BIOINFORMATICS ORIGINAL PAPER

Vol. 27 no. 18 2011, pages 2502–2509
doi:10.1093/bioinformatics/btr447

Sequence analysis

Advance Access publication August 2, 2011

SEED: efficient clustering of next-generation sequences

Ergude Bao[1], Tao Jiang[1], Isgouhi Kaloshian[2] and Thomas Girke[3,*]

[1]Department of Computer Science and Engineering, [2]Department of Nematology and [3]Department of Botany and Plant Sciences, University of California, Riverside, CA 92521, USA

Associate Editor: Alex Bateman

ABSTRACT

Motivation: Similarity clustering of next-generation sequences (NGS) is an important computational problem to study the population sizes of DNA/RNA molecules and to reduce the redundancies in NGS data. Currently, most sequence clustering algorithms are limited by their speed and scalability, and thus cannot handle data with tens of millions of reads.

Results: Here, we introduce SEED—an efficient algorithm for clustering very large NGS sets. It joins sequences into clusters that can differ by up to three mismatches and three overhanging residues from their virtual center. It is based on a modified spaced seed method, called block spaced seeds. Its clustering component operates on the hash tables by first identifying virtual center sequences and then finding all their neighboring sequences that meet the similarity parameters. SEED can cluster 100 million short read sequences in <4 h with a linear time and memory performance. When using SEED as a preprocessing tool on genome/transcriptome assembly data, it was able to reduce the time and memory requirements of the Velvet/Oasis assembler for the datasets used in this study by 60–85% and 21–41%, respectively. In addition, the assemblies contained longer contigs than non-preprocessed data as indicated by 12–27% larger N50 values. Compared with other clustering tools, SEED showed the best performance in generating clusters of NGS data similar to true cluster results with a 2- to 10-fold better time performance. While most of SEED's utilities fall into the preprocessing area of NGS data, our tests also demonstrate its efficiency as stand-alone tool for discovering clusters of small RNA sequences in NGS data from unsequenced organisms.

Availability: The SEED software can be downloaded for free from this site: http://manuals.bioinformatics.ucr.edu/home/seed.

Contact: thomas.girke@ucr.edu

Supplementary information: Supplementary data are available at *Bioinformatics* online

Received on April 11, 2011; revised on July 11, 2011; accepted on July 23, 2011

1 INTRODUCTION

In recent years, the data volumes generated by next-generation sequencing (NGS) technologies have been growing at a pace that has now begun to greatly challenge the data processing and storage capacities of modern compute systems (Medini *et al.*, 2008). Only 4 years ago, NGS technologies like Illumina's reversible terminator method or ABI's ligation approach created ~1 billion bases of DNA sequence information per instrument run which has now increased to over 300 billion bases per run with even shorter turnaround times (Holt and Jones, 2008). This corresponds approximately to a 4-fold increase of sequence data output per year. As a result of this rapid improvement of the technology, many exciting sequence-based research applications have evolved recently. These include genome resequencing of entire organism populations, personalized medicine, RNA-Seq, ChIP-Seq and many others (1000 Genomes Project Consortium *et al.*, 2010; Jothi *et al.*, 2008). Processing and storing the large amounts of data produced by these technologies is a major challenge for modern genome research. Thus, it is important to develop methods that can improve the efficiency of the analysis workflows for NGS data. To mention just a few, these include algorithms for processing the data more time and space efficiently (Langmead *et al.*, 2009; Li and Durbin, 2009a; Li *et al.*, 2009b) as well as data reduction approaches that aim to retain only the scientifically relevant and non-redundant information from NGS projects rather than everything (Leinonen *et al.*, 2010). For example, in genome resequencing projects one can greatly reduce the dataset sizes by storing only genetic variations, while removing the bulk of the sequence information that only confirms what is already known (Fritz *et al.*, 2011). Similarly, in quantitative NGS experiments for profiling pools of mRNAs, small RNAs or protein–DNA interactions one can convert the data to much less storage intensive tag counts at an early stage of the analysis workflow. Solutions that prevent or greatly minimize information loss are always preferred. However, with the current growth rates of NGS data many of them may soon become impractical, especially when the data sizes become the main time and financial bottleneck for conducting scientific experiments in the NGS field.

This study introduces a new algorithm capable of clustering NGS sets in size ranges of several hundred million entries using a modified spaced seed method (Lin *et al.*, 2008; Ma *et al.*, 2002). This method, hereafter referred to as SEED, efficiently joins sequences into clusters with user-definable similarity parameters ranging from 0 to 3 mismatches and overhanging ends with up to 3 nt in length. These mismatch features are important to make the method less sensitive to base call errors, imprecise molecular cleavage events or inaccurate adaptor trimming. The main utilities of SEED are the identification, enumeration and removal of redundant sequences in NGS data. In its current implementation, SEED is designed to function as a short read clustering tool with controllable mismatch parameters, but not as an error corrector like FreClu (Qu *et al.*, 2009). There are several practical applications of this clustering approach. First, the method can be used to reduce the complexity in NGS data by collapsing redundant reads to a single

*To whom correspondence should be addressed.

center sequence along with its frequency information. While this data reduction step results only in a minor information loss, it can greatly improve the run time, memory requirements, and quality of genome and transcriptome assemblies. Second, it can be used to determine the sequence diversity in quantitative NGS profiling datasets, such as RNA-Seq and ChIP-Seq, by enumerating very similar reads. The resulting numbers of unique versus redundant reads can be an important parameter for identifying technical problems in these datasets (e.g. low reproducibility due to bias in PCR amplification steps). Third, the method can be applied to discover clusters of microRNAs (miRNAs) directly from NGS data without the requirement of mapping the reads to a reference genome which is particularly important when working with unsequenced organisms (Johnson *et al.*, 2009; Montgomery *et al.*, 2008).

