An efficient classification algorithm for NGS data based on text similarity

Xiangyu Liao¹,*, Xingyu Liao²,*, Wufei Zhu³, Lu Fang³ and Xing Chen³

¹Department of Oncology, The First College of Clinical Medical Science, China Three Gorges University, Yichang Central People’s Hospital, Yichang, Hubei 443000, P.R. China; ²School of Information Science and Engineering, Central South University, Changsha, Hunan 410083, P.R. China and ³Department of Endocrinology, The First College of Clinical Medical Science, China Three Gorges University, Yichang Central People’s Hospital, Yichang, Hubei 443000, P.R. China

Abstract

With the advancement of high-throughput sequencing technologies, the amount of available sequencing data is growing at a pace that has now begun to greatly challenge the data processing and storage capacities of modern computer systems. Removing redundancy from such data by clustering could be crucial for reducing memory, disk space and running time consumption. In addition, it also has good performance on reducing dataset noise in some analysis applications. In this study, we propose a high-performance short sequence classification algorithm (HSC) for next generation sequencing (NGS) data based on efficient hash function and text similarity. First, HSC converts all reads into k-mers, then it forms a unique k-mer set by merging the duplicated and reverse complementary elements. Second, all unique k-mers are stored in a hash table, where the k-mer string is stored in the key field, and the ID of the reads containing the k-mer are stored in the value field. Third, each hash unit is transformed into a short text consisting of reads. Fourth, texts that satisfy the similarity threshold are combined into a long text, the merge operation is executed iteratively until there is no text that satisfies the merge condition. Finally, the long text is transformed into a cluster consisting of reads. We tested HSC using five real datasets. The experimental results showed that HSC cluster 100 million short reads within 2 hours, and it has excellent performance in reducing memory consumption. Compared to existing methods, HSC is much faster than other tools, it can easily handle tens of millions of sequences. In addition, when HSC is used as a preprocessing tool to produce assembly data, the memory and time consumption of the assembler is greatly reduced. It can help the assembler to achieve better assemblies in terms of N50, NA50 and genome fraction.

1. Introduction

High-throughput sequencing technologies produce voluminous amounts of DNA sequence data, which are employed in a wide array of biological applications including de novo genome assembly, sequence error correction, expression analysis and detection of sequence variants. Data volume generated by next generation sequencing technologies is growing at a pace that is now challenging the storage and data processing capacities of modern computer systems (Medini et al., 2008). Take genome assembly as an example, the de novo assembler ALLPATHS-LG (default k-mer size is 96) required approximately 3 weeks with 48 processors and approximately 0.5TB of RAM memory for the human whole-genome (3Gb) assembly (Gnerre et al., 2011). Clustering analysis is a method that identifies and groups similar objects. It is a powerful tool to explore and study large-scale complex data. By sequence clustering, a large redundant data set can be represented by a small non-redundant set, which could be crucial for reducing storage space, computational time and noise interference in some analysis applications. Errors can be detected, filtered or corrected using consensus from sequences with clusters. In addition, some additional challenges brought by the high-throughput sequencing technologies may also be solved by clustering (Li et al., 2012).

