100.00%
P8	Japonica chromosome 3 clone OJ1012B02	gbjAC135205.4j	1-1000	107302-106303	100.00%
	Japonica chromosome 3 clone OJ1364E02	gbjAC135208.3j	1-1000	6804-5805	100.00%
P9	Japonica chromosome 3 BAC clone OSJNBa0010E04	gbjAC096687.5j	1-1000	94347-95346	100.00%
	Japonica chromosome 3 BAC clone OJ1111 B11	gbjAC091247.4j	1-1000	17874-18873	100.00%
P10	Japonica chromosome 5 clone OSJNBa0036C12	gbjAC121363.2j	49-159; 328-906	47419-47529; 47698-48276	68.00%
P11	Japonica chromosome 6 clone P0535G04	dbjjAP000399.1j	536-1000; 3-441	35376-35840; 35935-36372	90.00%
P12	Japonica chromosome 7 PAC clone P0681F05	dbjjAP004674.2j	1-376; 515-609; 676-1000	22752-23127; 23266-23360; 23427-23751	79.00%
	Indica DR8 gene	gbjDQ176424.1j	515-609; 676-1000	66-160; 227-551	41.00%
P13	Japonica chromosome 7 PAC clone P0681F05	dbjjAP004674.2j	1-376; 515-609; 676-1000	22752-23127; 23266-23360; 23427-23751	79.00%
P14	Japonica chromosome 7 BAC clone OJ1121-A05	dbjjAP005908.3j	1-230; 322-966	119199-119428; 119520-120194	87.00%
P15	Japonica chromosome 11 clone OSJNBb0005H02	gbjAC134053.4j	278-1000	57508-56786	72.00%

Table 2. List of identified promoters and their positions in the assembled sequences.

Promoter	NNPP (predicted promoter position (score))	Promoter 2.0 (predicted promoter position (score), likelihood)
P1	181-231 (0.93); 252-302 (0.97); 894-944 (1.0)	Not Detected
P2	8-58 (0.97); 383-433 (0.98); 443-493 (0.99)	600 (1.081), highly likely
P3	332-382 (0.88)	Not Detected
P4	79-129 (0.99); 751-801 (0.83)	Not Detected
P5	463-513 (0.84); 797-847 (0.98); 847-897 (0.99)	600 (1.128), highly likely
P6	242-292 (0.97); 403-453 (0.82); 747-797 (0.95)	Not Detected
P7	27-77 (0.90); 35-85 (0.95); 183-233 (0.97)	800 (1.209), highly likely
P8	312-362 (0.85); 337-387 (0.80); 919-969 (0.89)	Not Detected
P9	886-936 (0.99); 950-1000 (0.84)	500 (0.640), Marginal
P10	359-409 (0.96)	Not Detected
P11	757-807 (0.93)	300 (0.620), Marginal
P12	370-420 (0.82); 477-527 (0.85); 683-733 (0.94)	800 (0.676), Marginal
P13	370-420 (0.82); 477-527 (0.85); 683-733 (0.94)	800 (0.676), Marginal
P14	Not Detected	800 (0.619), Marginal
P15	423-473 (0.93)	700 (0.639), Marginal

Table 3. Number of identified transcription factor binding sites in the assembled sequences.

Promoter	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15
Number	16	11	9	12	16	16	17	10	14	12	12	14	14	9	5

Next, all of the 8 reconstructed sequences for each length were then used as a query sequence to perform BLAST to find out their best matches. All the reconstructed sequences aligned the best with *O. sativa* Japonica variety. The alignment covered 94 - 100% of the reconstructed sequences with eValue 0.0 and also with 97 - 99% identity (Table 4). When these reconstructed were validated using promoter prediction software, they showed similar results as described in Tables 2 and 3.

Later, we constructed promoter sequences of 4 sets of genes; each set representing genes differentially expressed in the rice varieties IR29, Pokkali, FL47 and IR63, respectively. Our algorithm was capable of constructing a

maximum of 1319 (67.4%) promoter sequences out of 1957 reference promoters (Table 5). Among these, 754 constructed sequences contained gaps. We reconstructed these sequences with GFS algorithm using L = 5 - 10, which returned a total of 723 complete promoter sequences. Thereby, our algorithm was capable to completely reconstruct 1288 promoters (65.82%) out of the total 1957 reference promoters. However, L = 5 provided the highest number of complete promoters sequences (133 promoters), whereas L = 8 produced the lowest number of promoters (102 promoters). The reconstructed promoter sequences from length 7, results the highest identity ranging 93.5% - 100% and e-Value ranging 0.0 - $9e^{-37}$. The number of promoter sequences obtained decreased with the increasing value of L, interestingly except for L = 9. However, the average identities of the reconstructed promoters with the reference sequences was decreased to L = 5 (Table 6).