While in the past decade there has been extensive research on sequence family clustering for handling datasets in the range of hundreds of thousand entries (e.g. Li and Godzik, 2006), there has been very limited development of methods for clustering the much larger sequence volumes from NGS experiments with hundreds of millions of entries. The short list of tools capable of clustering data sizes in the range of at least several million sequences includes UCLUST and FreClu (Edgar, 2010; Qu *et al.*, 2009). Most other clustering tools in this area are designed to solve problems related to EST analysis, such as pre-clustering of ESTs to facilitate their downstream assemblies (Hazelhurst *et al.*, 2008; Huang and Madan, 1999; Picardi *et al.*, 2009; Rao *et al.*, 2010).

In the following, we first describe the theory behind the SEED clustering algorithm as well as the design of its software implementation. We then illustrate and discuss its time, memory and accuracy performance by using both simulated and real NGS datasets. The real datasets were specifically chosen to evaluate the algorithm's efficiency for several application areas, including complexity reduction of RNA-Seq profiling experiments in the absence of a reference, prediction of mature miRNAs, and transcriptome and genome assemblies.

2 METHODS

2.1 Overview of the algorithm

To cluster NGS by similarity, SEED indexes the reads by using the open hashing technique and a special class of spaced seeds (Lin *et al.*, 2008), called *block spaced seed*. Once the reads are stored in hash tables, SEED clusters them by first creating a *virtual center sequence* for each cluster and then finding all the reads that are within a certain similarity threshold to the center sequence. The following is a short overview of the algorithm. More details are provided in the next subsections.

A. Indexing

(1) Initialize the indexing if the longest and the shortest read sequences do not differ by more than five bases in length.

(2) Use the first seed in a chosen set of block spaced seeds to hash the sequences into a hash table.

(3) Repeat step A.2 with each block spaced seed of the set and store their results in separate hash tables.

B. Clustering

(1) Select an arbitrary sequence, identify for it all sequences within twice the mismatch threshold and compute their virtual center sequence.

(2) Find for the virtual center sequence all sequences with the allowed number of overhanging bases and mismatches. Then remove these sequences from the hash tables.

(3) Repeat steps B.1 to B.2 until the hash tables are empty.

2.2 Indexing and hash tables

Spaced seeds were introduced by Ma *et al.* (2002) as a time-efficient method for sequence similarity searching. Several NGS alignment tools are based on this method. These include Eland (Anthony J. Cox, unpublished data), MAQ (Li *et al.*, 2008), SeqMap (Jiang and Wong, 2008) and ZOOM (Lin *et al.*, 2008). The general framework of spaced seeds can be summarized as follows. A spaced seed of length l is a binary string of l bits. When the seed is used in matching a query string of length l with another string, the bit 1 demands a match while the bit 0 tolerates a mismatch. Such a seed can also be conveniently used to index sequences of length l in hashing. For example, the spaced seed '01110' will file the sequences 'CAAAG' and 'TAAAA' into the same bucket, as well as all other 5mers with an 'AAA' in the middle. The weight w of a spaced seed is its number of 1's. It directly affects the size of the hash tables in the above indexing scheme, and thus memory usage. The parameter k is usually a predefined value, and the size of a set of spaced seeds is denoted as c. The details of designing a set of spaced seeds with full search sensitivity for given values of l,w,k will be discussed in Section 2.4.

The hash table data structure used in SEED is shown in Supplementary Figure S1. Each hash table corresponds to a spaced seed, and each bucket in it corresponds to a word of w bases. A bucket consists of a header and a dynamically allocated array of pointers. The header points to an array, and each pointer in the array references a sequence. During the clustering process, a tag will be assigned to the pointers where the sequences have been assigned to clusters to indicate their removal from the hash tables. In addition, there is an array of unsigned integers (not shown in Supplementary Figure S1) for storing the number of pointers in each bucket. Suppose that n is the total number of sequences. The memory usage s in bytes B on a 64-bit machine can be estimated as follows:

$$s=3\times 4^{w+1}c+4nc+\left(\lceil\frac{l}{4}\rceil+1\right)n \tag{1}$$

In Supplementary Figure S1, from left to right, the headers take $4^{w}\times c\times 8B=2\times 4^{w+1}cB$ memory, where $8B$ is the memory required for a pointer on a 64-bit machine in a straightforward implementation. However, integer offsets can be used instead of real pointers to reduce the memory footprint of a pointer to $4B$. The c hash tables take $n\times c\times 4B=4ncB$ memory. The memory requirement for storing the sequences themselves is $n\times\lceil\frac{l}{4}\rceil B=n\lceil\frac{l}{4}\rceil B$ and nB for the tags. In addition, the array for storing the number of items in each bucket takes $4^{w}\times c\times 4B=4^{w+1}cB$ memory. Combined together, the total memory required is $3\times 4^{w+1}c+4nc+\lceil\frac{l}{4}\rceil nB$. For example, if there are 1 million sequences of 36 bp, $w=12$ and $c=10$, then the memory requirement totals: $s=3\times 4^{12+1}\times 10+\ 4\ \times 1M\times 10+(\lceil\frac{36}{4}\rceil+1)\times 1\mathrm{MB}=1970\mathrm{MB}$.

2.3 Design of block spaced seed set

While other spaced seeds methods are more common, especially in the NGS alignment field, we have chosen *block spaced seeds* for NGS clustering, because they are conceptually simple and easy to optimize.

DEFINITION 1. *A block spaced seed is a binary string consisting of a sequence of blocks of equal length, where each block contains either all 0's or all 1's.*

The seed sets used by various short read alignment tools are usually heuristic designs. With the exception of ZOOM, they provide suboptimal solutions, but with good performance in practice. Typically, their seed sets are often the outcome of manual optimization procedures for a given read length and number of mismatches. In contrast to this, an optimal set of *block spaced seeds* for a given read length and number of mismatches can be automatically identified with Algorithm 1 (see below). Note that such an optimal set of block spaced seeds typically represents a suboptimal solution for general spaced seeds. We first state a theorem upper bounding the size of an optimal block spaced seed set. The proof of the theorem and the analysis of Algorithm 1 are both available in the Supplementary Materials.