In order to design fast and accurately clustering algorithms for next generation sequencing (NGS) sequences, researchers put forward some new solutions. Typical clustering algorithms for NGS sequences includes: CD-HIT (Li & Godzik, 2006), SEED (Bao et al., 2011), Uclust (Edgar, 2010), and DNAclust (Ghodsi et al., 2011). CD-HIT is a comprehensive clustering package (Huang et al., 2010; Li & Chang, 2017). CD-HIT uses a greedy incremental algorithm. Basically, sequences are first ordered by decreasing length, and the longest one becomes the seed of the first cluster. Then, each remaining sequence is compared with existing seed. If the similarity with any seed meets a pre-defined cutoff, it is grouped into that cluster; otherwise, it becomes the seed of a different cluster. CD-HIT uses a heuristic based on statistical k-mer filtering to accelerate clustering calculations. It also has a multi-threading function, so it can be implemented in parallel on multi-core computers (Li, 2015; Bruneau et al., 2018).
SEED, Uclust and DNACLUST have been developed using greedy incremental approaches similar to that introduced by CD-HIT (James et al., 2017; Rahman et al., 2017). Uclust (Ennis et al., 2016; Jiang et al., 2016) follows CD-HIT’s greedy incremental approach, but it uses a heuristic called Usearch for fast sequence comparison. It also gains speed by comparing a few top sequences instead of the full database. DNACLUST (Rahman et al., 2017; Kim, 2015) also follows CD-HIT’s greedy incremental approach, it requires a suffix array to index the hash units that store low reliability unique computational complexity and runtime consumption, some (Rahman et al., 2017) few top sequences instead of the full database. DNACLUST fast sequence comparison. It also gains speed by comparing a incremental approach, but it uses a heuristic called Usearch for very reliable unique elements. Second, all k-mers unique are stored in a hash table, where the k-mer string is stored in the key field, and the ID of the reads containing the k-mer are stored in the value field. More read IDs stored in the value field indicate the more reads that share the unique k-mer, it also indicates that the unique k-mer is more reliable. To reduce computational complexity and runtime consumption, some hash units that store low reliability unique k-mers are filtered out. Third, each hash unit is transformed into a short text consisting of reads. Fourth, short texts that satisfy the similarity threshold are combined into a long text, the merge operation is executed iteratively until there is no text that satisfies the merge condition. Finally, the long text is transformed into a final cluster consisting of reads. The efficient hash function can quickly complete the preliminary clustering of the data, and then the data in a hash unit is converted into a short text, and the alignment between the mass reads becomes the similarity comparison between the short texts, which greatly reduces the complexity of the computation. The application of these two technologies enables the HSC to adapt to large-scale data sets.

2.1. Notations
The entire read set is denoted as R, and we assume that there are n reads in R. Let \( r_i \) be the i-th read \( (i = 1,2,\ldots,n) \), and \( |r_i| \) be the length of the read i. Given a fixed length k of k-mer, the read i can be represented as a list of \( (|r_i| - k + 1) \) k-mers. The unique k-mer set is formed by merging the duplicated and reverse complementary elements in original k-mer set. N50 is the length for which the collection of all contigs of that length or longer covers at least half an assembly. NG50 is the length for which the collection of all contigs of that length or longer covers at least half the reference genome. Misassemblies is the number of mis-assembly events and local mis-assembly events. NA50 is the value of N50 after contigs have been broken at every misassembly event. NG50 is the value of NG50 after contigs have been broken at every misassembly event. Genome fraction (%) is the proportion of the genome reference being covered by output contigs.

2.2. Generating a unique k-mer set
The unique k-mer set can be obtained using many of the currently available tools, such as Jellyfish (Marcais & Kingsford, 2011), DSK (Rzik et al., 2013) and KMC2 (Deorowicz et al., 2015). In this study, we used DSK to obtain the unique k-mer set. Note that before converting read to k-mers, the k-mer size k needs to be determined. k should be kept small to avoid the overuse of computer memory. To estimate genomic characters, the k-mer size k should be determined under the logic that the space of k-mer \( (4^k) \) should be several times larger than the genome size \( (G) \), so that few k-mers derived from different genomic positions will merge together by chance, that is, most k-mers in the genome will appear uniquely. In practice, we often require the k-mer space to be at least five times larger than the genome size \( (4^k > 5G) \), and be less than the upper limit of system memory usage (Liu et al., 2013). In addition, as the value of k increases, the time required for the k-mer frequency statistics also increases. Therefore, when selecting the size of k, it is necessary to comprehensively consider the memory requirement, the time requirement and the accuracy requirement. In this study, we set the value of k to be 15, which has fully considered the above factors. The rule for k-mer size selection is shown in Equation (1).

