ATLANTIS PRESS

*Advances in Intelligent Systems Research (AISR), volume 141*

International Conference on Applied Mathematics, Modelling and Statistics Application (AMMSA 2017)

# The Genome Assembly Model for Next-Generation Sequencing Data

Yirong Wang, Chengdong Wei*, Xiaodong Zhang and Tailin Cen

School of Mathematical and Statistics, Guangxi Teachers Education University, China

*Corresponding author

***Abstract*—At present, next-generation sequencing technology are quickly applied to every field of life science research. Per base has higher coverage and lower cost. Shorter reads and higher error rates from these new instruments necessitate the development of new algorithm and software[1]. We describe an assembly algorithm for next-generation sequencing data. The algorithms developed to solve this problem are based on de Bruijn graph, greedy strategy and quicksort. We explain the algorithm and present the results of assembling a bacterial artificial chromosome(BAC). The value of scaffold N50 is 71613, and N90 is 157742. And the final running time is 20.796 seconds. The value of N50 and N90 reflect the ability of scaffold sequence covering reference genome, the bigger the better. Therefore, the algorithm seems to be good for solving short reads assembly problem.**

***Keywords-de Bruijn graph; greedy strateg; quicksort; algorithm***

## I. INTRODUCTION

For each organism, the genome contains the genetic information of the whole organism. Obtaining the genetic information of the organism quickly and accurately has great significance to biological research. For now, sequencing technology is developing towards the direction of high throughput and low cost. The characteristics of huge amounts of data, short reads and low accuracy, present a rather grim challenge for the whole genome sequence assembly[2]. Genome sequencing is an important research field of current biology, and genome assembly is the primary problem. Therefore, appear lots of genome assembly algorithm and assembly software.

Steven L. Salzberg et al[3] , valuated several of the leading de novo assembly algorithms on four different short-read data sets, all generated by Illumina sequencers. Wenyu Zhang et al[4], provide the information of adaptivity for each program, then above all, compare the performance of eight distinct tools against eight groups of simulated datasets from Solexa sequencing platform. Daniel R. Zerbino and Ewan Birney[5] have developed a new set of algorithms, collectively called "Velvet" to manipulate de Bruijn graphs for genomic sequence assembly. Michael C. Schatz et al[6], describe the issues associated with short-read assembly, the different types of data produced by second-gen sequencers, and the latest assembly algorithms designed for these data. Anveshi Charuvaka and Huzefa Rangwala[7] present an evaluation of assembly of simulated short-read metagenomic samples using a state-of-art de Bruijn graph based assembler. Aarti Desai et al[8], identify the minimum depth of sequencing required for de novo assembly for different sized genomes using graph based assembly algorithms and real datasets. Sara El-Metwally et al[9], address the basic framework of next-generation genome sequence assemblers, which comprises four basic stages: preprocessing filtering, a graph construction process, a graph simplification process, and postprocessing filtering. In 2004, an algorithm that indirectly makes sequence comparisons in with respect to the size of the genome was presented by Maulik K. Shah et al[10]. Thomas C Conway and Andrew J Bromage[11] present a memory-efficient representation of the de Bruijn assembly graph using succinct data structures which allowed to represent the graph in close to the minimum number of bits.

The performance of the existing algorithms still have much room for improvement. It is urgent to develop a new genome sequence assembly algorithm, which can meet the needs of practical application, according to the characteristics of the new generation sequencing data. According to the main challenge what the genome sequence assembly is facing, and we have gotten a deeper understand about the main strategies which the genome sequence assembly use now, namely the Greedy strategy, Overlap/ Layout/ Consensus(OLC), de Bruijn graph strategy and so on, we put forward a new algorithm based on de Bruijn graph, greedy strategy and quicksort to assemble.

## II. MODEL PREPARATION

### A. Experimental Sequence Data

We obtained sequence data for the genome of a bacterial artificial chromosome (BAC) from the "Shenzhen cup" mathematical modeling camp in China (2014). The sequence was generated by Illumina Hiseq2000. The full-length is about 120000 base pairs and the sequencing depth is about 70X. The format of reads is shown as below:

- The data file format is FASTQ and every 4 line in the file indicates a reads.
- The first line contains the index sequence and the mark of read1 or read2, and "@" followed by sequence ID.
- The secend line is the base sequence represented by "ACGTN".
- In the third line, "+" denotes that the sequence ID is omitted.
- The fourth line is the quality value sequence. ASCII value of the characters minus sixty-four is the quality value.