THEOREM 1. *For any given l, w, k, there exists a set of block spaced seeds with length l and weight w that guarantees full search sensitivity with respect to k mismatches if and only if $k \leq \frac{l}{gcd(l,w)} - \frac{w}{gcd(l,w)}$, where $gcd(l,w)$ denotes the greatest common divisor of l and w. Moreover, for any $k \leq \frac{l}{gcd(l,w)} - \frac{w}{gcd(l,w)}$, $\binom{\frac{w}{gcd(l,w)}+k}{k}$ block spaced seeds of length l and weight w would suffice to guarantee full search sensitivity with respect to k mismatches.*

Algorithm 1 BestSeedSet(l, k)

```
$m = \infty$
for $w = 13$ to $11$ do
    if $k \leq \frac{l}{gcd(l,w)} - \frac{w}{gcd(l,w)}$ then
        $c = \binom{\frac{w}{gcd(l,w)}+k}{k}$
        if $m > 2 \times 4^{w+1} c$ then
            $m = 2 \times 4^{w+1} c$
            $w_0 = w$
            $c_0 = c$
        end if
    end if
end for
generate block spaced seed set
return $w_0$ and $c_0$
```

Although it is desirable to maximize w in order to be time efficient, the memory complexity given in Equation (1) suggests that we should minimize w (and c) in order to be memory efficient. Therefore, we should seek a balance between time and space. Supplementary Table S1 shows the memory usages (headers only), seed weights and numbers of seeds required for several read lengths ranging from 25 to 35, where the seed weights and numbers of seeds for each read length are calculated using Algorithm 1 and the memory usages calculated using Equation (1) with similarity threshold $k = 3$. Clearly, if a set of block spaced seeds guarantees full sensitivity for read sequences of length l, then it also guarantees full sensitivity for sequences of length more than l. Moreover, we can always pad spaced seeds with 0's so they have the same length as the reads. Thus, for a specific pair of weight w and number c, the length l listed in the table should be regarded as the minimum read length that w and c support. Since a row with a small l, large w, small c, and small memory usage s is desirable, we choose the row with $l = 30$, $w = 12$, $c = 10$ and $s = 1.25$ GB in our experiments (where the reads are 36 bp long, up to three overhanging bases are allowed on each side and up to three mismatches are tolerated). Supplementary Table S2 lists the 10 block spaced seeds used in our experiments.

2.4 Clustering

The actual sequence clustering component of SEED is an iterative process consisting of three major steps. First, an arbitrary sequence x is selected and hashed using each block spaced seed to locate c buckets. The sequences in the c buckets with at most fk mismatches to the sequence x are identified by a simple Hamming distance calculation, where k is the maximum number of mismatches allowed in a cluster, and f is set to be 2 as a factor of k. The consensus of the resulting sequence set is computed to obtain a virtual center sequence. Second, the virtual center sequence is hashed using each block spaced seed, and the sequences from all the resultant buckets are retrieved. A cluster is formed to include all the sequences with $\leq k$ mismatches to the virtual center sequence. The clustered sequences are removed from the hash tables. Third, to also include sequences that largely overlap with x but with overhanging ends, the virtual center sequence is shifted (actually, rotated) to the left and to the right within the maximum allowed shift distance (predefined value from 0 to 3). For each shifted center sequence, all sequences in the hash tables are added to the cluster that are within k mismatches to the center and then they are also deleted from the hash tables. The above steps are repeated until all sequences have been assigned to clusters and deleted from the hash tables.

Our choice of $f = 2$ in the initial clustering (step one) is based on the following considerations. Given a cluster of sequences with $\leq k$ mismatches to its center, an arbitrarily selected sequence in the cluster has $\leq 2k$ mismatches to any sequence in the set. With this setting, the method can collect all sequences belonging to a cluster even if the randomly chosen seed sequence is far away from the true center of a cluster. The final virtual center sequence—generated from this candidate set—will then provide a reasonable approximation of the true center.

After the clustering, each sequence will be part of a cluster with one or more members. The final results are stored in two cluster result files. One tabular file lists the complete set of reads with their corresponding cluster identifiers. The second file is the clustered FASTQ file containing, for each cluster, only its center sequence along with the corresponding quality scores (see below).

2.5 Incorporating base calling quality values

NGS data contain base calling quality information usually in the form of Phred scores (Cock *et al.*, 2010). To incorporate this quality information into the clustering process, the SEED algorithm allows the user to specify two optional quality value (QV) constraints. The first constraint QV1 specifies when a mismatch should be ignored. That is, a mismatch is ignored if and only if the sum of the Phred scores of the two mismatching bases is lower than the specified QV1 threshold value. The second constraint QV2 specifies when mismatches should be regarded as critical difference in clustering. That is, two sequences are joined in a cluster only if the sum of the Phred scores of all their mismatching bases is below the $QV2$ threshold value. Therefore, $0 \leq QV1 \leq 93 \times 2$ and $0 \leq QV2 \leq 93 \times 6$ in this article since our similarity threshold allows at most three mismatches (Cock *et al.*, 2010). Note that using SEED with the QV information results in a larger memory footprint, because the Phred scores of all sequences need to be read into memory. Since filtering the sequences by quality prior to the clustering may be often an attractive alternative, QV is an optional parameter in the SEED program.

