fastp: an ultra-fast all-in-one FASTQ preprocessor

Shifu Chen[1,2,*], Yanqing Zhou[1], Yaru Chen[1], Jia Gu[2]

[1]HaploX Biotechnology.

[2]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences.

*To whom correspondence should be addressed.

Abstract:

Motivation: Quality control (QC) and preprocessing of FASTQ files are necessary steps to provide clean data for downstream analysis. Traditionally, for each operation, such as QC, adapter trimming and quality filtering, a different tool is used. These tools are usually not fast enough since they are mostly developed in high-level programming languages like Python and Java, and provide limited multi-threading support. Also, the necessity to read and load data for multiple times makes the preprocessing slow and I/O inefficient.
Results: We developed *fastp* as an ultra-fast FASTQ preprocessor with most useful QC and data filtering features. It can perform quality control, adapter trimming, quality filtering, per-read quality cutting and lots of other operations within a single scan of the FASTQ data. It also supports unique molecular identifier (UMI) preprocessing, poly tail trimming, output splitting, and base correction for paired-end data. It can automatically detect the adapters for both single-end and paired-end FASTQ data. This tool is developed in C++ and has multi-threading support. Based on our evaluation, *fastp* is 2~5 times faster than other FASTQ preprocessing tools like Trimmomatic or Cutadapt, in spite of that *fastp* performs much more operations than the latter ones.
Availability and Implementation: The open-source code and corresponding instructions are available at: https://github.com/OpenGene/fastp
Contact: chen@haplox.com

1. Introduction

Quality control and preprocessing of the sequencing data are essential for obtaining high quality and high confidence variants in down-stream data analysis. The data can have adapter contamination, base content biases, and overrepresented sequences. More seriously, the library preparation and sequencing steps always involve errors and may cause wrong representations of the original nucleic acid sequences. In recent years, sequencing technologies, especially the next-generation sequencing (NGS) technology have been broadly used in clinical applications, especially the noninvasive prenatal testing (NIPT) (Bianchi et al., 2015) and cancer diagnosis. For example, the liquid biopsy technology (Esposito, Criscitiello, Trapani, & Curigliano, 2017), which seeks for cancer-related biomarkers in the circulating system, can be used to help diagnosing cancers and making personalized treatment decisions. As a major technology of liquid biopsy, cell-free tumor DNA (catena) sequencing is used to detect tumor-derived DNA fragments from plasma, urine and other circulating liquids. The

ctDNA sequencing data are usually very noisy and the detected mutations are usually with ultra-low mutation allele frequencies (MAF). Quality control and data preprocessing are especially important for detecting such low-MAF mutations to eliminate false positives and false negatives.

Quality control and preprocessing of FASTQ data might be considered as resolved problems since there are already some existing tools. For examples, FASTQC (Andrews) is a Java-based quality control tool providing per-base and per-read quality profiling features. Cutadapt (Martin, 2011) is a widely used adapter trimmer, which also provides some read filtering features. Trimmomatic (Bolger, Lohse, & Usadel, 2014), as another tool that is widely used for trimming adapters, can also perform quality pruning using algorithms like sliding window cutting. SOAPnuke (Y. Chen et al., 2018) is a recently published tool for adapter trimming and read filtering, with the implementation of MapReduce on Hadoop systems.

However, in the past practice multiple different tools for FASTQ data quality control and preprocessing were used. For example, a typical combination is to use FASTQC for quality control, Cutadapt for adapter trimming and Trimmomatic for read pruning and filtering. The necessity to read and load data for multiple times makes the preprocessing slow and I/O inefficient. The reason of using them in a combination is that there doesn't exist a single tool that can address well all these problems. The authors developed AfterQC (S. Chen et al., 2017) to integrate quality control, adapter trimming, data filtering, and other useful functions in one single tool. AfterQC is a convenient tool that can perform all the necessary operations and output HTML-based reports within a single scan of FASTQ files. It also provides a novel algorithm to correct bases by looking for the overlapping of paired-end reads. However, since AfterQC is developed in Python, it is relatively slow and too time-consuming for processing large FASTQ files.

