Starcode: sequence clustering based on all-pairs search

Eduard Valera Zorita 1,2, Pol Cuscó 1,2 and Guillaume Filion 1,2*

1 Genome Architecture, Gene Regulation, Stem Cells and Cancer Programme, Centre for Genomic Regulation (CRG), Dr. Aiguader 88, 08003 Barcelona, Spain.
2 Universitat Pompeu Fabra (UPF), Barcelona, Spain.

ABSTRACT

Motivation: The increasing throughput of sequencing technologies offers new applications and challenges for computational biology. In many of those applications, sequencing errors need to be corrected. This is particularly important when sequencing reads from an unknown reference such as random DNA barcodes. In this case, error correction can be done by performing a pairwise comparison of all the barcodes, which is computationally complex problem.

Results: Here we address this challenge and describe an exact algorithm to determine which pairs of sequences lie within a given Levenshtein distance. For error correction or redundancy reduction purposes, matched pairs are then merged into clusters of similar sequences. The efficiency of starcode is attributable to the poucet strategy, which uses the redundancy of alphabetically sorted barcodes, starcode outperforms sequence clustering algorithms in both speed and precision.

Availability and implementation: The C source code is available at http://github.com/gui1laume/starcode.

Contact: guillaume.filion@gmail.com

1 INTRODUCTION

All sequencing technologies have a certain degree of imprecision. For instance, the Illumina platform (Margulies et al., 2005) has a 1-2% error rate consisting of substitutions (Dohm et al., 2008; Nakamura et al., 2011) and the PacBio platform has a 15% error rate consisting of insertions and deletions (Eid et al., 2009). The enormous throughput of such technologies has recently created additional needs for developing efficient error correction algorithms.

Sequencing errors can be discovered by comparing the reads to a reference genome. However, such a reference is not always available. When the sequences are random or taken from an unknown source, clustering is the main strategy to correct the errors. For instance, this situation arises when using random barcodes to track cells or transcripts (Schepers et al., 2008; Akhtar et al., 2013). Sequencing errors will create erroneous (nonexistent) barcodes that have to be removed.

Sequence clustering can be viewed as a community detection problem on graphs, where nodes represent sequences and edges represent matches between related sequences. The process consists of a matching phase (the most computationally intensive), where the graph is constructed, and a clustering phase where communities are identified.

Here we describe a sequence clustering algorithm called “starcode” in reference to clusters of random barcodes, which typically have a star shape. Starcode is based on all-pairs search, i.e. all the pairs of sequences below a given Levenshtein distance are identified during the graph construction phase. Matching is carried out by lossless filtration, followed by an exhaustive search on the branches of a prefix trie. The novelty of the algorithm is the poucet strategy, which uses the redundancy of alphabetically sorted sequences to avoid unnecessary recomputations and gain speed.

In this article we present and benchmark starcode. We show that on real biological datasets, starcode is orders of magnitude faster than existing sequence clustering software. Even though starcode was designed for error correction, we also show that it can be used for other problems. As an illustration, we use it to identify enriched motifs in a bacterial genome and in protein-RNA interaction experiments.

2 METHODS

2.1 Inexact string matching using tries

The matching method of starcode is based on a variation of the Needleman-Wunsch (NW) algorithm (Needleman and Wunsch, 1970). In the original algorithm (Figure 1a), the Levenshtein distance between two sequences is found by applying a recurrence relation throughout a matrix of mn terms (the edit matrix), where m and n are the respective sequence lengths. The complexity of this dynamic programming approach is $O(mn)$.

In many instances, the information of interest is to find out whether the sequences are $\tau$-matches (i.e. their distance is less then or equal to a fixed threshold $\tau$). In that case, the complexity can be reduced to $O(\tau \min(m, n))$. Instead of computing all the terms of the edit matrix, it is initialized as shown on Figure 1b and only the terms around the diagonal are computed. If a diagonal term has a value greater than $\tau$, the process is halted because the sequences are not $\tau$-matches.