\[
4^k > 5G
\]  

(1)

2.3. Storing all unique k-mers in a hash table
After the unique k-mer set is generated, all unique k-mers are stored in a hash table, where the k-mer string is stored in the key field, and the ID of the reads containing the k-mer are stored in the value field. In this study, we chose SDM (Jain & Pandey, 2012) as the hash function, which is widely used in various
applications. The mathematical expression of SDBM is shown in Equation (2), where \( ch \) represents the ASCII of each character in the sequence, >> is a bitwise left shift operator, \( h \) represents the hash value, it initialized as zero. SDBM is a standard hash function which has very fewer chances of collisions, even in a very large scale data set.

\[
h = ch + (h << 6) + (h << 16) - h
\]

(2)

The number of reads stored in the value field can reflect the credibility of the unique k-mer. The smaller the number of read IDs stored in the hash unit, the lower the frequency of the corresponding unique k-mer. It is well known that the extremely low frequency k-mer is often caused by sequencing errors or sequencing bias (Kelley et al., 2010; Sohn & Nam, 2016). In the case of a large sequencing depth, there is a high possibility that the low-frequency k-mer is an erroneous k-mer. Keeping these extremely low frequency k-mers in a hash table not only increases the cost of calculations, but also affects the quality of downstream applications. In order to reduce the computational complexity and improve the quality of downstream analysis, hash units that store less than 10 read IDs will be removed.

2.4. Transforming each hash unit into a short text

The storage structure of a hash unit can be represented as: \( H(< \text{unique k-mer i}>, < \text{read IDa, ..., read IDn}>) \). The ID of read indicates the serial number of the read in the library (read set). In this step, each hash unit is transformed into a short text. The short text consists of read strings corresponding to the read IDs stored in the hash unit. In order to facilitate downstream applications, the short text is set to fasta format. The principle of converting a hash unit to a short text is shown in Fig. 2.

2.5. Calculation of similarity between short texts

Text comparison takes the form of solving the similarity between two or more texts. The higher the similarity, the more similar the two texts are. Text comparison is commonly used in fields such as...
data retrieval, natural language processing and information matching (Inzalkar & Sharma, 2015; Liu et al., 2017; Pu et al., 2017). The comparison of similarity between texts can be divided into methods based on character similarity (Wen et al., 2017), statistics based similarity (Lin et al., 2014) and semantic-based similarity (Omaras et al., 2015). The character-based similarity calculation method is a basic algorithm for similarity calculation of text, the most representative character-based similarity calculation algorithm is the edit distance (Levenshtein distance) algorithm (Wang et al., 2010), which is used to solve the minimum number of edits required to covert one text to another text. In the Levenshtein distance based similarity calculation method, the smaller the edit distance between two or more texts, the higher the similarity between two or more texts. Although the similarity calculation methods based on the Levenshtein distance are widely used, they only consider the text similarity from the character level while not considering the influence of the high-frequency substrings, thus leading to low accuracy of similarity. In this study, we calculated the similarity between texts based on the vector space model (VSM). VSM is the simplest way to represent a document for the purpose of information retrieval. We assume there are M documents in the library, and V is the set of the vocabulary term/words occurring in the library. The document in VSM is represented by a \(|V|\)-dimensional vector, each element of the vector represents the frequency of occurrence of each word in the document.