In this paper, we present *fastp*, an ultra-fast tool to perform quality control, read filtering and base correction for FASTQ data. It covers most features of FASTQC + Cutadapt + Trimmomatic + AfterQC, while runs 2~5 times faster than running anyone of them. Besides the functions that are available in these tools, *fastp* provides additional features like unique molecular identifier (UMI) preprocessing, per-read polyG tail trimming and output splitting. *fastp* provides QC reports for both the data before filtering and after filtering within a single HTML page, which enables us to compare the quality statistics changed by the preprocessing step directly. *fastp* can automatically detect adapter sequences for both single-end and paired-end Illumina data. In contrast with above tools that are developed in Java or Python, *fastp* is developed in C/C++ with solid multi-threading implementation, which makes it much faster than the peers. Furthermore, based on the functions for correcting or eliminating sequencing errors, *fastp* can obtain even better clean data than conventional tools.

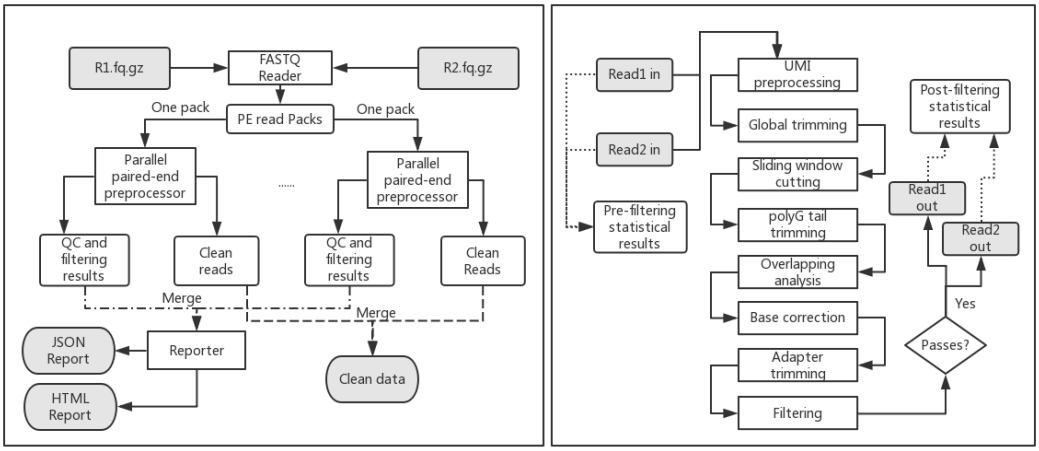
2. Methods

As an all-in-one FASTQ preprocessor, *fastp* provides functions such as quality profiling, adapter trimming, read filtering and base correction. It supports both single-end and paired-end short read data and also provides basic supports for long read data, which are typically generated by PacBio and Nanopore sequencers. In this section, we will first present the overall design of this tool, and then explain how the major modules work.

2.1 Overall design

fastp is designed for multi-threading parallel processing. Reads loaded from FASTQ files will be packed with a size of *N* (*N*=1000). Each pack will be consumed by one thread in the pool, and each read of the pack will be processed. Each thread has an individual context to store the statistical values of the reads it processed, such as per-cycle quality profiles, per-cycle base contents, adapter trimming results and k-mer counts. These values will be merged after all reads are processed, and a reporter will generate reports in both HTML and JSON formats. *fastp* reports the statistical values for both pre-filtering and post-filtering data, so that one can compare how data quality changes after the filtering is done.

fastp supports both single-end and paired-end data. While most steps of SE and PE data processing are similar, PE data processing requires some additional steps like overlapping analysis. For the sake of simplicity, we only demonstrate the main workflow of paired-end data preprocessing, which is shown in Fig.1.



(a) main workflow (b) paired-end preprocessor

Fig. 1 Main workflow of paired-end data processing (a) and the paired-end preprocessor of one read pair (b). In the main workflow, a paired of FASTQ files are loaded and packed, and each read pair will be processed individually in the paired-end preprocessor, which is described in (b).

2.2 Adapter trimming

fastp supports automatic adapter trimming for both single-end and paired-end Illumina data and uses different algorithms for each of these tasks. For single-end data, the adapter sequences are detected by assembling the high frequency read tails, while, for paired-end data, the adapter sequences are detected by finding the overlapping of each pair.