This method can be used to match sequences against a prefix tree, also known as a trie (Ukkonen, 1995). The terms of the edit matrix are updated row-wise while a depth-first search traverses the trie (Figure 2). Every time a node is visited, a row is computed, and every time the search backtracks, a row is erased. If the threshold value $\tau$ is exceeded for a diagonal term, the Levenshtein distance for all the downstream sequences is also necessarily greater than $\tau$. Therefore, no more hits are to be discovered in this path and the depth-first search backtracks to the parent node. When the process halts, every tail node (corresponding to a sequence of the database) on the path of this search is a $\tau$-match of the query. This method is efficient because it
eliminates large areas of the search space, and because the NW comparison of the query with each prefix of the database is computed only once.

2.2 The poucet search algorithm

The search strategy can be further improved. If two consecutive queries share a prefix of length $k$, the succession of computations up to the $k$-th row of the edit matrix will be exactly the same for both queries. Therefore, computation intermediates can be stored in the nodes of the trie, so that the next trie search can start at depth $k$. However, storing the rows of the edit matrix in the nodes meets some difficulty. Indeed, on the $k$-th row, the terms on the right side of the diagonal depend on characters that are not shared between the two queries. This issue is solved by storing in each node a combination of row and column terms that form an angle shape, looking like a horizontally flipped L (Figure 3). Using this structure, the computation intermediates stored in a node at depth $k$ depend only on the first $k$ characters of the query.

To take full advantage of this property, the input sequences are sorted alphabetically, which maximize prefix sharing between consecutive queries. In the fairy tale “Le Petit Poucet”, the hero seeds white pebbles for his older brothers to find their way home, which is reminiscent of the way a smaller query (in alphabetical order) paves the way for the next. We therefore called this search algorithm “poucet”.

2.3 Lossless filtration

When a query has no match, it is advantageous to omit the trie search. To this end, starcode uses a partition approach similar to that described by T. The query is partitioned in $\tau+1$ segments. Assuming that all the segments have length at least $\tau$, then every $\tau$-match present in the database will contain at least a verbatim copy of one of the query segments. Indeed, there are at most $\tau$ editions between the query and the match to be distributed.

This end, starcode uses a partition approach similar to that described by T. The query is partitioned in $\tau+1$ segments. Assuming that all the segments have length at least $\tau$, then every $\tau$-match present in the database will contain at least a verbatim copy of one of the query segments. Indeed, there are at most $\tau$ editions between the query and the match to be distributed.
in \( \tau + 1 \) regions, so at least one segment is unmodified. Due to potential insertions and deletions in the preceding segments, the shared segment may be shifted up to \( \tau \) nucleotides on the left (all insertions) or on the right (all deletions) from its original position in the query.

These observations are the basis of a filtration method with 100% specificity. More precisely, the segments are defined as follows: the first \( \tau \) nucleotides of the sequence are removed, and the rest of the sequence is divided into \( 4 \) series of contiguous segments. Each series is partitioned in the same way as the original sequence, \( \tau \)-mers of the sequence are looked up in the indexes according to the scheme of Figure 4, and then proceeds in two phases. In the build phase, a distinct trie is built from the sequences of each block according to the algorithm described above. In the second, all the sequence blocks are queried against all the other tries.

If the queries are partitioned in \( N \) blocks, the first phase consists of \( N \) seek and construct jobs, while the second consists of \( N(N - 1)/2 \) query jobs. In each phase, the jobs show no dependency on each other, so the matching algorithm can be efficiently parallelized provided \( N \) is larger than the number of independent threads.