2.6 SEED system design

2.6.1 General features SEED has been implemented in C++ as a stand-alone cross-platform tool for Linux, OS X and Windows operating systems. It expects sequences formatted in standard FASTQ format. It can be run in the three modes *ordinary*, *fast* and *short*. The ordinary mode uses block spaced seeds of weight 12 as listed in Supplementary Table S2 and supports read sequences of length 36–100 bp. The fast mode uses block spaced seeds of weight 13 and supports sequences of length 58–100 bp. The short mode uses block spaced seeds of weight 6 and supports sequences as short as 21 bp. The fast mode provides the fastest processing time, but requires long sequences and slightly more memory than the ordinary mode. The short mode is suitable for small datasets of short sequences like miRNA sequences, but it is slower than the ordinary mode. The default setting is the ordinary mode.

2.6.2 Performance optimization To optimize the time and memory performance of SEED, we have implemented the following features.
Memory performance

- Each base stored in memory corresponds to two bits.
- Only one copy of each sequence is stored in memory, while the hash tables store pointers to all duplicates.
- The pointers are integer offsets, requiring 4 bytes each instead of 8 bytes on a 64-bit machine.

Time performance

- A garbage collection is performed in short intervals to prevent long chaining events. Pointers to already processed sequences that have been assigned to clusters are discarded.
- A different set of block spaced seeds of weight 13 is used in the fast mode for sequences of lengths >58 bp. The 1's in the spaced seeds are positioned as close to the 3′ ends as possible. The latter results in more evenly distributed sequences in the hash table and reduces the bucket sizes. This is important because the read quality near the 3′ end is usually lower, which could be the cause of mismatches among sequences belonging to the same cluster.

3 EVALUATION

3.1 Test results with simulated data

To test the performance of SEED, we generated 1000 random center sequences. For each of these, we randomly generated sequences with mismatches and overhanging ends, so that the number of center sequences was the number of true clusters. The main objectives of these tests were to determine how well SEED clusters the sequences with respect to the number of clusters, and the number of falsely assigned members in them compared with the true clusters. In the following, the latter aspect is referred to as the false positive ratio (FPR), which is the number false positive members divided by the size of a cluster averaged for all clusters in a set. In addition, the same tests were used to empirically determine the time and memory performance of the algorithm. In each test, we changed only one parameter while keeping the remaining parameters constant. The results of these tests are presented in Supplementary Tables S3a–S3g. They include tests for the number of sequences, sequence length, number of true clusters, number of mismatches, number of overhanging ends and QV1/QV2 constraints, respectively. The QV mode of the program was only used for the corresponding tests in Supplementary Tables S3f–S3g.

The time to cluster with SEED 10–100 million sequences of 40 bp in length increases linearly from 24 to 233 min, respectively (Supplementary Table S3a). For the same dataset, the memory footprint increases only sublinearly from 2.6 to 8.0 GB. When clustering sequence sets of increasing lengths, then the time also increases linearly, while the memory usage shows no change (Supplementary Table S3b). With increasing numbers of true clusters, the time requirement also changes sublinearly and the memory usage stays almost constant (Supplementary Table S3c). The number of clusters with at least 5000–10 000 members assembled by SEED is consistently smaller than the number of true clusters in the test datasets (Supplementary Tables S3a–S3g). However, the FPR in the cluster sets is almost exclusively 0. This means that SEED tends to split true clusters into smaller ones, but without contaminating them with false positive members from other clusters. This behavior is extremely important for many practical applications, because false cluster assignments would result in information loss, while splitting the clusters into smaller ones will not remove any important sequences. For instance, in assembly projects removing redundant sequences will help to reduce the memory requirements, but when the clusters are contaminated with false positives then the clustering will remove many sequences that may be important for an optimal assembly. Due to the more incremental similarity transitions among clusters in real datasets, one would expect the higher FPRs than with simulated data. This can be seen in the subsequent tests on real datasets. However, the FPRs on real datasets are still impressively low (see Section 3.2).

More mismatches require extra memory for bucket allocation, but the compute time decreases due to shorter chains (Supplementary Table S3d). The number of clusters shows the same trend, because the similarity threshold decreases with the number of mismatches allowing more sequences to be assigned to clusters. For similar reasons, the memory requirements shown in Supplementary Table S3e grow with increasing numbers of overhanging residues. However, the time requirements are increasing in this case, because the relative differences among the sequences dominate the clustering time. Also, the number of large clusters decreases, because more shifts tend to reduce the cluster sizes.

When running SEED in the quality aware QV mode (Supplementary Tables S3f and S3g), the quality scores need to be imported into the clustering process, which increases its memory footprint by ∼15%. The time requirements decrease with increasing threshold values of QV constraints, because greater threshold values tend to assign more sequences to clusters in each pass. In case of QV1, the number of large clusters increases, because more sequences can be assigned to clusters for greater QV1 values.

3.2 Test results with real data

3.2.1 Datasets and experimental design The performance and utility spectrum of SEED for real data was tested on four different types of NGS data that were downloaded form NCBI's Sequence Read Archive (SRA). In all cases, the sequence data were based on Illumina's NGS technology. They included experiments from the following application areas: genome resequencing (sample SRX016064 from *Rhodobacter sphaeroides*), ChIP-Seq (samples SRR038848-SRR038851 from *Arabidopsis thaliana*; Kaufmann *et al.*, 2010), RNA-Seq (samples SRR064149-SRR064152 from *Arabidopsis thaliana*; Jiao and Meyerowitz, 2010) and small RNA-Seq (samples SRR032112-SRR032115 from *Arabidopsis thaliana*; Hsieh *et al.*, 2009). The ChIP-Seq dataset was used to compare SEED with other clustering methods. Both the genome resequencing and the RNA-Seq datasets were used to evaluate the utility of SEED for *de novo* genome and transcriptome assembly projects with respect to improvements of the memory footprints and the contig sizes of the final results. Another test included a small RNA dataset for evaluating SEED's efficacy in identifying clusters of mature miRNA sequences in the absence of a reference genome.