The adapter sequence detection algorithm is based on two assumptions. The first assumption is that there is only one adapter in the data; the second one is that adapter sequences are only in the read tails. These two assumptions are valid for major next-generation sequencers like Illumina HiSeq series, NextSeq series and NovaSeq series. We first count the last 10bp of N reads (N = 1M), and record the 10bp sequences with frequency of occurrence > 0.0001. The low-complexity sequences are removed since they are usually caused by sequencing artifacts. A simple assembly algorithm is applied to recover adapter sequence from the 10bp sequences set S, which is described by pseudo code in Algorithm 1.

Algorithm 1: adapter sequence detection for single-end data

```
while true:
    changed = false
    foreach (s1, s2) in S:
        if overlapped(s1, s2) and offset(s1,s2)==1:
            ms = merge(s1,s2)
            S.add(ms); S.remove(s1); S.remove(s2)
            changed = true
    if changed == false:
        break
a = S.longest()
if a.length() > Threshold:
    detected adapter a
else:
    detected no adapters
```

For paired-end data, *fastp* seeks for an overlapping of each pair, and considers the bases that fall out of the overlapped regions as adapter contents. The algorithm of overlapping detection is derived from our previous work AfterQC. Comparing to sequence matching based adapter-trimming tools like Cutadapt and Trimmomatic, an obvious advantage of overlapping analysis based method is that it can trim the adapters with very few bases present in the read tail. For example, most sequence matching based tools require a hatchment of at least three bases, so it cannot trim adapters with only one or two bases. In contrast, *fastp* can trim adapters with even only one base in the tail.

Although *fastp* can detect adapter sequences automatically, it still provides interfaces to set specified adapter sequences for trimming. For SE data, if the adapter sequence is given, then automatic adapter sequence detection will be disabled. For PE data, the adapter sequence will be used for sequence matching based adapter trimming only when *fastp* fails to detect a good overlap of this pair.

2.3 Base correction

For paired-end data, if one pair of reads can be detected with a good overlap, the bases within the overlapped region can be compared. If the reads are of high quality, they are usually completely reverse-complemented.

If some mismatches are found within the overlapped region, *fastp* will try to correct the

mismatches. *fsatp* only corrects the mismatched base pair with imbalanced quality score, which means one of them is with high quality score (>Q30) while the other is with low quality score (<Q15). To reduce false correction, fastp only performs a correction if the total mismatch number is less than a threshold T (T = 5).

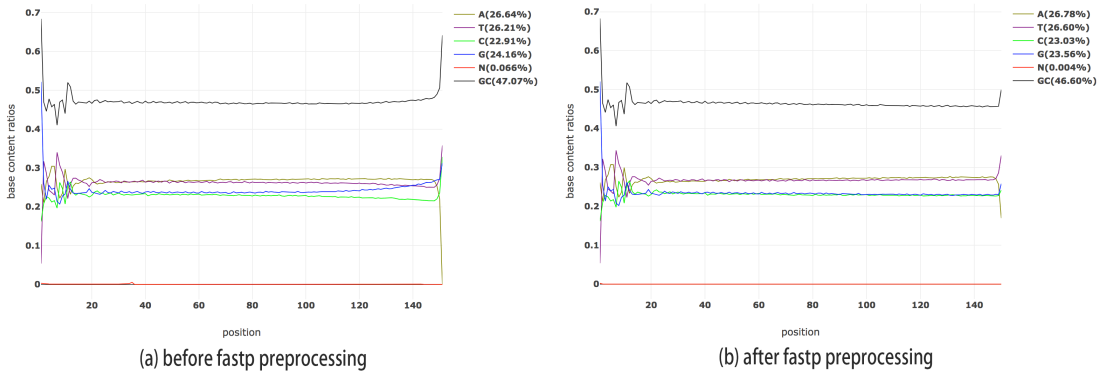
2.4 Sliding window quality cutting

To improve the read quality, *fastp* supports a sliding window method to drop the low quality bases of each read's head and tail. The window can slide from the 5' to 3', or from 3' to 5', and the average quality score within the window is evaluated. If the average quality is lower than a threshold, the bases in the window will be marked as discarded and the window will be moved forward by one base, otherwise the algorithm stops.