Table 4. Results of 8 reconstructed promoter for different length of consensus sequences.

File id	Length	Score	eValue	Identity	Coverage (%)
Os02t0686800-01	5	1496	0.0	98%	94.3%
	7	1496	0.0	98%	94.3%
	9	1496	0.0	98%	94.3%
Os03t0184100-01	5	1653	0.0	97%	100%
	6	1653	0.0	97%	100%
	9	1653	0.0	97%	100%
Os03t0826800-01	5	1742	0.0	98%	100%
	7	1742	0.0	98%	100%
Os07t0572100-01	5	1443	0.0	93%	100%
	8	1437	0.0	93%	100%
Os07t0529600-02	9	1075	0.0	99%	94%
	10	1548	0.0	95%	100%
Os01t0307500-01	7	1786	0.0	99%	100%
Os01t0613800-01	10	1792	0.0	99%	100%
Os03t0265900-01	5	1417	0.0	99%	96.8%

The aforementioned results indicated that the newly assembled sequences were comparable and homologous with reference rice *O. sativa* Japonica promoter sequences. Thereby, this proposed algorithm can be used to construct

the sequence of a genetic element at least up to 1000 bases from a generously defined raw genome sequence data, given that the reference sequence is available.

Table 5. Pokkali promoters obtained using the proposed algorithm.

Set	Blast*Assemb*	After replacing N	Percentage
Set 1	609	244	59
Set 2	180	101	43
Set 3	219	203	73
Set 4	311	175	43

Table 6. Number of constructed promoters based on consensus sequence length (L).

Length	Promoters	eValue	Identities (%)
5	133	0.0 - $3e^{-22}$	90.25 - 100
6	128	0.0 - $2e^{-14}$	89.00 - 100
7	124	0.0 - $9e^{-37}$	93.50 - 100
8	102	0.0 - $6e^{-24}$	91.20 - 100
9	129	0.0 - $3e^{-14}$	89.75 - 100
10	107	0.0 - $9e^{-27}$	88.75 - 100

Conclusion

We have proposed an algorithm which constructs genetic elements from raw genome sequence with the help of reference sequence. For this, eValue-based weighted consensus generation showed better results. Furthermore, our algorithm also uses suffix array which is a time-and-space efficient algorithm for string matching. By using this, we have successfully identified the missing characters in the constructed genetic elements which results in complete genetic element. Using our algorithm, named SeqDev, we have successfully constructed 1288 promoter sequences of Pokkali rice variety. In future, we will further validate the efficacy and efficiency of the proposed method by assembling genetic elements from other available raw sequences and comparing this method with other available comparative genome assembly software/methods. Furthermore, our ultimate future plan is improve this algorithm and to build a complete desktop tool which will be used to construct any kind of genetic element from raw genome.

References

Altschul SF, Gish W, Miller W, Myers EW and **Lipman DJ** (1990) Basic local alignment search tool. J. Mol. Bio. **215**(3): 403-410.

Butler J, MacCallum I, Kleber M, Shlyakhter IA, Belmonte MK, Lander ES and **Jaffe DB** (2008) ALLPATHS: de novo assembly of whole-genome shotgun microreads. Genome Res. **18**(5): 810-820.

Chaisson MJ and **Pevzner PA** (2008) Short read fragment assembly of bacterial genomes. Genome Res. **18**(2): 324-330.

Cotsaftis O, Plett D, Johnson AA, Walia H, Wilson C, Ismail AM and **Baumann U** (2011) Root-specific transcript profiling of contrasting rice genotypes in response to salinity stress. Mol. Plant **4**(1): 25-41.

Dohm JC, Lottaz C, Borodina T and **Himmelbauer H** (2007) SHARCGS, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing. Genome Res. **17(11):** 1697-1706.

Farrer RA, Kemen E, Jones JD and **Studholme DJ** (2009) De novo assembly of the Pseudomonas syringae pv. syringae B728a genome using Illumina/Solexa short sequence reads. FEMS Microbiol. Let. **291**(1): 103-111.

Homer N, Merriman B and **Nelson SF** (2009) BFAST: an alignment tool for large scale genome resequencing. PLoS ONE **4**(11): e7767.

Ilie L, Fazayeli F and **Ilie S** (2011) HiTEC: accurate error correction in high-throughput sequencing data. Bioinformatics **27**(3): 295-302.

Jeck WR, Reinhardt JA, Baltrus DA, Hickenbotham MT, Magrini V, Mardis ER and **Jones CD** (2007) Extending assembly of short DNA sequences to handle error. Bioinformatics **23**(21): 2942-2944.

Li H, Ruan J and **Durbin R** (2008) Mapping short DNA sequencing reads and calling variants using mapping quality scores. Genome Res. **18**(11): 1851-1858.