### *B. Data Preprocessing*

Before genomes assembly, we need to filtrate the sequence reads including unreliability, low quality and artificial adding to ensure the accuracy of the assembly. The specific requirements are as follows:

- Remove the reads contained the “N” or “.”[12].
- Remove the reads of low quality (The quality value of reads is less than 7 and the base content is higher than 50%).
- Remove the reads whose “A” content is greater than or equal to 0.9.

Therefore, we need to do the following work:

*1) Gene encoding[13]*

a) *Binary encoding:* Because the DNA sequence is composed of A, T, C, G four base, which cannot be handled directly by computer, we use binary encoding. We use the four binary digits of 00(A), 01(T), 10(C) and 11(G) to represent the four bases of DNA sequence in this dissertation.

b) *Bit operation:* Bit operation is the fastest way as to computers' process speed. In order to further improve the efficiency of storage, we adopt the bit operation. The encoding program of this dissertation is obtained by C++ bit operation.

c) *Array storage[14]:* In order to avoid too much computer memory occupied by pointer when constructing the de Bruijn graph, we use arrays to store the data and then sort the data. Another advantage is easy to search. Therefore, we can use the quick and efficient method, such as binary search, to find a particular k-mer quickly. But if we use the chain table to store, the search speed will be slower.

*2) Reads quicksort[15]:* Quicksort is based on divide and conquer method and is a very popular sorting algorithm invented by C.R.A. Hoare[16] in 1962. Quicksort can sort a list of data elements significantly faster than any of the common sorting algorithms[17]. Since the data of reads have been encoded by binary, we need to improve the quicksort for short reads. In the following, we describe a reads quicksort algorithms developed by the author. The reads quicksort can be enumerated in the following steps:

Step 1: Start from the first reads in data file.

Step 2: The procedure partition: Sort the base in the order of A (00), T (01), C (10) and G (11) , and place the ith reads marked as 0 on the left and the ith reads marked as 1 on the right.

Step 3: Repeat step 2 for left and right interval until each interval only exist a read.

C++ code to implement this method for our specific problem is as follows.

```
void quick_sort(Reads s[], int l, int r,int k)
{
        int i = l, j = r;
        Reads x;
        while(i<j)
        {
            while(s[i].read[k]=='0'&&i<r)//Record the sequence
number of read of which first position is marked as '0'
                i++;
            while(s[j].read[k]=='1'&&j>l)
                j--;
            if(i<j)
            {
                x=s[i];
                s[i]=s[j];
                s[j]=x;
                i++;
                j--;
            }
        }
        if(i-1>l)
            quick_sort(s, l, i-1,k+1); //Recursive invocation
        if(r>i)
            quick_sort(s, i, r,k+1);//Recursive invocation
}
```

The principle of reads quicksort is outlined in Figure 1.

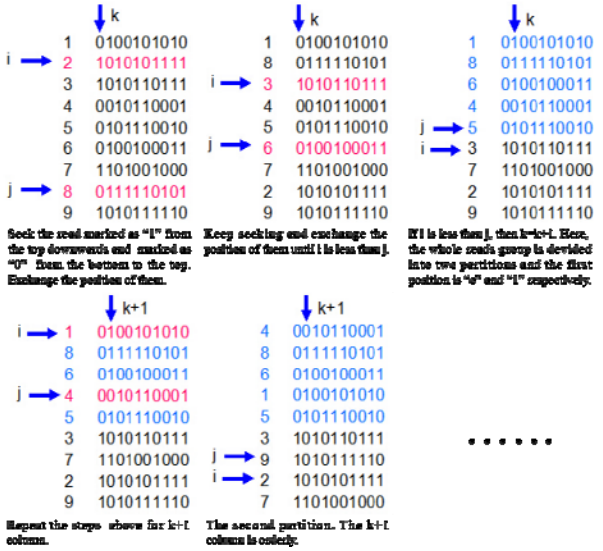


FIGURE I. THE PRINCIPLE OF READS QUICKSORT

*3) Remove the repeating segments of reads by the greedy strategy:* After finishing the reads quicksort, we obtain an orderly reads cluster. And then we remove the repeating segments of reads by the greedy strategy. The steps can be enumerated as follows:

Step 1: Calculate the value of overlap of the current ith reads and the (i+1)th reads.