2.5 polyG tail trimming

PolyG is a common issue that can be observed from Illumina NextSeq and NovaSeq series since they are based on two-color chemistry. Such systems use two different lights (i.e. red and green) to represent four bases, one base with only red light signal detected will be called as C, and one base with only green light signal detected will be cased as T. For one base with both red and green light detected, it will be called as A, and for the base with no light detected, it will be called as G. However, as the sequencing by synthesis (SBS) goes to later cycles, the signal strength of each DNA cluster becomes weaker. This issue will cause some T and C be wrongly interpreted as G in the read tails, and this problem is called polyG tail.

fastp can detect and trim the polyG in the read tails. It checks the flow cell identifier to determine whether the data is from Illumina NextSeq or NovaSeq sequencers, and if so automatically enables polyG tail trimming. PolyG tail issue can cause serious base content separation problem, which means A and T, or C and G have much different base content ratio. Fig. 2 shows an example of data with polyG tail issue and how it is addressed with *fastp* preprocessing.



(a) before fastp preprocessing (b) after fastp preprocessing

Fig. 2. The base content ratio curves generated by *fastp* for one Illumina NextSeq FASTQ file. (a) before *fastp* preprocessing, and (b) after *fastp* preprocessing. From (a), we can find that the G curve is abnormal, and G/C curves are separated. From (b), we can find that the G/C separating problem is eliminated.

2.6 UMI preprocessing

Recently, a unique molecular identifier (UMI) technology is proposed to reduce the background noise and improve sensitivity for detecting ultra-low frequency mutations in deep sequencing applications (i.e. ctDNA sequencing). UMI method can be used to remove duplications and generate high quality consensus reads. It has been adopted by a lot of sequencing methods like Duplex-Sew (Kennedy et al., 2014) and iDES (Newman et al., 2016). For Illumina sequencing platforms, UMI can be integrated in sample index or inserted DNA. UMI should be shifted to the read identifier for it to be kept by alignment tools like BWA (Li & Durbin, 2009) or Bowtie (Langmead & Salzberg, 2012).

Some tools have already been developed for preprocessing UMI integrated FASTQ data, such as UMI-tools (Tom Sean Smith, 2018) and umis (Valentine Svensson, 2018). However, these tools are not efficient enough, and require an individual execution that consumes more I/O and computational resource. *fastp* supports UMI preprocessing with very little overhead. It supports UMI in either sample index or inserted DNA, or both. Comparing to the UMI-tools or umis, *fastp* runs about 3x faster even when *fastp* performs other tasks (i.e. QC and filtering) simultaneously. The performance evaluation result will be shown in next section.

2.7 Output splitting

Parallel processing for NGS data has become a new trend, especially in the cloud-computing environment. In a typical parallel NGS data processing pipeline, an original FASTQ file will be split to multiple pieces, and each piece will be ran with aligners and alignment adjustment tools to obtain its BAM file. These BAM files can be merged to be different forms for parallel variant calling.

fastp supports two splitting modes: splitting by file lines and splitting by file numbers. The latter one is more complicated since *fastp* has to evaluate the total lines of the input files, which is especially nontrivial for GZIP compressed data. *fastp* evaluates the total lines of input file by comparing the stream size of the first 1M reads.

2.8 Overrepresented sequence analysis

Some sequences or even entire reads can be overrepresented in FASTQ data. Analysis of these overrepresented sequences can give an overview of some sequencing artifacts like PCR over duplication, polyG tails and adapter contamination. FASTQC has provided an overrepresented sequence analysis module. However, according to the author's introduction, FASTQC only tracks the first 1M reads of the input file to conserve memory. We suggest that inferring the overall distribution from the first 1M reads is not a solid solution since the starting reads in Illumina FASTQ data usually originate from the edges of flowcell lanes, which may have lower quality and different patterns from the overall distribution.

Different from FASTQC, *fastp* samples all reads evenly to evaluate the overrepresented sequences to eliminate the partial distribution bias. To achieve efficient implementation of this feature, we designed a two-step method. In the first step, *fastp* completely analyzes the first 1.5M base pairs of the input FASTQ, to obtain a list of sequences with relatively high

occurrence frequency in different sizes. In the second step, *fastp* samples the entire file and counts the occurrence of each sequence. Finally, the sequences with high occurrence frequency will be reported.