![Algorithm 1 Starcode algorithm](image)

**Algorithm 1 Starcode algorithm**

1: Define: \( \tau \)
2: Variables: \( \text{seed, start} = 0, \text{height, seq, trie, lastseq, k} \)
3: Containers: \( \text{hits, pebbles} \)
4: READ sequence file
5: \( \text{height} \rightarrow \text{Determine maximum sequence length} \)
6: PAD sequences up to \( \text{height} \)
7: SORT sequences alphabetically
8: \( k \leftarrow \text{Compute filter segment lengths} \)
9: \( \text{trie} \leftarrow \text{Create an empty trie of height height} \)
10: INSERT root node of \( \text{trie} \) in \( \text{pebbles} \) at depth 0
11: for all \( \text{sequences do} \)
12: \( \text{seq} \leftarrow \text{GET next sequence} \)
13: if at least one \( k \)-mer of \( \text{seq} \) is in the filter index then
14: \( \text{seed} \leftarrow \text{LENGTH of shared prefix between current and next sequence} \)
15: \( \text{start} \leftarrow \text{LENGTH of shared prefix between seq and lastseq} \)
16: \( \text{CLEAR hits} \)
17: \( \text{CLEAR pebbles at depth} \geq \text{start} \)
18: for all \( \text{pebbles at depth start do} \)
19: \( \text{node} \leftarrow \text{GET next node from pebbles} \)
20: \( \text{call POUCET(seq, node, seed, hits, pebbles)} \)
21: end for
22: \( \text{PROCESS hits and LINK matches to seq} \)
23: \( \text{lastseq} \leftarrow \text{seq} \)
24: end if
25: INSERT \( \text{seq} \) path in \( \text{trie} \)
26: INSERT \( \text{seq} \) \( k \)-mers into the filter index
27: end for

![Algorithm 2 Poucet search algorithm](image)

**Algorithm 2 Poucet search algorithm**

1: procedure POUCET(query, node, seed, hits, pebbles):
2: \( \text{COMPUTE node-specific column following NW} \quad \triangleright \text{Fig.1} \)
3: for all child nodes in \( \text{node do} \)
4: \( \text{COMPUTE child-specific row following NW} \quad \triangleright \text{Fig.1} \)
5: \( \text{COMPUTE center value using row and column} \quad \triangleright \text{Fig.1} \)
6: if center value \( \geq \tau \) then \( \triangleright \text{Mismatches exceeded.} \)
7: \( \text{continue with next child} \)
8: end if
9: if node depth = \( \text{height} \) then \( \triangleright \text{Hit found.} \)
10: \( \text{SAVE node sequence in hits} \)
11: \( \text{continue with next child} \)
12: end if
13: if node depth \( \leq \text{seed} \) then
14: \( \text{SAVE node in pebbles at current depth} \)
15: end if
16: \( \text{call poucet(query, child, seed, hits, pebbles)} \)
17: end for
18: end procedure
2.6 Clustering

The default clustering algorithm of starcode is designed to correct sequencing error. This method uses message passing (MacKay, 2002) to identify and count “canonical” sequences (also referred to as centroids in the clustering terminology). By default, each sequence transfers its read count to its closest \( \tau \)-match provided the latter has at least 5 times more counts. If the condition is not met, the transfer does not take place. If the sequence has several equally close \( \tau \)-matches, the counts are split equally among them. The process is repeated recursively, starting from sequences with lowest read count. The sequences with a positive read count at the end of the process are considered canonical. Clusters consist of all the sequences transferring their read counts to the same canonical sequence (sequence transferring their read counts to different canonicals are discarded). Note that the radius of the clusters can be higher than the maximum distance used for matching.

Since no sequencing technology has an error rate higher than 20%, it is expected that sequences appearing from sequencing errors will always have 5 times or lower read count than the canonical sequence. Otherwise, sequences are more likely unrelated, or both are derived from the same canonical sequence. This behavior can be modified with the command-line option \( \text{cluster-ratio} \) to allow for a more flexible or more strict clustering, e.g. to cluster unique input sequences together, \( \text{cluster-ratio} \) must be set to 1. For other sequence clustering problems, starcode implements a multi-purpose algorithm called “sphere clustering”. In sphere clustering, sequences are sorted by frequency of occurrence. Starting from the most frequent, each sequence becomes canonical and claims all its \( \tau \)-matches, which forms a cluster of radius \( \tau \) (hence the name). Claimed sequences are immediately removed, so that they can belong to only one cluster.