### 2.5.1. Similarity measure in VSM

Let \(T = \{T_1, T_2, \ldots, T_n\}\) be a set of unique \(k\)-mers for all documents. Let \(W = \{W_1, W_2, \ldots, W_n\}\) be a set of weight for all unique \(k\)-mers. For example, \(W_i\) represents the importance of the unique \(k\)-mer \(T_i\) in set \(T\). We can construct \(n\)-dimensional space based on set \(T\) and \(W\), where \((W_1, W_2, \ldots, W_n)\) are the coordinate values corresponding to \((T_1, T_2, \ldots, T_n)\). The above conversion can be used to map a document to a point in vector space. Let any document be represented by \(D = (W_1, W_2, \ldots, W_n)\). The calculation of weight of \(W\) is shown in Equation (3), where \(TF_i\) represents the frequency of the \(i\)-th unique \(k\)-mer appearing in the document \(D\), \(n_i\) represents the number of documents in a document collection, and \(n\) represents the number of occurrences of \(T_i\) in the document collection.

\[
W_i = TF_i \times \log_2(N/n_i)
\]  

### 2.5.2. Similarity calculation based on cosine coefficient

\(TF_i\) represents the frequency of the \(i\)-th unique \(k\)-mer appearing in the document \(D\), and \(IDF\) represents the frequency of anti-document. If there are fewer documents containing the term \(T\), then the \(IDF\) is larger, which indicates that the term \(T\) has a good distinguishing ability. Otherwise, the distinguishing ability of \(T\) is poor. Let the target document \(D'\) be represented by \(D' = (W'_1, W'_2, \ldots, W'_n)\). Then, the similarity between documents \(D\) and \(D'\) can be calculated from Equation (4).

\[
sim(d, d') = \frac{\sum_{k=1}^{n} W_k d_k W'_{jk}}{\sqrt{\sum_{k=1}^{n} W_k^2 \sum_{k=1}^{n} W'_{jk}^2}}
\]

![Fig. 3. Classification performance of HSC on simulated data. The y-axis represents the position on the reference, and the x-axis represents the ID of the cluster.](image)

**Table 1.** Details of datasets.

| Species     | R.spha | M.absb | V.chol | Hum14 |
|-------------|--------|--------|--------|-------|
| Library     | lib1   | lib2   | lib3   | lib4  |
| Sequencing type | Hi-Seq | Hi-Seq | Hi-Seq | Hi-Seq |
| Genome size (Mbp) | 4.6    | 3.9    | 5.0    | 88.29 |
| Read length (bp) | 101    | 100    | 100    | 101   |
| Number of reads | 4628173 | 3957421 | 5090491 | 109037804 |
| Coverage    | ~210   | ~100   | ~95    | ~102  |
| Insert size (bp) | 220    | 335    | 335    | 155   |

3. **Experiments and results**

To evaluate the performance of HSC, a simulated dataset and four real datasets were used in the experiments in this study. The simulated dataset was generated by a very famous data simulation tool called wgsim (Li, 2011) ([https://github.com/lh3/wgsim](https://github.com/lh3/wgsim)). The first three real datasets (R.spha, M.absb and V.chol) were downloaded from the GAGE-B website ([http://ccb.jhu.edu/gage_b/](http://ccb.jhu.edu/gage_b/)), the last real dataset (Hum14) was downloaded from the GAGE website ([http://gage.cb.cb.umd.edu/](http://gage.cb.cb.umd.edu/)). Details about these four real datasets are shown in Table 1.

In simulation experiments, a fragment on the Staphylococcus albus reference sequence was selected as a
reference, and Wgsim was used as a simulator to generate a library consisting of paired-end reads. In this experiment, the length of reads was set to 101, and the size of k-mer used in the classification was set to 15. The minimum length of the hash unit was set to 10. The total number of paired-end reads which are generated by the Wgsim simulator was one million. The error rate of reads was set to 0.5%. HSC completed the classification within 5 minutes. In order to evaluate the performance of HSC on simulation dataset, we used bowtie2 (Langmead & Salzberg, 2012) to align reads in a cluster to the reference sequence, and record the location of these reads on the reference sequence, respectively. The classification results are shown in Fig. 3, where the y-axis represents the position on the reference, and the x-axis represents the ID of the cluster. It can be seen from Fig. 3 that HSC divided a million reads into five clusters, and the distribution of these five clusters on the reference sequence is relatively independent.