Besides the occurrence times, *fastp* also records the occurrence positions of overrepresented sequences. This information is quite useful for the diagnosis of sequence quality issues. Some sequences tend to appear in the read head, while some sequences tend to appear in the read tail. The distribution of overrepresented sequences is visualized in the HTML report. Fig. 3 shows a demonstration of overrepresented sequence analysis result.

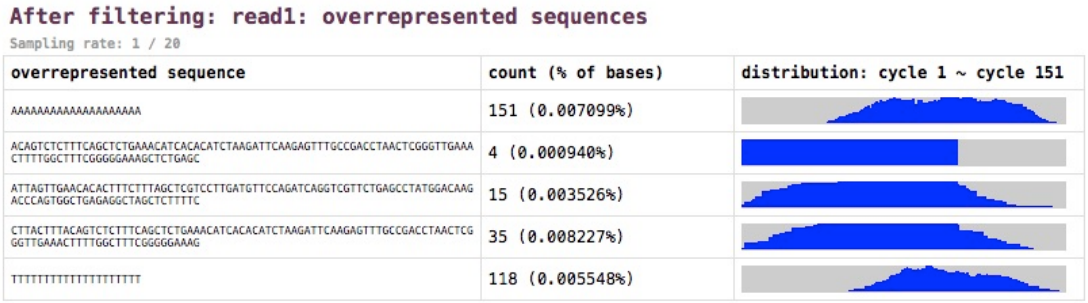


Fig. 3. The result of overrepresented sequence analysis. The right column shows the histogram of occurrence among all the sequencing cycles.

2.9 Quality control and reporting

fastp supports filtering reads by low quality base percentage, N base number and read length. The filtering process is trivial so it is not described here. *fastp* records the number of reads that were filtered out according to different filtering criteria.

fastp provides comprehensive information for the quality profiling results. Different from FASTQC, *fastp* provides results for both pre-filtering data and post-filtering data. This enables the possibility to evaluate filtering effect by comparing the figures directly.

fastp reports the result in both JSON and HTML format, while the JSON report contains all the data visualized in the HTML report. The format of the JSON report is manually optimized to be easily readable by humans. The HTML report is a single standalone web page, with all figures created dynamically using JavaScript and web canvas. It is worth mentioning that *fastp* provides a full k-mer occurrence table for all 6-bp sequences. An online demonstration of the HTML report can be found at: http://opengene.org/fastp/fastp.html

3. Results

We conducted several experiments to evaluate the performance of *fastp* in both speed and quality. We chose FASTQC, Cutadapt, Trimmomatic, SOAPnuke and AfterQC for performance comparison, and the results showed that *fastp* is much faster than these tools, while providing similar or even better quality.

3.1 Speed evaluation

We compared the speed of all the six tools by preprocessing the B17NCB1 dataset, which is provided by National Center for Clinical Laboratories (NCCL) of China. This

dataset is paired-end, containing 9,316 M bases. We evaluated the used time for both paired-end (PE) and single-end mode. All tools were ran with single thread for fairer comparison. The result is shown in Table 1.

Table 1. Speed comparison of *fastp* and other software.

Tools	PE / SE	Time (min)	Throughput (read/s)
fastp	SE	6	129397.9
	PE	13.3	116750.0
FASTQC	SE	13	59722.1
	PE	25.8	60185.1
Cutadapt	SE	18	43132.6
	PE	24.6	63120.9
SOAPnuke	SE	30.7	25289.5
	PE	32.5	47777.7
AfterQC	SE	25.2	30809.0
	PE	57.2	27146.4
Trimmomatic	SE	31.9	24338.2
	PE	60.9	25497.1

From Table 1, we can see that *fastp* is much faster than other tools. The tool with second speed is FASTQC, which takes about 2x the time of *fastp*. However, FASTQC only performs quality control, while *fastp* performs quality control (for both pre-filtering data and post-filtering data), data filtering and other operations. The other tools take 3x~5x time of *fastp*. Since *fastp* is natively designed for multi-thread processing, it may show even higher performance when executed in real applications in multi-thread mode. Since some tools do not support multi-threading, multi-threading performance comparison is not provided here.