Step 2: If the value of overlap is equal to the length of the reads, assign the number of (i+1)th reads of -1.

Step 3: Repeat the first and the second step.

Step 4: Delete all the reads numbered by -1.

## III. THE ESTABLISHMENT OF MODEL

Since genome sequencing was produced, the algorithm for sequence assembly is constantly being developed and improved. This paper presents a compound assembly algorithm that combines the de Bruijn graph with the greedy strategy. In this algorithm, the whole assembly process was divided into two phases: the reads assembly and the contigs assembly. The reads assembly phase is mainly to build the contings. Becasuse the assembly algorithm is based on the de Bruijn graph, the greedy

strategy and quicksort, we first need to build the de Bruijn graph before contigs construction. In this way, not only can improve the accuracy of the solution but also can reduce the consumption of memory and the occupation of CPU resources in the final improve the operation efficiency. The contigs assembly phase consists of the process of the determination of contigs relative position, the connection of contigs and the filling of gaps. And it was eventually assembled into scaffolds. In this way, it often has a relatively better assembly effect.

*A. Reads Assembly*

*1) Obtain the k-mer in each reads:* We first need to read the data file because it contains the information of the base sequence and corresponding quality of reads. We save the information of reads in advance, then each reads is read sequentially when we read the data files. Assume that the set of these reads is $F=\{f_1,f_2\ldots f_n\}$. We divide the reads into several equal length short sequences composed of continuous bases. All possible substrings of length k (termed k-mers) are generated from the sequence reads and the k-mer data set is built. A k-mer graph is a form of de Bruijn graph[18]. Its nodes represent k-mers in the graph. Its edges indicate that the adjacent k-mers overlap by exactly k-1 letters. Figure 2 give a specific example of obtaining k-mers from reads. Here, F is a set consisting of two reads, the k-mer length is 3, a read sequence is AGATACT. Its nodes are shown in figure 2 above, and its edges are shown in figure 2 below. In this way, we can construct a de Bruijn graph. The result of splice sequence can be get in the graph.

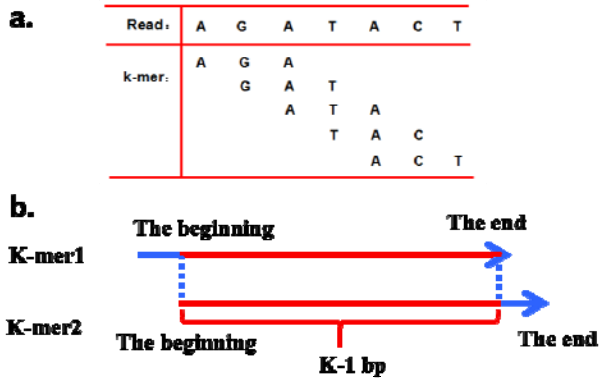


FIGURE II. EXAMPLE OF THE K-MER IN EACH READS. NODES ARE K-MERS AND EDGES ARE OVERLAPS.

*2) K-mers Quicksort:* In this algorithm, a key step is to construct de Bruijn graph. However, there are a lot of repeat fragments in gene. Therefore, the de Bruijn graph need to be simplified. Prior to this, we proceed the k-mers quicksort in each read for easier to simplify. Its basic idea is similar to reads quicksort, and specific steps are as follows:

Step 1: Start from the first k-mer in data file.

Step 2: The procedure partition. Sort the base in the order of A (00), T (01), C (10) and G (11) , and place the ith k-mers marked as 0 on the left and the ith k-mers marked as 1 on the right.

Step 3: Repeat step 2 for left and right interval until each interval only exist a k-mer.

*3) Simplify the k-mers group by the Greedy strategy:* After the completion of k-mers quicksort, we get an orderly k-mers group. Then, use the greedy algorithm to remove k-mers of duplicate and low quality values. The steps are as follows:

Step 1: Calculate the overlap value of the ith k-mers and the (i+1)th k-mers.

Step 2: If the overlap value is equal to k-1, calculate the quality value of the last gene between the two k-mers according to the position information of the k-mers.

Step 3: If the quality of the ith k-mers is greater than the quality of the (i+1)th k-mers, then assign the number of (i+1)th k-mers of -1, otherwise assign the number of ith k-mers of -1.