In most test experiments, the NGS datasets were clustered with SEED. Subsequently, the resulting center sequences were used as input datasets for the downstream analysis steps that are commonly used in different application fields, such as assembly and genome/transcriptome alignment steps. The final results were then compared with results obtained without SEED preprocessing.

3.2.2 Cluster quality tests To evaluate how well SEED clusters NGS data, we designed test experiments with real datasets where we benchmarked its performance against the 'true' clusters obtained from genome alignment results. For comparison purposes, we also included the clustering software UCLUST and the assembly tool SSAKE in these tests (Edgar, 2010; Warren *et al.*, 2007). The former was chosen as a software representative with utilities similar to SEED's. In contrast to this, the typical use case of assembly tools is different, but when they are run on short reads with very stringent overlap criteria, then they can fulfill in parts the utility requirements of an NGS clustering tool. Among the many assembly tools available, SSAKE was chosen here because its output format provides the read positions in the contigs which simplified the downstream post-processing of the results. As test data, we used the four ChIP-Seq sets from *Arabidopsis thaliana*. These samples were selected because ChIP-Seq data contain highly variable enrichments of read pileups (peaks) along the chromosomes which is a relatively realistic and also challenging situation when testing the performance of a NGS clustering tool. The true clusters for these datasets were obtained by aligning the reads with Bowtie against the *Arabidopsis* reference genome while allowing up to three mismatches in the alignments. Subsequently, all aligned sequences that completely overlapped with other sequences in the pileup were assigned to clusters with two or more members. Sequences with no or only partial overlaps to other reads were assigned to singlet clusters. The resulting dataset is referred to as the 'true' cluster set, because it resembles an almost ideal benchmark result of high quality. To obtain meaningful results for the other tools, we used for them comparable parameters. SEED clustering was run with up to three mismatches, but no overlapping ends to match the constraints of the alignment-based reference cluster set. For UCLUST, we used comparable parameters by setting the identity parameter to $\frac{l-k}{l}$. Similarly, SSAKE was run with settings that were optimized to obtain only clusters of almost identical sequences. Most importantly, its parameter for the number of matched positions was set to $l-k$.

Table 1 gives an overview of the clustering results obtained by the different methods. Compared with the other methods, SEED has at least a 3- to 10-fold better time performance than the other two methods, but its memory requirements are not as low as UCLUST's. With respect to the cluster qualities, SEED performs consistently better than the other methods by showing the highest Jaccard index values relative to the true clusters. The Jaccard index is a commonly used similarity measure for comparing clustering results, where values close to 0 indicate low similarities and values closer to 1 higher similarities among the evaluated cluster sets. In addition, we used the clustering results presented in Table 1 to compare the prediction performance of SEED with the other methods. For this, we plotted in Supplementary Figure S2 the FPRs against the true positive rates (TPR). The FPR is defined as FP/(FP+TN) and the TPR as TP/(TP+FN). The individual variables were determined by finding in the results those clusters that show a minimum similarity x to the true clusters. TP is the number of sequences in each cluster contributing to the similarities; FP is the number of sequences in the clusters that do not contribute to the similarities; TN is the number of sequences not in the clusters that should not contribute to the similarities; FN is the number of sequences not in the clusters that should contribute to the similarities. In the resulting graph (Supplementary Fig. S2), SEED shows the best performance by having consistently the highest TPR values and in most cases lower FPR values as well. The better sensitivity and specificity of SEED is most likely linked to its virtual center sequence for guiding the clustering process. This approach provides relative accurate approximations of the true cluster centers. In this regard, UCLUST is less conservative by centering its clusters around a single seed sequence. In addition, SEED is optimized to cluster very similar NGS reads with variable arrangements of mismatch positions. In contrast to this, UCLUST is optimized for detecting a wider range of sequence similarities based on common word matches in its initial search step. This approach is more likely to miss certain high similarity matches that fall below the word size limit of the algorithm. However, the latter feature appears to be less critical, because even when UCLUST is used in its *optimal* mode, where it does not dependent on common word matches (see rows with UCLUSTo in Table 1), the performance of SEED is still better.

One concern with the seed algorithm could be that its clustering results may vary depending on which read is chosen first in the random selection process to initialize the formation of the virtual center sequence of a cluster. To address this, we also performed tests on the four ChIP-Seq datasets from *Arabidopsis thaliana* where we varied the factor f to compute the virtual center sequence as well as the order of reads (data not shown). The quality of the resulting cluster sets was evaluated again with the Jaccard index. With increasing values of f from 1 to 4, the Jaccard index showed only minor differences (<0.01) for the four datasets. We set $f=2$ as the default value in all of our experiments, since it gave one of the best results in our tests and it is also a reasonable choice based on the discussion in Section 2.5. Similarly, changing the orders of reads resulted in insignificant changes of the Jaccard index (<0.01). These tests indicate a relatively stable performance of SEED with respect to these parameter changes.

Table 1. Clustering with different methods

Method	No. of clusters	No. of clusters identical with true ones	Jaccard index	Time	Memory (GB)
SRR038848 (4 962 666 reads aligned)					
True	1 106 780				
SEED	973 627	632 209	0.96	00:06:12	2.3
UCLUST	977 904	618 101	0.92	01:28:54	0.4
UCLUSTo	976 871	622 028	0.92	01:44:25	0.4
SSAKE	1 431 122	650 596	0.86	00:20:09	3.0
SRR038849 (2 435 754 reads aligned)					
True	973 673				
SEED	880 920	512 270	0.97	00:04:02	2.2
UCLUST	873 784	500 982	0.94	00:36:23	0.4
UCLUSTo	873 135	502 654	0.94	00:42:43	0.4
SSAKE	1 070 654	515 574	0.91	00:13:56	2.3
SRR038850 (5 386 160 reads aligned)					
True	3 365 685				
SEED	3 151 149	664 359	0.95	00:09:47	2.8
UCLUST	3 086 836	669 243	0.88	04:13:09	1.4
UCLUSTo	3 084 657	674 211	0.88	07:12:52	1.4
SSAKE	3 814 607	599 858	0.86	00:51:38	6.9
SRR038851 (3 148 061 reads aligned)					
True	2 182 354				
SEED	2 038 577	287 903	0.94	00:06:28	2.5
UCLUST	2 096 534	297 756	0.84	01:37:47	0.9
UCLUSTo	2 094 080	300 539	0.85	01:45:00	0.9
SSAKE	2 540 359	214 013	0.77	00:34:10	4.6

The clustering results for four ChIP-Seq samples are shown for the true clusters (alignment based method), SEED, SSAKE, and UCLUST with and without its *optimal* mode. The 'true' cluster data were used as references to compute the Jaccard index in the fourth column.