In the experiments of four real datasets, we selected R. sphaeroides (Rhodobacter sphaeroides), M. abscessus (Mycobacterium abscessus), V. cholerae (Vibrio cholerae) and Hum14 (Human 14) to test the performance of HSC. Details about these four datasets are shown in Table 1. The classification results of HSC on these four real datasets are shown in Table 2, Table 3, Table 4 and Table 5. In these experiments, we used SOAPdenovo2 (Luo et al. 2012) as an assembler, and the size of k-mer was set to 49 bp during the assembly. The evaluation indicators of N50,

Table 2. The classification results of HSC on lib1.

| Methods | No. clusters | SimT  | Time (m) | Mem (GB) | N50 (bp) | NG50 (bp) | N50 (bp) | NG50 (bp) | N50 (bp) | NG50 (bp) | GF (%) |
|---------|--------------|-------|----------|----------|----------|-----------|----------|-----------|----------|-----------|-------|
| SEED    | 1056109      | 0.96  | 7.9      | 2.5      | 7321     | 6278      | 4832     | 4544      | 78.21    |
| CD-HIT  | 750276       | 0.90  | 347.5    | 5.2      | 6543     | 4951      | 4725     | 4631      | 79.03    |
| Uclust  | 734981       | 0.90  | 227.5    | 0.5      | 5476     | 4461      | 3638     | 2315      | 61.45    |
| HSC     | 1013212      | 0.96  | 5.8      | 2.3      | 9213     | 7480      | 7342     | 6914      | 82.35    |

*No. clusters* indicates the number of clusters, *Time (m)* indicates the time consumption in clustering, *Mem (GB)* indicates the memory consumption in clustering, *SimT* indicates the threshold of similarity, *GF (%)* indicates the genome fraction. Bold values represent best results.

Table 3. The classification results of HSC on lib2.

| Methods | No. clusters | SimT  | Time (m) | Mem (GB) | N50 (bp) | NG50 (bp) | N50 (bp) | NG50 (bp) | N50 (bp) | NG50 (bp) | GF (%) |
|---------|--------------|-------|----------|----------|----------|-----------|----------|-----------|----------|-----------|-------|
| SEED    | 1012136      | 0.96  | 6.3      | 1.9      | 6714     | 5417      | 4416     | 4212      | 74.33    |
| CD-HIT  | 610462       | 0.90  | 235.3    | 4.7      | 6140     | 5378      | 5003     | 4098      | 72.34    |
| Uclust  | 602981       | 0.90  | 206.1    | 0.3      | 4817     | 4568      | 3954     | 3571      | 67.78    |
| HSC     | 976542       | 0.96  | 4.2      | 1.7      | 6544     | 6354      | 5215     | 4568      | 76.21    |

*No. clusters* indicates the number of clusters, *Time (m)* indicates the time consumption in clustering, *Mem (GB)* indicates the memory consumption in clustering, *SimT* indicates the threshold of similarity, *GF (%)* indicates the genome fraction. Bold values represent best results.

Table 4. The classification results of HSC on lib3.

| Methods | No. clusters | SimT  | Time (m) | Mem (GB) | N50 (bp) | NG50 (bp) | N50 (bp) | NG50 (bp) | N50 (bp) | NG50 (bp) | GF (%) |
|---------|--------------|-------|----------|----------|----------|-----------|----------|-----------|----------|-----------|-------|
| SEED    | 1160217      | 0.96  | 9.1      | 2.7      | 8905     | 7541      | 6840     | 7038      | 83.36    |
| CD-HIT  | 821654       | 0.90  | 371.2    | 7.2      | 7312     | 6547      | 6259     | 5996      | 79.18    |
| Uclust  | 834012       | 0.90  | 312.3    | 0.7      | 6340     | 5946      | 5536     | 5438      | 76.54    |
| HSC     | 1139807      | 0.96  | 8.7      | 2.5      | 10361    | 9154      | 8325     | 8078      | 85.46    |

*No. clusters* indicates the number of clusters, *Time (m)* indicates the time consumption in clustering, *Mem (GB)* indicates the memory consumption in clustering, *SimT* indicates the threshold of similarity, *GF (%)* indicates the genome fraction. Bold values represent best results.