Step 4: Calculate the overlap value of the ith k-mers and the (i+2)th k-mers. If the overlap is not equal to k-1, then i= i+1. Repeat the process of step1 to step3. If the overlap value is equal to k-1, calculate the quality value of the last gene between the two k-mers according to the position information of the k-mers.

Step 5: If the quality of the ith k-mers is greater than the quality of the (i+2)th k-mers, then assign the number of (i+2)th k-mers of -1, otherwise assign the number of ith k-mers of -1.

Step 6: Repeat all steps above.

Step 7: Delete all the k-mers of number -1.

*4) De Bruijn graph construction:* The algorithm based on de Bruijn graph is the most widely used to the next-generation sequencing data. At present, many assembly softwares expanded are also based on the de Bruijn graph. The typical assembly softwares are: ALLPATHS, ABySS, Euler-SR, SOAPdenovo and Velvet[19].

Therefore, in order to better realize the algorithm based on de Bruijn graph and greedy strategy, we need to construct the de Bruijn graph. The algorithm cleverly map the k-mers with overlapping relationship together. In that case, the computational complexity and the memory consumption is reduced. The basic steps are as follows.

Step 1: Select the available k-mers0 (lock equal to 0).

Step 2: If there are multiple available k-mers0 (up to 4) to be selected, it means that there are error k-mers in the k-mers0. At the moment, we need to select the the highest quality value of the error gene as the only k-mers0, then assign the lock of the rest k-mers to -1.

Step 3: Use the dichotomy to find the next k-mers1 which can be spliced with the current k-mers0.

Step 4: Mark the value of next_kmer of current k-mer0 as k-mers1, and then mark the value of front_kmer of k-mers1 as k-mers0.

Step 5: Repeat all steps above until the available k-mers is empty.

*5) Contigs Construction:* An overall consideration of the information of k-mers involved in the splicing and a splicing

problem study considering from the whole k-mers are the basic idea of the whole genome short sequence splicing algorithm based on de Bruijn graph and greedy strategy. The main approach is as follows. Encapsulate the data of the assembling k-mers in an object one by one. The object store the number, content, position and the lock (If the lock is equal to 0 means that it can be spliced, else -1 means it can't be spliced) of k-mers in the reads. Considering the data characteristics of k-mers in object comprehensively, we designed the following contigs construction process:

Step 1: Search the ith k-mers of which front_kmer is -1, n=1, and then build a contig.

Step 2: Add the last gene recorded as next_kmer to the end of contigs according to the next_kmer of (i+n)th k-mers, and then the length of contigs plus 1.

Step 3: If the next_kmer of (i+n)th k-mers is not -1, then n=n+1, and search the (i+n)th k-mers recorded as next_kmer.

Step 4: Repeat step 2 until the next_kmer of (i+n)th k-mers is -1.

Step 5: Repeat step1 to step5 until the end.

*B. Contigs Assembly*

This phase take the contigs generated in reads assembly stage and paired data file as input. Through the process of contigs relative position determination, contigs connection and gaps filling, a longer length scaffolds is formed

*1) Contigs Relative Position Determination:* Before the contigs assembly, we need to fix the relative position and direction between contigs. If we map the paired data to the contigs generated by reads assembly stage, there usually have several pairs of reads mapped to different contigs. If there are overlaps between the two contigs, we need to see if the distance between the two paired reads in a reasonable range. If the distance is in the reasonable scope and the number of paired reads achieve the setting threshold, we consider the two contigs are adjacent to each other and should be linked together. If there are no overlaps between the two contigs, we consider there exist gaps between them and need to fill them in subsequent gaps filling operation. The result of practice proves that this approach is consistent with the actual situation of the target genome.

Therefore, we abandoned the general approach to make the paired reads data map to the whole contigs, but only take the sequence fragments of L bp at each end of the contigs. This way not only reduces the memory consumption but also improve the computing speed. If the contigs length is less than L, we take the whole sequence. The L value is related to the overall mean value of the k-mers and greater than the average value in general. For instance, if the average length is 200 bp, the L value is 300 bp.