2.7 Benchmark conditions

All the tests were performed on a 16-core dual-processor Intel Xeon E5-2687W v2 system with 256 GB of DDR3-RAM at 1866 MHz. Command-line parameters were set equivalently in all softwares to run in single-core mode allowing up to 3 mismatches for input sequences of length 50. Tables 1 and 2 summarize the execution options used in simulation and real data sets, respectively.

Table 1. Software execution options used in simulation benchmark.

<table>
<thead>
<tr>
<th>software</th>
<th>command-line options</th>
</tr>
</thead>
<tbody>
<tr>
<td>starcode-1.0</td>
<td>starcode -d3</td>
</tr>
<tr>
<td>slidesort-2</td>
<td>slidesort_v2 -d 3 -t E -c DNA</td>
</tr>
<tr>
<td>cd-hit-est-4.6.1</td>
<td>cd-hit-est -n 9 -c 0.9 -M 0 -r 0</td>
</tr>
<tr>
<td>seed-1.4.1</td>
<td>SEED --mismatch 3</td>
</tr>
</tbody>
</table>

Table 2. Software execution options used in real data benchmark.

<table>
<thead>
<tr>
<th>software</th>
<th>command-line options</th>
</tr>
</thead>
<tbody>
<tr>
<td>starcode-1.0</td>
<td>starcode -d3</td>
</tr>
<tr>
<td>slidesort-2</td>
<td>slidesort_v2 -d 3 -u -t E -c DNA</td>
</tr>
<tr>
<td>cd-hit-est-4.6.1</td>
<td>cd-hit-est -n 8 -c 0.94 -M 0</td>
</tr>
<tr>
<td>seed-1.4.1</td>
<td>SEED --mismatch 3 --shift 3</td>
</tr>
<tr>
<td>rainbow-2.0.3</td>
<td>rainbow cluster -m 3</td>
</tr>
</tbody>
</table>

3 RESULTS

3.1 Presentation and basic performance

Starcode is a general purpose DNA sequence clustering tool with a strong focus on error correction. Errors are assumed to be mismatches, insertions or deletions (the implementation presented here matches sequences with up to 8 errors). The input sequences can be single or paired-end reads, with an upper limit of 1024 nucleotides (512 for paired-end). Sequences may be of variable length, they may be trimmed and filtered for quality or not. File formats compatible with starcode are raw sequence, raw sequence with count, FASTA or FASTQ (in which case starcode ignores the quality). Starcode either returns detailed information of the clustering results, i.e. canonical sequences, cluster sizes and the complete list of their constituent sequences. Alternatively, only the canonical sequences are printed, which is useful to filter out redundant sequences from input files. By default, clustering is performed under the assumption that divergence occurs from experimental errors (sequencing errors, PCR mutations etc.) and a more general algorithm is also available for other clustering problems (an example of which is given in section 3.3).

We show the basic performance and scalability of starcode on a dataset of pseudorandom sequences (Figure 5). The standard configuration is a set of 1,000,000 sequences of length 40, running on 1 thread and with a maximum Levenshtein distance of 3. In each test, only one parameter is modified while the others are kept constant. Since the clustering step does not require additional memory allocation and is significantly faster than all-pairs search, the performance results presented in Sections 3.1 and 3.2 apply for both message-passing and sphere clustering algorithms.