Table 5. The classification results of HSC on lib4.

| Methods | No. clusters | SimT  | Time (m) | Mem (GB) | N50 (bp) | NG50 (bp) | N50 (bp) | NG50 (bp) | N50 (bp) | NG50 (bp) | GF (%) |
|---------|--------------|-------|----------|----------|----------|-----------|----------|-----------|----------|-----------|-------|
| SEED    | 1732567      | 0.96  | 50.7     | 5.8      | 3854     | 3720      | 3543     | 3259      | 65.42    |
| CD-HIT  | 956346       | 0.90  | 556.9    | 10.5     | 3645     | 3259      | 3186     | 3098      | 61.54    |
| Uclust  | 987495       | 0.90  | 476.5    | 2.9      | 3247     | 2763      | 2294     | 2044      | 63.84    |
| HSC     | 1587746      | 0.96  | 47.7     | 5.5      | 4361     | 3681      | 3641     | 3315      | 68.63    |

*No. clusters* indicates the number of clusters, *Time (m)* indicates the time consumption in clustering, *Mem (GB)* indicates the memory consumption in clustering, *SimT* indicates the threshold of similarity, *GF (%)* indicates the genome fraction. Bold values represent best results.
NG50, NA50, NGA50 and genome fraction were obtained by Quast (Gurevich et al., 2013). The experimental results show that HSC cluster 100 million short reads within 2 hours, and it has excellent performance in reducing memory consumption. For example, as shown in Table 2, HSC generated 1,013,212 clusters requiring 2.3GB of memory (as compared to the tool of CD-HIT, which produced 750,276 clusters requiring 5.2GB memory). Compared to existing methods, HSC is much faster than other tools, it can easily handle tens of millions of sequences. For example, as shown in Table 5, HSC generated 1,587,746 clusters on the Human 14 data set, which took 47.7 minutes (as compared to the tool of CD-HIT, which produced 956,346 clusters requiring 556.9 minutes). In addition, when HSC is used as a preprocessing tool to produce assembly data, the memory and time consumption of the assembler is greatly reduced. It can help the assembler to achieve better assemblies in terms of NG50, NA50, NGA50 and genome fraction. For example, as shown in Table 3, HSC generated 976,542 clusters, SOAPdenovo2 assembled 76.21% of V. cholerae genome with NA50 of 5.215 kb based on these clusters (as compared to the tool of SEED, SOAPdenovo2 assembled 74.33% of V. cholerae genome with NA50 of 4.416 kb based on clusters generated by SEED).

4. Discussion

In this study, we proposed a HSC for NGS data based on efficient hash function and text similarity. HSC has applied two important techniques. The first is an efficient storage structure based on hash function, and the second is a precise text similarity calculation model based on cosine coefficients. The efficient hash function can quickly complete the preliminary clustering of the data, and then the data in a hash unit is converted into a short text, the alignment between the mass reads becomes the similarity comparison between the short texts, which greatly reduces the complexity of the computation. The time complexity of alignment between reads in the library is O(n^2). HSC uses the hash function to achieve an initial classification of reads, the time complexity of this process is O(n). After that, HSC converts the two-to-two comparison between reads into an alignment between short texts (suppose a short text contains r reads on average, the number of short texts is m), the time complexity of this process can be expressed as O(m^2) (m ≃ n/r). The application of these two technologies makes HSC processing large-scale data sets faster than other tools.