Ma Q, Liu B, Zhou C, Yin Y, Li G and **Xu Y** (2013) An integrated toolkit for accurate prediction and analysis of cis-regulatory motifs at a genome scale. Bioinformatics **29**(18): 2261-2268.

Manber U and **Myers G** (1993) Suffix arrays: a new method for on-line string searches. Siam J. on Comput. **22**(5): 935-948.

Medvedev P, Georgiou K, Myers G and **Bundo M** (2007) Computability of models for sequence assembly. Algor Bioinfor. **4645:** 289-301.

Miller JR, Delcher AL, Koren S, Venter E, Walenz BP, Brownley A and **Sutton G** (2008) Aggressive assembly of pyrosequencing reads with mates. Bioinformatics **24**(24): 2818-2824.

Miller JR, Koren S and **Sutton G** (2010) Assembly algorithms for next-generation sequencing data. Genomics **95**(6): 315-327.

Myers EW (1995) Toward simplifying and accurately formulating fragment assembly. J. Comput. Biol. **2**(2): 275-290.

Nishito Y, Osana Y, Hachiya T, Popendorf K, Toyoda A, Fujiyama A and **Sakakibara Y** (2010) Whole genome assembly of a natto production strain Bacillus subtilis natto from very short read data. BMC Genomics **11**: 243.

Oehmen CS and **Baxter DJ** (2013) ScalaBLAST 2.0: rapid and robust BLAST calculations on multiprocessor systems. Bioinformatics **29**(6): 797-798.

Pop M, Phillippy A, Delcher AL and **Salzberg SL** (2004) Comparative genome assembly. Briefings in Bioinfor. **5**(3): 237-248.

Pop M and **Salzberg SL** (2008) Bioinformatics challenges of new sequencing technology. Trends in Genet. **24**(3): 142-149.

Quigley KM, Davies SW, Kenkel CD, Willis BL, Matz MV and **Bay LK** (2014) Deep-sequencing method for quantifying background abundances of Symbiodinium types: exploring the rare Symbiodinium biosphere in reef-building corals. PLoS ONE **9**(4): e94297.

Rahman T, Chowdhury T, Tabrejee S, Chowdhury MK and **Shoyaib M** (2014) BLASTAssemb: An approach to construct genetic elements using BLAST. *In:* Software, Knowledge, Information Management and Applications (SKIMA), 2014 8th International Conference on, pp. 1-5.

Salzberg SL, Sommer DD, Puiu D and **Lee VT** (2008) Gene-boosted assembly of a novel bacterial genome from very short reads. PLoS Comput. Biol. **4**(9): e1000186.

Schneeberger K, Ossowski S, Ott F, Klein JD, Wang X, Lanz C and **Weigel D** (2011) Reference-guided assembly of four diverse Arabidopsis thaliana genomes. Proc. Nat. Aca. Sci. **108**(25): 10249-10254.

Simpson JT, Wong K, Jackman SD, Schein JE, Jones SJ and **Birol I** (2009) ABySS: a parallel assembler for short read sequence data. Genome Res. **19**(6): 1117-1123.

Shrestha AMS, Frith MC and **Horton P** (2014) A bioinformatician's guide to the forefront of suffix array construction algorithms. Brief. Bioinfor. **15(2):** 138-154.

Vezzi F, Cattonaro F and **Policriti A** (2011) e-RGA: enhanced reference guided assembly of complex genomes. EMBnet. J. **17**(1): 46-54.

Wang B, Ekblom R, Bunikis I, Siitari H and **Höglund J** (2014) Whole genome sequencing of the black grouse (*Tetrao tetrix*): reference guided assembly suggests faster-Z and MHC evolution. BMC Genomics **15**(1): 180.

Xie Y, Wu G, Tang J, Luo R, Patterson J, Liu S and **Wang J** (2014) SOAPdenovo-Trans: de novo transcriptome assembly with short RNA-Seq reads. Bioinformatics **30**(12): 1660-1666.

Zerbino DR and **Birney E** (2008) Velvet: algorithms for de novo short read assembly using de Bruijn graphs. Genome Res. **18**(5): 821-829.

Zimin AV, Marçais G, Puiu D, Roberts M, Salzberg SL and **Yorke JA** (2013) The MaSuRCA genome assembler. Bioinformatics **29**(21): 2669-2677.

Zhao F, Zhao F, Li T and **Bryant DA** (2008) A new pheromone trail-based genetic algorithm for comparative genome assembly. Nucleic acids Res. **36**(10): 3455-3462.

Zhao K and **Chu X** (2014) G-BLASTN: accelerating nucleotide alignment by graphics processors. Bioinformatics **30**(10): 1384-1391.