Table 2. Assembly tests

Preprocessing	No. of sequences to assemble (read length)	No. of contigs	N50	Mean length of contigs	Memory for assembly (GB)	Time for assembly	Memory for clustering	Time for clustering
Genome assembly								
None	51 448 694 (36 bp)	2230	5143	2039	9.7	07:53:54	–	–
SEED	10 644 813 (36 bp)	1918	6504	2382	5.7	01:11:59	4.1 GB	03:41:29
Random sampling	10 644 813 (36 bp)	2924	3855	1531	2.5	01:12:25	–	–
Transcriptome assembly								
None	72 295 211 (37 bp)	21 014	452	338	28	15:08:36	–	–
SEED	29 841 222 (37 bp)	12 988	507	391	22	05:59:33	8.7 GB	04:09:51
Random sampling	29 841 222 (37 bp)	12 868	396	315	12	05:57:09	–	–

The assembly results with Velvet/Oases are shown for the genome resequencing data set from *Rhodobacter sphaeroides* (upper panel) and the transcriptome RNA-Seq data set from *Arabidopsis thaliana* (lower panel). The table compares row wise the results for the following preprocessing steps of the raw sequences: no preprocessing, preprocessing with SEED, random sampling of the same number of reads obtained with SEED. The parameters used for SEED were ≤3 mismatches, ≤3 overhanging ends and QV mode disabled. The corresponding cluster size distributions for the genome assembly are given in Figure 1.

3.2.3 Assemblies assisted with SEED Assemblies rank among the most challenging computational problems in the NGS field (Birney, 2011). Partially, this is because they tend to be an iterative and time consuming improvement process with highly variable outcomes for different datasets (Miller *et al.*, 2010). Moreover, their memory requirements and execution times are often so extensive that larger datasets can only be assembled on high performance compute systems with considerable CPU and memory resources. To improve this, we tested SEED for upstream processing prior to assembly and then analyzed the time and memory requirements of the assembly step, as well as the qualities of the resulting contigs. The assembly components of these tests were performed with Velvet, which is one of the most widely used assembly tools for NGS data (Schmidt *et al.*, 2009; Zerbino and Birney, 2008). To run the assemblies with optimized parameters, the Velvet Optimiser tool was used. The genome assemblies were performed with Velvet only, and the transcriptome assemblies included both Velvet and its transcriptome-specific Oases component. All software tools were run on a single CPU core (64-bit 2.4 GHz Xeon Quad Core Harpertown) to allow fair comparisons of their time and memory usages.

(A) Genome assembly: Table 2 and Figure 1 summarize the assembly results for the genome resequencing dataset from *Rhodobacter sphaeroides* with Velvet. These tests were performed with and without SEED preprocessing. A random set was included for comparison, where we assembled the same number of sequences as obtained in the preprocessing step with SEED, but by randomly selecting the reads from the raw dataset. Compared with the non-preprocessed dataset, the assembly time and memory requirements in the SEED dataset are greatly reduced by 84.8 and 41.2%, respectively (Table 2, upper panel). With respect to the quality of the assembly results, several commonly used quality measures improved in the SEED dataset compared with the non-preprocessed dataset: the number of contigs decreased by 14.0%, mean length of the contigs increased by 16.8% and N50 value increased by 26.5%. The latter is the contig length where 50% of the entire assembly is contained in contigs of at least this value. In contrast to this, the corresponding measures in the dataset generated by random sampling show the opposite trend. A more detailed overview of the cluster size distributions in the three result sets is given in Figure 1.

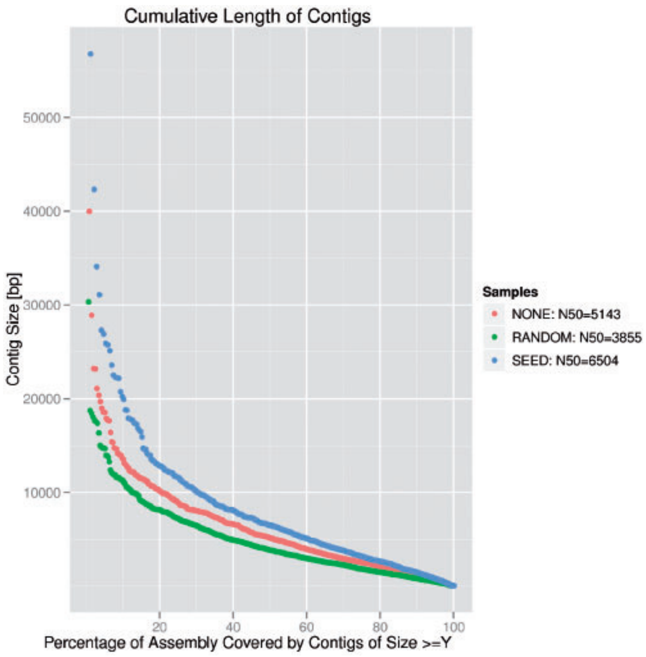

Fig. 1. Cumulative contig sizes of genome assemblies. The plot compares the cumulative contig size distribution of the Velvet assembly results presented in the upper panel of Table 2 (for details see table legend). In this plot, the N50 value is the contig size (*Y*-axis) at 50% of the assembly coverage (*X*-axis).