3.2 Quality evaluation

To evaluate the adapter trimming and quality pruning of *fastp* comparing to other tools (AfterQC, SOAPnuke, Trimmomatic, Cutadapt), we used an Illumina NextSeq PE150 dataset (S017). Among these tools, *fastp* and AfterQC can trim adapters by overlapping analysis, while the other tools require input of adapter sequences. We evaluated the suspected adapters by searching adapter sequences from the post-filtering data. The comparison result is shown in Table 2.

Table 2. The comparison of adapter trimming and N base numbers.

S017	Base (M)	N Base	Reads (M)	Suspected adapter
Raw data	3401	1162348	22.5	4158940
fastp	3108	7876	21.3	607
AfterQC	3059	7319	21.3	81924
SOAPnuke	2992	19687	19.8	1861641
Trimmomatic	3117	59441	22.5	320
Cutadapt	3298	1161235	22.5	505

From Table 2, we can find that the suspected adapter in the *fastp*-filtered data is only

slightly higher than Trimmomatic and Cutadapt. This little difference may be caused by the different thresholds used in these tools. SOAPnuke gave the worst result of adapter trimming since it required at least half-length hatchment of the adapter sequences. From another aspect, we can find that the post-filtering data of *fastp* and AfterQC contain much less N base content than other tools, indicating that these two tools performed higher quality filtering than Trimmomatic and Cutadapt.

To further evaluate the effectives of filtering, we mapped the data filtered by different tools to the reference genome hg19 using BWA-MEM, and evaluated the mapping results using Samtools (Li et al., 2009). The mismatches, clips and improper mappings were recorded for the purpose of evaluation. In our view, uncorrected sequencing errors contribute majorly to the mismatches, while residual adapters contribute majorly to the clips and improper mappings. The comparison result is shown in Table 3.

Table 3. Mismatches, clips and single read maps of the data filtered with different tools.

S017	Total Map Base (M)	Mismatch Base (M)	Total Map Read (M)	Clip Read (M)	Single Read Map
Raw data	3390	19.8	22.4	6.16	46025
fastp	3107	10.7	21.2	0.31	287
AfterQC	3053	12.3	21.2	0.64	34099
SOAPnuke	2981	18.3	19.7	3.46	36888
Trimmomatic	3111	12.8	22.1	0.75	229111
Cutadapt	3287	19.8	22.4	1.05	46195

From Table 3, we can learn that *fastp* generated the lowest number of mismatches, clipped reads and single read mapped reads. Trimmomatic and Cutadapt generated much more clipped or single read mapped reads. Since Trimmomatic, Cutadapt and SOAPnuke are all based on adapter sequence matching, they may fail to detect the adapters when the adapter sequence is with only a few bases. For example, Cutadapt requires at least 3bp matching of the adapter sequence and the read for it to be recognized as an adapter. If the adapter is with only one or two bases, they won't be detected, and usually will be reported as a mismatch or soft clip by the alignment tools.

3.3 UMI evaluation

Unique molecular identifier (UMI) technology is widely used in cancer sequencing, especially the ctDNA sequencing. To analyze the NGS data with UMI integrated, the FASTQ preprocessor should shift the UMI from the reads to the read identifiers. We ran the UMI preprocessing of a FASTQ of 4Gb Illumina PE150 data, with *fastp*, UMI-tools and umis. The execution time was recorded and the result is shown in Table 4.

Table 4. UMI preprocessing time comparison of *fastp*, umis and UMI-tools.

Tools	Time (min)	Throughput (read/s)
fastp	4.6	104302.9

umis	12.43	38599.6
UMI-tools	28.38	16906.4

From Table 4 we can see that *fastp* is about 2.7x faster than umis, and about 6.1x faster than UMI-tools. This evaluation was conducted with GZIP input and uncompressed output since umis doesn't support GZIP output. Since *fastp* can achieve high performance for UMI preprocessing, recently it has been adopted by the popular NGS pipeline framework bcbio-nextgen (Brad Chapman, 2018).

4. Discussion

In this paper, we introduced *fastp*, an ultra-fast all-in-one FASTQ preprocessor. *fastp* is a versatile tool that can perform quality profiling, read filtering, read pruning, adapter trimming, polyG tail trimming, UMI preprocessing and other operations within a single scan of FASTQ files. Additionally, it can split the output to multiple files for parallel processing.