Figure 3a shows the running time of starcode as a function of the number of input sequences \( n \). In double logarithmic scale the trend is a straight line with slope 1.5, suggesting that the running time complexity of starcode is lower than quadratic (the naive implementation of all-pairs search). Note that the sequences of this dataset have no match, see section 3.2 for an evaluation of the performance on more realistic datasets. Figure 5b shows that the running time grows exponentially as a function of the maximum Levenshtein distance used for clustering. The reason is that the trie fans out exponentially and the search bails out at a greater depth as the maximum distance increases. As a function of the sequence length, the running time first increases but then plateaus and stays low (Figure 5c). Beyond a threshold length, the filtering algorithm starts to be efficient, and most of the queries are resolved without searching the trie. Finally, we show the scalability of starcode with increasing number of threads in Figure 5d. The search algorithm is fully parallel and the relative performance increases linearly up to 12 threads. The bending observed thereafter has two sources; the first is that the input reading and clustering steps are brief but not parallel, the second is due to hardware limitations, i.e. there is insufficient memory bandwidth to satisfy the increased demand of parallel memory accesses.

3.2 Benchmark

We benchmarked starcode against the sequence clustering algorithms slidesort (?), seed (?), rainbow (?) and cd-hit (Fu et al., 2012). Even though slidesort is an all-pairs search algorithm, it was included in the benchmark because sequence comparison is the most computationally intensive step of the sequence clustering problem. Rainbow runs exclusively on paired-end reads, whereas the other tools run on single reads, for this reason all the tools could not be run on the same dataset.
While starcode achieves perfect clustering on all 4 datasets, the clustering achieved by seed and cd-hit is incomplete. Both tools identify approximately 40 false clusters per true cluster on all the datasets (Figure 6a). We also observed that slidesort found 5-10% less 3-matches than starcode on all the datasets (Figure 6b). We were surprised by this result because slidesort is claimed to be an exact algorithm. However, this was clearly not the case when we ran additional tests on smaller datasets where naive pairwise comparisons are feasible (more information is available on the starcode repository, see http://github.com/gui11aume/starcode/tree/master/misc).

The running time of the different tools as a function of the size of the clusters is shown on Figure 6c. The running time of slidesort and starcode show linear and sub-linear trends, respectively. Seed and cd-hit run approximately in constant time regardless of the cluster size. In spite of this result, the performance of starcode remains competitive, even for clusters of 1 million sequences. The memory usage is shown in Figure 6c. The smallest memory footprint is achieved by slidesort and cd-hit, with a maximum difference of an order of magnitude with respect to the other tools. Note that the comparison with slidesort is not completely fair since it does not hold in memory the full graph necessary for clustering. The memory usage of starcode is the highest for clusters of size 1000, but it decreases and becomes lower than the memory usage of seed as the size of the cluster increases. In conclusion, starcode was the only tool to achieve perfect precision on these datasets at a price of increased memory footprint. Considering the exactness of the output, starcode maintains a competitive performance in terms of running time.

The performance on artificial data is not always in agreement with the performance on experimental datasets. Typical experiments present additional difficulties. For instance, the sizes of the clusters may be uneven and the reads may contain near constant regions that usually degrade the performance of filter-based algorithms. We benchmarked sequence clustering algorithms on the problem of clustering TRIP barcodes (Akhtar et al., 2013). Briefly, the principle of TRIP (Thousands of Reporters Integrated in Parallel) is to tag reporter transcripts with random barcodes and measure the abundance of barcodes in the RNA as a proxy for gene expression. There is no reference to match aberrant barcodes against, because the tagging sequences are unknown.

The basic properties of the datasets used for benchmarking are summarized in Table 3. Dataset 1 (SRR950457) has been pre-processed to extract the barcode and remove the constant part of the reads. Only barcodes between 15 and 17 nucleotides were included in the file. Dataset 2 (PRJEB7686) consists of raw Illumina single reads. These datasets differ by the read size, the total read count and by the empirical cluster sizes. According to the output of starcode, the largest clusters of dataset 1 contain approximately 70,000 sequences, whereas dataset 2 contains 4 clusters with more than 1 million sequences. Dataset 3 (SRR950477) has been included to benchmark starcode against rainbow in paired-end clustering mode.