In the previous, we have completed the reads quicksort and k-mer quicksort, and stored the information of number, content, position, etc, of reads and k-mers in the object. Hence, it is quite convenient to map the reads to contigs. But when paired reads data map into contigs, some paired reads only have a read mapped to the contigs and the other can not find the location in the contigs, some are mapped to different locations of one or multiple contigs. Namely, the matching position of a single read is not unique. This brings a great trouble to the determination of the relative position between contigs. The best solution generally is to consider as the two contigs are adjacent to each other in the target genome when the pair reads matched to different contigs reach a certain threshold.

*2) Contigs Connection:* From the above we only know the relative position of contigs but they do not really connect together. When the contigs connect, we need to calculate the overlaps between contigs. Due to the relative position between the contigs have been confirmed, we can calculate the distance between the contigs according to the paired reads of two adjacent contigs. Then the size of overlaps between contigs is confirmed. At the moment, the distance between contigs can be greater or less than zero. If the distance is less than zero, it means that there exist overlaps. If the distance is greater than zero, there exist gaps.

Below we take contigs A and contigs B for example to detailedly describe the contigs connection process. If only one read of paired reads is mapped to the two adjacent contigs, the calculation formula is $g = \overline{m} - l_1 - l_2$, as shown in figure 3. If several reads of paired reads are mapped to the two adjacent contigs, the calculation formula is $d(A,B) = E(g) = \frac{1}{n}\sum_{i=1}^{n} g_i$, as shown in figure 4.

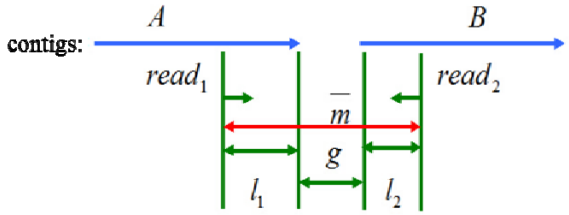


FIGURE III. ONLY ONE READ OF PAIRED READS IS MAPPED TO THE TWO ADJACENT CONTIGS.

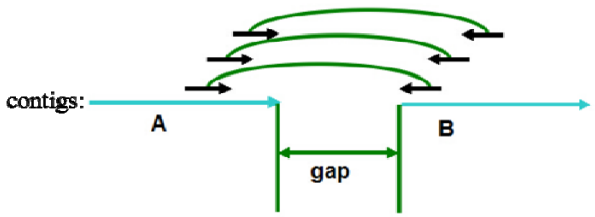


FIGURE IV. SEVERAL READS OF PAIRED READS ARE MAPPED TO THE TWO ADJACENT CONTIGS.

*3) Gaps Filling:* During the stage of contigs connecting, the contigs contained overlaps are connected together and merged into a longer contigs. But there exist gaps between the adjacent and unconnected contigs, and we need to fill them. The way of filling the gaps is to construct the missing bases sequence by partial sequence splicing.

## IV. THE SOLUTION OF THE MODEL

In the process of reads assembly, we first encapsulate the reads data in the object one by one. In order to read and dock, the object store the reads number, content and position. As you can see from the figure 5, the number of 12260, 12261, 34858 and 41014 in the read1 contain the "N", and the number of 12261 contain two "N" especially. The number of low-quality reads is 0. The number of reads whose quality value is greater than or equal to 0.9 is also 0. In the figure 6, only the number of 28815, 37835 and 38992 in the read2 contain the "N", and the number of 38992 contain two "N" especially. The number of low-quality reads is 0. The number of reads whose quality value is greater than or equal to 0.9 is also 0.

Next, we use the algorithm to removed the incorrect base sequences, and get 46841 valid reads. The number of k-mer is 682969. One indicator commonly used to evaluate the efficacy of assembly is the N50 statistic. The N50 is a weighted median of the lengths of the sequences, equal to the length of the longest sequences L such that the sum of the lengths of sequences greater than or equal in length to L is greater than or equal to half the length of the genome being assembled[20]. N90 is another statistic to assess assemblies. The N90 is equal to the length of the longest sequences L such that the sum of the lengths of sequences greater than or equal in length to L is greater than or equal to 90% the length of the genome being assembled. Various combinations of sequencing libraries were used to obtain the best scaffold N50s and N90s using SOAPdenovo[21]. The value of N50 and N90 reflect the ability of scaffold sequence covering reference genome, the bigger the better. In this paper, the value of N50 is 71613, and the value of N90 is 157742. And the final running time is 20.796 seconds.