We have evaluated the performance of speed and quality of *fastp* against other tools. The evaluation result showed that *fastp* is much faster than the peers, and provided the highest quality of data filtering.

fastp is an open-source software. Due to its high speed and good quality for FASTQ file quality control and filtering, *fastp* has gained a lot of community users.

5. Acknowledgement

The authors would like to thank the *fastp* community users for their good catches of bugs and valuable advices.

6. Funding

This study was financed partially by National Science Foundation of China (NSFC Project No.61472411), and Special Funds for Future Industries of Shenzhen (Project No. JSGG20160229123927512)

Reference

Andrews, S. A quality control tool for high throughput sequence data. http://www.bioinformatics.bbsrc.ac.uk/projects/fastqc/.

Bianchi, D. W., Chudova, D., Sehnert, A. J., Bhatt, S., Murray, K., Prosen, T. L., . . . Halks-Miller, M. (2015). Noninvasive Prenatal Testing and Incidental Detection of Occult Maternal Malignancies. *JAMA, 314*(2), 162-169. doi:10.1001/jama.2015.7120

Bolger, A. M., Lohse, M., & Usadel, B. (2014). Trimmomatic: a flexible trimmer for Illumina sequence data. *Bioinformatics, 30*(15), 2114-2120. doi:10.1093/bioinformatics/btu170

Brad Chapman, R. K., Lorena Pantano et al. (2018). Validated, scalable, community developed variant calling, RNA-seq and small RNA analysis. https://github.com/chapmanb/bcbio-nextgen.

Chen, S., Huang, T., Zhou, Y., Han, Y., Xu, M., & Gu, J. (2017). AfterQC: automatic filtering, trimming, error removing and quality control for fastq data. *BMC Bioinformatics, 18(Suppl 3)*(80), 91-100. doi:10.1186/s12859-017-1469-3

Chen, Y., Chen, Y., Shi, C., Huang, Z., Zhang, Y., Li, S., . . . Chen, Q. (2018). SOAPnuke: a MapReduce acceleration-supported software for integrated quality control and preprocessing of high-throughput sequencing data. *Gigascience, 7*(1), 1-6. doi:10.1093/gigascience/gix120

Esposito, A., Criscitiello, C., Trapani, D., & Curigliano, G. (2017). The Emerging Role of "Liquid Biopsies," Circulating Tumor Cells, and Circulating Cell-Free Tumor DNA in Lung Cancer Diagnosis and Identification of Resistance Mutations. *Curr Oncol Rep, 19*(1), 1. doi:10.1007/s11912-017-0564-y

Kennedy, S. R., Schmitt, M. W., Fox, E. J., Kohrn, B. F., Salk, J. J., Ahn, E. H., . . . Loeb, L. A. (2014). Detecting ultralow-frequency mutations by Duplex Sequencing. *Nat Protoc, 9*(11), 2586-2606. doi:10.1038/nprot.2014.170

Langmead, B., & Salzberg, S. L. (2012). Fast gapped-read alignment with Bowtie 2. *Nat Methods, 9*(4), 357-359. doi:10.1038/nmeth.1923

Li, H., & Durbin, R. (2009). Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics, 25*(14), 1754-1760. doi:10.1093/bioinformatics/btp324

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., . . . Genome Project Data Processing, S. (2009). The Sequence Alignment/Map format and SAMtools. *Bioinformatics, 25*(16), 2078-2079. doi:10.1093/bioinformatics/btp352

Martin, M. (2011). Cutadapt removes adapter sequences from high-throughput sequencing reads. *EMBnet journal, 17*(1), 10-12.

Newman, A. M., Lovejoy, A. F., Klass, D. M., Kurtz, D. M., Chabon, J. J., Scherer, F., . . . Alizadeh, A. A. (2016). Integrated digital error suppression for improved detection of circulating tumor DNA. *Nat Biotechnol, 34*(5), 547-555. doi:10.1038/nbt.3520

Tom Sean Smith, A. H. a. I. S. (2018). UMI-tools: Modelling sequencing errors in Unique Molecular Identifiers to improve quantification accuracy. *Genome Research*.

Valentine Svensson, R. K. e. a. (2018). Tools for processing UMI RNA-tag data. https://github.com/vals/umis.