The running times of starcode, seed, slidesort, rainbow and cd-hit are summarized in Table 4. We accommodated the distance threshold for the first dataset to compensate for the reduced sequence length. Both starcode and slidesort were executed with the option ‘-d 2’ and the identity for cd-hit was set to ‘-c 0.85’. We were not able to run seed on dataset 1 due to limitations on the
minimum sequence length. Starcode was significantly faster than the other tools on all the datasets. Seed and cd-hit came in second position with a running time approximately 35 and 20 times greater on datasets 1 and 2, respectively. Rainbow was nearly an order of magnitude slower in the job of clustering paired-end reads. We did not record the exact running times past 10 days since this is several orders of magnitude higher than the running time of starcode.

The memory footprint of the different tools on the same datasets is shown in Table 5. The values represent the peak memory usage throughout the run on the datasets described above. On short reads (dataset 1), starcode outperforms the other tools taking advantage of the trie compaction. On dataset 3, starcode had a significantly larger memory usage than rainbow. Starcode and cd-hit used similar amount of memory on dataset 2. Both needed twice as much memory as slidesort, which has the advantage of not storing the complete graph during the all-pair comparison.

### 4 DISCUSSION AND CONCLUSION

Starcode is a solid algorithm for sequence clustering based on all-pairs matching. It achieves high precision, and on experimental datasets it can be faster than popular heuristics. By design, starcode is tailored to process high throughput sequencing data on multi-core platforms with sufficient amount of memory. Due to its superior precision and faster running time, it fills a gap among available software, by allowing to take full advantage of middle to high end hardware.

It is somewhat surprising that starcode is significantly faster than competing tools on experimental datasets, whereas seed and cd-hit are faster on artificial datasets. Starcode was developed ground up from TRIP experimental datasets and the poucet search was selected for giving the best empirical results. We speculate that the trie structure benefits from the entropy deficit that is typically observed in experimental data versus pseudorandom reads.

The speed and precision of starcode also makes it useful for other clustering tasks, such as identifying enriched motifs in microbial genomes and in experimental data. Here we have given two examples of such applications. In the first, we recover a known enriched 12-mer in the genome of *Neisseria meningitidis*.

In the second, we recover the motif of the human RNA binding protein SRSF1.
SRSF1 and notice that it seems to consist of two halves separated by a linker. This hypothesis is consistent with the fact that SRSF1 binds RNA through two consecutive RNA Recognition Motifs (RRM) that are known to bind 3–4 nucleotides in a row (Daubner et al., 2013). The Levenshtein distance, which incorporates insertions and deletions is more likely to capture bi-partite binding motifs than position weight matrix representations. The use of a clustering method to tackle this problem is unusual, but it illustrates the potential advantages of distance-based approaches.

One of the reasons why starcode appears to be faster than alternative tools is that it is designed to cluster relatively similar sequences. When clustering related but divergent sequences, the Levenshtein distance will have to be increased, leading to exponentially longer running times (Figure 5b). However, for the imporant practical case of correcting errors introduced by sequencing, starcode illustrates that there is still room for developing algorithms that are both faster and more accurate than the current state of the art.

ACKNOWLEDGEMENT

We would like to thank Maria Chatzou for her precious feedback on the preliminary version of this manuscript and Heng-Chang Chen for performing the Drosophila TRIP experiments.

Funding: This research was funded by the Government of Catalonia (Dept. of Economy and Knowledge) and the Spanish Ministry of Economy and Competitiveness (Centro de Excelencia Severo Ochoa 2013-2017) (SEV-2012-0208). P.C. fellowship is partly financed by the Spanish Ministry of Economy and Knowledge (Dept. of Economy and Knowledge) and the Spanish Ministry of Education. P.C. and C.C. were supported by a predoctoral fellowship of the Training Subprogram: predoctoral fellowships for the training of PhD students (FPI) 2013.

REFERENCES