## V. CONCLUSION

The assembly algorithm in this paper obey the assessment standard above. The value of N50 and N90 is relatively large, and the assembly effect is relatively good. But the number of k-mer is too much, there's room for improvement. Fortunately, the running speed is very fast, only 20.796 seconds.

## ACKNOWLEDGMENT

The authors thank the anonymous for their valuable suggestions to improve the quality of this paper. This paper was partially supported by the Key Laboratory for data science of Universitie of Guangxi Province(8852), National Natural Science Foundation of China (11561010) and Innovation Project of Guangxi Graduate Education (YCSW2017188).

## REFERENCES

[1] Ratan A, Assembly algorithms for next-generation sequence data, Dissertations & Theses - Gradworks, 2009.

[2] Luo R, Liu B, Xie Y, et al, "SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler." GigaScience, 2012, vol.1, pp.18.

[3] Salzberg, S. L., et al, "GAGE: A critical evaluation of genome assemblies and assembly algorithms," Genome Research, 2011, vol.22, pp.557-567.

[4] Zhang, W, et al. "A practical comparison of de novo genome assembly software tools for next-generation sequencing technologies," Plos One, 2011, vol. 6, pp.e17915.

[5] Zerbino D R, Birney E, "Velvet: algorithms for de novo short read assembly using de Bruijn graphs," Genome Research, 2008, vol.18, pp.821-829.

[6] Schatz M C, Delcher AL, and Salzberg S L, "Assembly of large genomes using second-generation sequencing," Genome Researc, 2010, vol.20, pp.1165-1173.

[7] Charuvaka A, Rangwala H, "Evaluation of short read metagenomic assembly," Bmc Genomics, 2011, vol.Suppl12, pp.1-13.

[8] Desai A, Marwah V S, Yadav A, et al, "Identification of Optimum Sequencing Depth Especially for De Novo Genome Assembly of Small Genomes Using Next Generation Sequencing Data," Plos One, 2013, vol.8, pp.e60204.

[9] El-Metwally S, Hamza T, Zakaria M, et al, "Next-Generation Sequence Assembly: Four Stages of Data Processing and Computational Challenges," Plos Computational Biology, 2013, vol.9, pp.e1003345-e1003345.

[10] Shah M K, Lee H J, Rogers S A, et al, "An Exhaustive Genome Assembly Algorithm Using K-Mers to Indirectly Perform N-Squared Comparisons in O(N)," IEEE Computer Society, IEEE Computational Systems Bioinformatics Conference, pp.740-741, 2004.

[11] Conway T C, Bromage A J, "Succinct data structures for assembling large genomes," Bioinformatics, 2011, vol.27, pp.479-86.

[12] Chen C, Khaleel S S, Huang H, et al, "Software for pre-processing Illumina next-generation sequencing short read sequences," Source Code for Biology & Medicine, 2014, vol. 9, pp.8-8.

[13] Sadasivan E, Cedeno M M, Rothenberg S P, "Characterization of the gene encoding a folate-binding protein expressed in human placenta. Identification of promoter activity in a G-rich SP1 site linked with the tandemly repeated GGAAG motif for the ets encoded GA-binding protein. Journal of Biological Chemistry," 1994, vol.26, pp.4725-35.

[14] Rusu F, Cheng Y, "A Survey on Array Storage, Query Languages, and Systems," Short Term Thermal Energy Storage, 2013.

[15] Hoare, C. A R, "Quicksort," Computer Journal, 1962, vol. 5, pp.10-15.

[16] Sedgewick R, "The analysis of Quicksort programs," Acta Informatica, 1977, vol. 7, pp.327-355.

[17] Mishra A D, Selection of Best Sorting Algorithm for a Particular Problem, 2009.

[18] Miller J R, Koren S, Sutton G, "Assembly algorithms for next-generation sequencing data," Genomics, 2010, vol.95, pp.315-327.

[19] Nagarajan N, Pop M, "Sequence assembly demystified," Nature Reviews Genetics, 2013, vol.14, pp.157-167.

[20] Earl D, Bradnam K, John J S, et al, "Assemblathon 1: a competitive assessment of de novo short read assembly methods," Genome Research, 2011, vol.21, pp.2224-2241.

[21] Krishnan N M, Pattnaik S, Jain P, et al, "A Draft of the Genome and Four Transcriptomes of a Medicinal and Pesticidal Angiosperm Azadirachta indica," BMC Genomics, 2012, vol.13, pp.1-13.