It is well known that extremely low frequency k-mers are often caused by sequencing errors or sequencing bias. In the case of a large sequencing depth, there is a high possibility that the low-frequency k-mer is an erroneous k-mer. Keeping these extremely low frequency k-mers in a hash table not only increases the cost of calculations, but also affects the quality of downstream applications. In order to reduce the computational complexity and improve the quality of downstream analysis, hash units that store less than 10 read IDs were removed in this study. This operation allows the HSC to produce a better quality data set for downstream analysis than other tools.

5. Conclusion

In this study, we proposed a HSC for NGS data based on efficient hash function and text similarity. We tested HSC using a simulation dataset and four real datasets. The experimental results show that HSC cluster 100 million short reads within 2 hours, and has excellent performance in reducing memory consumption. Compared to existing methods, HSC is much faster than other tools and can easily handle tens of millions of sequences. In addition, when HSC is used as a preprocessing tool to produce assembly data, the memory and time consumption of the assembler is greatly reduced. It can help the assembler to achieve better assemblies in terms of NG50, NA50, NGA50 and genome fraction.

6. Future works

The calculation method of similarity between short texts is the core of the classification algorithm in this study. Although, HSC performs better than other methods on various datasets, there is still much room for improvement in its performance. Next, we will further study the similarity comparison between short texts and propose a more superior comparison algorithm. In addition, low quality read filtering is also a very important operation in classification, which affects the quality of downstream analysis directly. We will also further study the new method of low-quality read filtering.

Acknowledgments. This work has been supported by the National Natural Science Foundation of China under Grant No.81202303, No.31302056, the Health Technology and Development Research Plan of Yichang under Grant No.A12301-07, No.A13301-10, and the Research Development Foundation of Center Hospital of Yichang under Grant No.KFJ2011025.

Authors. Xiangyu Liao received her Ph.D. degree in Basic Medicine from Norman Bethune College of Medicine, Jilin University, Changchun, China, in 2011. She is currently an attending doctor in the Department of Oncology, China Three Gorges University & Yichang Central Peoples Hospital, China. Her current research interests include tumor immunology, tumor-targeted therapy, Bioinformatics. She has published four SCI journal papers.

Xingyu Liao is currently working towards his Ph.D. degree in computer science from the Central South University, Changsha, China. His main research interests include genome assembly and sequence analysis.

Wufei Zhu received his Ph.D. degree in immunology from Norman Bethune College of Medicine, Jilin University, Changchun, China, in 2011. He is currently an associate chief physician in the Department of Endocrinology, China Three Gorges University & Yichang Central Peoples Hospital, China. His current research interests include autoimmune disease of Endocrinology, the effect of AIRE in autoimmune disease and immune tolerance, gene sequencing of AIRE in APS-1 patients. He has published more than 20 papers including six SCI journal papers.

Lu Fang is currently working towards her Masters degree at China Three Gorges University, her main research interests are in autoimmune disease of Endocrinology, include late autoimmune diabetes and the effect of AIRE in autoimmune diseases.

Xing Chen received a Masters degree in internal medicine from Three Gorges University, China, in 2015. He is a resident physician in the Department of Endocrinology, First Clinical Medical School of China Three Gorges University. His current main research interests are in autoimmune disease of Endocrinology, especially the mechanism of AIRE in immune tolerance. He has published several papers including one SCI journal paper.

References

Bao E, Jiang T, Kaloshian I, Girke T (2011) SEED: efficient clustering of next-generation sequences. Bioinformatics 27(18), 2502–2509.

Bruncau M, Mottet T, Moulin S, Kerbiriou M, Chouly F, Chretien S, Guyeux C (2018) A clustering package for nucleotide sequences using Laplacian Eigenmaps and Gaussian Mixture Model. Computers in Biology and Medicine 93, 66–74.

Deorowicz S, Kokot M, Grabowski S, Debudaj-Grabysz A (2015) KMC 2: fast and resource-frugal k-mer counting. Bioinformatics 31(10), 1569–1576.