In this plot, the SEED dataset shows in comparison to the other tests the highest cumulative contig sizes.

(B) Transcriptome assembly: To also test whether SEED preprocessing could provide improvements for assemblies of transcriptomes, we performed similar tests with the chosen RNA-Seq dataset from *Arabidopsis thaliana*. When using SEED, both the time and memory requirements decreased by 60.4 and 21.4%, respectively. In addition, the mean contig length and the N50 value could be increased by 15.7 and 12.2%, respectively.

Table 3. miRNA profiling with SEED

Samples	No. of sequences	No. of clusters (size ≥10)	miRNAs identified (all samples 96%)	PCC
SRR032112 (Root−Pi)	5 142 120	37 315	76.1	0.91
SRR032113 (Root+Pi)	4 919 514	38 193	83.3	0.89
SRR032114 (Shoot−Pi)	4 862 947	46 776	89.4	0.82
SRR032115 (Shoot+Pi)	5 003 481	43 176	86.6	0.87

The table gives for the four small RNA samples from Hsieh *et al.* (2009) the number of sequences in each data set, the number of clusters obtained by SEED with ≥10 members, the relative number of miRNAs covered by these clusters, and the PCCs for the published read counts and the ones obtained by SEED.

The above results on genome and transcriptome data clearly indicate that SEED preprocessing can improve the performance of downstream sequence assemblies using Velvet with respect to compute time, memory usage and quality parameters of the final contigs. Time and memory improvements are the main advantages here, whereas quality enhancements of the final results are likely to vary depending on the specific challenges presented by different sequence types. Investigating which datasets are particularly affected by this and how SEED exactly improves the quality of assemblies (e.g. error correction), goes beyond the scope of this study. When assembling transcriptome data, SEED clustering will help to reduce the extreme redundancies of very abundant mRNA species in these datasets, while maintaining the important information relevant for many RNA-Seq applications. On the other hand, when assembling genomes with highly repetitive sequences, often it will be necessary to perform SEED preprocessing with very stringent mismatch settings (e.g. $k \leq 1$), because higher numbers of mismatches in SEED clustering may eliminate information critical to achieve an optimal assembly of highly similar genomic regions.

3.2.4 Discovery and profiling of miRNAs with SEED To explore the potential utility of SEED for identifying and profiling mature miRNA clusters in unsequenced organisms, we performed the following tests. First, we clustered with SEED, the raw sequences from four different NGS samples from a recently published small RNA profiling study in *Arabidopsis thaliana* (Hsieh *et al.*, 2009). In this study, the authors determined by NGS the expression profiles of 180 miRNAs from root and shoot tissues both grown in the presence and absence of phosphate (Pi). Subsequently, we identified for all miRNAs profiled in the published study the corresponding center sequences in the SEED clustering results. In this association step, the center and mature miRNA sequences had to fully overlap and show not more than one mismatch. Finally, we compared the sequence counts (expression profiles) for each of the miRNAs in the published dataset with the size of the corresponding SEED clusters (Table 3). Considering only clusters with at least 10 sequences, 76.1–89.4% of the miRNAs in the published dataset could be associated with SEED clusters. The likelihood of finding this many overlaps just by chance is very low (random sampling test $P < 10^{-5}$). On average, these clusters contain 20–48% more sequences than clusters obtained by a simple counting approach of absolutely identical reads (data not shown). The Pearson's correlation coefficients (PCC) for the sequence counts for each miRNA in the published dataset and the corresponding SEED clusters are relatively high for all four samples (PCC: 0.82-0.91). This high correlation, and the high coverage of known miRNAs detected by these tests, illustrate SEED's utility for identifying in unsequenced genomes candidate clusters of mature miRNA sequences and obtaining for them relatively reliable expression data. A challenge in real datasets without a reference genome will be the identification of the correct miRNA clusters among the much larger pool of unrelated clusters (third column in Table 3). This can be largely overcome by sequence similarity searching. Here, one can identify clusters with similarities to known miRNAs, which are often evolutionary conserved. In addition, one can easily eliminate by similarity searching against reference databases the typical contaminants in small RNA datasets, such as ribosomal RNAs or transposons.

4 CONCLUSIONS AND FUTURE WORK

In this study, we introduced SEED as an efficient method for clustering very large NGS datasets while allowing up to three mismatches and three overhanging residues to their virtual center. The method gains its performance from a block spaced seed method that greatly accelerates the downstream clustering process. With increasing numbers of sequences, the method shows a linear time and memory performance. It is able to cluster on a single CPU core 100 million sequences in less than four hours, while using not >8 GB of memory. These are very reasonable resource requirements for modern computers. The current implementation of SEED is optimized to handle sequences of 21–100 bp in length. This matches at the moment the length range of most of the widely used NGS technologies, such as Illumina's reversible terminator method.

SEED's application spectrum is very broad. While most of its use cases fall into the data preprocessing area, it also has utilities as stand-alone discovery application for organisms where reference genome or transcriptome sequences are not available. For instance, it can be used in those cases to identify clusters of short DNA or RNA molecules that are abundant in genome or transcriptome samples, such as miRNAs or transposons. As preprocessing and data reduction tool, SEED is very efficient in improving the time and memory requirements of downstream NGS data processing routines, such as genome and transcriptome assemblies, often by a factor of 2- to 5-fold, based on the NGS test datasets used in this study. Moreover, reducing the redundancies in NGS data with SEED does not negatively impact the quality of the contigs in downstream assembly steps. In case of the Velvet/Oasis assembler, the N50 values of transcriptome and genome assemblies could be improved with SEED preprocessing by 12–27%.

Similarity-based clustering can be an efficient approach to remove undesirable redundancies in NGS data. However, the removal of redundant reads will unavoidably be accompanied by an information loss in the data. While this can be often a desirable and/or tolerable consequence for many downstream analysis routines, the information loss can also negatively influence the outcome of certain applications, such as discovery of mutations (e.g. SNPs) or assemblies of repetitive genomic regions. As a general rule, if high resolution of very similar reads is important for a NGS project, then similarity clustering should be restricted to identical reads or not used at all.

In future, we will expand the performance and utility spectrum of SEED on several levels. First, we will optimize the method by further improving its memory footprint and time performance. A parallelized version of the algorithm can be easily implemented by issuing many simultaneous queries to the hash tables while using locks for managing interprocess dependencies. Second, we will improve its minimum and maximum sequence length limits to support clustering of sequences that are shorter or longer than 21 or 100 bp, respectively. Finally, additional input and output formats will be implemented in SEED to provide support for a wide spectrum of upstream and downstream software tools and programming environments.

ACKNOWLEDGEMENT

We acknowledge the support of the core facilities at the Institute for Integrative Genome Biology (IIGB) at UC Riverside.

Funding: USDA National Institute for Food and Agriculture (NIFA-2010-65106-20675 to I.K.); National Science Foundation (ABI-0957099 to T.G., IOB-0420152 to T.G., IGERT-0504249 to T.G., IIS-0711129 to T.J.).

Conflict of Interest: none declared.

REFERENCES

Birney,E. (2011) Assemblies: the good, the bad, the ugly. *Nat. Methods*, **8**, 59–60.

Cock,P.J. *et al.* (2010) The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Res.*, **38**, 1767–1771.

Durbin,R.M. *et al.*; 1000 Genomes Project Consortium. (2010) A map of human genome variation from population-scale sequencing. *Nature*, **467**, 1061–1073.

Edgar,R.C. (2010) Search and clustering orders of magnitude faster than BLAST. *Bioinformatics*, **26**, 2460–2461.

Fritz,M.H.Y. *et al.* (2011) Efficient storage of high throughput sequencing data using reference-based compression. *Genome Res.*, **21**, 734–740.

Hazelhurst,S. *et al.* (2008) An overview of the wcd EST clustering tool. *Bioinformatics*, **24**, 1542.

Holt,R.A. and Jones,S.J. (2008) The new paradigm of flow cell sequencing. *Genome Res.*, **18**, 839–846.

Hsieh,L.C. *et al.* (2009) Uncovering small RNA-mediated responses to phosphate deficiency in Arabidopsis by deep sequencing. *Plant Physiol*., **151**, 2120–2132.

Huang,X. and Madan,A. (1999) CAP3: A DNA sequence assembly program. *Genome Res.*, **9**, 868.

Jiang,H. and Wong,W. (2008) Seqmap: mapping massive amount of oligonucleotides to the genome. *Bioinformatics*, **24**, 2395.

Jiao,Y. and Meyerowitz,E.M. (2010) Cell-type specific analysis of translating RNAs in developing flowers reveals new levels of control. *Mol. Syst. Biol.*, **6**, 419–419.

Johnson,C. *et al.* (2009) Clusters and superclusters of phased small RNAs in the developing inflorescence of rice. *Genome Res.*, **19**, 1429–1440.

Jothi,R. *et al.* (2008) Genome-wide identification of in vivo protein-DNA binding sites from ChIP-Seq data. *Nucleic Acids Res.*, **36**, 5221–5231.

Kaufmann,K. *et al.* (2010) Orchestration of floral initiation by APETALA1. *Science*, **328**, 85–89.

Langmead,B. *et al.* (2009) Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol.*, **10**, 25–34.

Leinonen,R. *et al.* (2010) The European Nucleotide Archive. *Nucleic Acids Res.*, **39**, 28–31.

Li,W. and Godzik,A. (2006). Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, **22**, 1658–1659.

Li,H. *et al.* (2008) Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Res.*, **18**, 1851.

Li,H. and Durbin,R. (2009a) Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, **25**, 1754–1760.

Li,H. *et al.*; 1000 Genome Project Data Processing Subgroup (2009b) The sequence alignment/map format and SAMtools. *Bioinformatics*, **25**, 2078–2079.

Lin,H. *et al.* (2008) ZOOM! Zillions of oligos mapped. *Bioinformatics*, **24**, 2431–2437.

Ma,B. *et al.* (2002) PatternHunter: faster and more sensitive homology search. *Bioinformatics*, **18**, 440–445.

Medini,D. *et al.* (2008) Microbiology in the post-genomic era. *Nat. Rev. Microbiol.*, **6**, 419–430.

Miller,J.R. *et al.* (2010) Assembly algorithms for next-generation sequencing data. *Genomics*, **95**, 315–327.

Montgomery,T.A. *et al.* (2008) AGO1-miR173 complex initiates phased siRNA formation in plants. *Proc. Natl Acad. Sci. USA*, **105**, 20055–20062.

Picardi,E. *et al.* (2009) EasyCluster: a fast and efficient gene-oriented clustering tool for large-scale transcriptome data. *BMC Bioinformatics*, **10** (Suppl. 6), S10.

Qu,W. *et al.* (2009). Efficient frequency-based de novo short-read clustering for error trimming in next-generation sequencing. *Genome Res*, **19**, 1309–1315.

Rao,D. *et al*. (2010) PEACE: Parallel Environment for Assembly and Clustering of Gene Expression. *Nucleic acids research*, **38** (Suppl. 2), W737.

Schmidt,B. *et al*. (2009) A fast hybrid short read fragment assembly algorithm. *Bioinformatics*, **25**, 2279–2280.

Warren,R.L. *et al*. (2007) Assembling millions of short DNA sequences using SSAKE. *Bioinformatics*, **23**, 500–501.

Zerbino,D.R. and Birney,E. (2008) Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res*, **18**, 821–829.