

# Fast clustering of large EST data sets within a fixed size of memory



Andrea Hansen, Korbinian Schneeberger, Na Yin, Xiao Liu, Andreas Groscurth, Stephen Rudd\* and Klaus Heumann

Biomax Informatics AG, Lochhamer Straße 11, D-82152 Martinsried, Germany  
\*Institute for Bioinformatics (MIPS), National Research Center for Environment and Health, Ingolstädter Landstr. 1, D-85764 Neuherberg, Germany

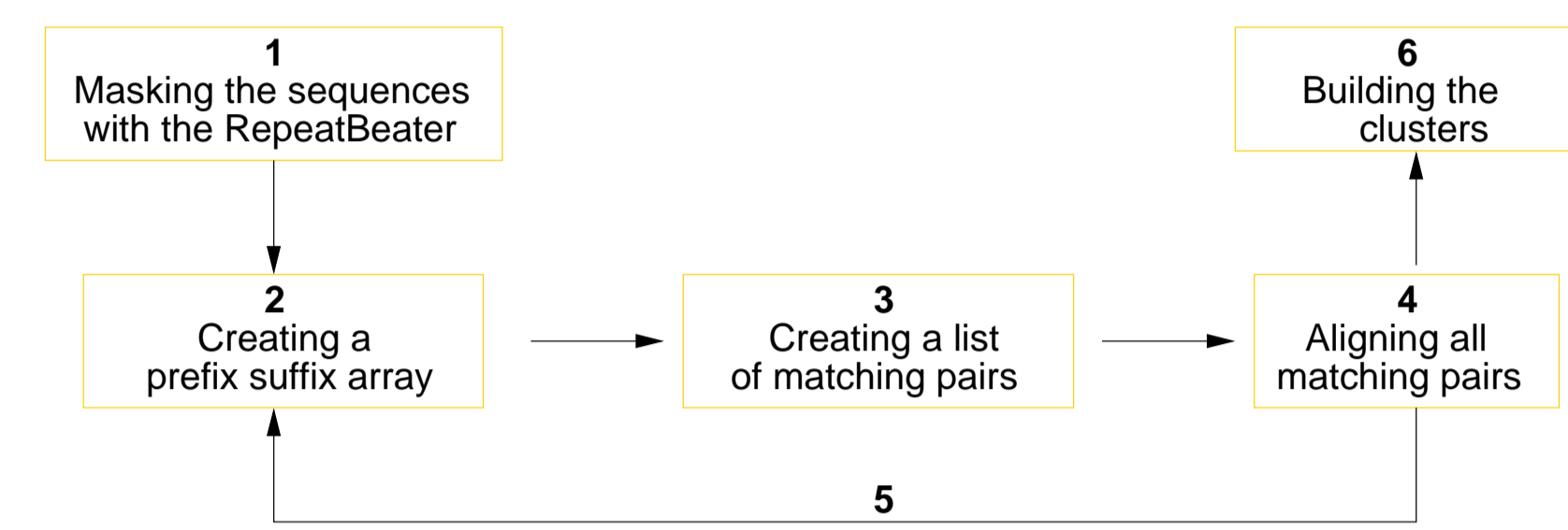
www.biomax.de

## Introduction

Expressed sequence tags (ESTs) are short DNA fragments derived from expressed genes. Clustering of ESTs is essential for obtaining information about gene identity and expression. We present a new clustering algorithm based on a suffix array. Successive processing of the sequences allows dynamic clustering within the RAM, which eliminates the need to additionally access slow mass storages. By assigning a reasonable amount of memory to the algorithm, the self-organizing memory control can be used to perform clustering on machines with little memory.

To avoid building clusters based on repetitive regions, we have developed a tool for masking repeats. RepeatBeater detects repeats (mono- to hexanucleotide) with a speed that is linear to the size of the input. Subsequently, detected repeats are disregarded by the clustering algorithm.

In addition to the simple alignment algorithm used during the clustering, a high-quality dynamic programming algorithm has been added to the application. This allows multiple queries to be performed on the whole data set using the same suffix array. The query result is an XML file containing an exhaustive alignment including score, identity, similarity and E value. These results are similar to the BLAST output.

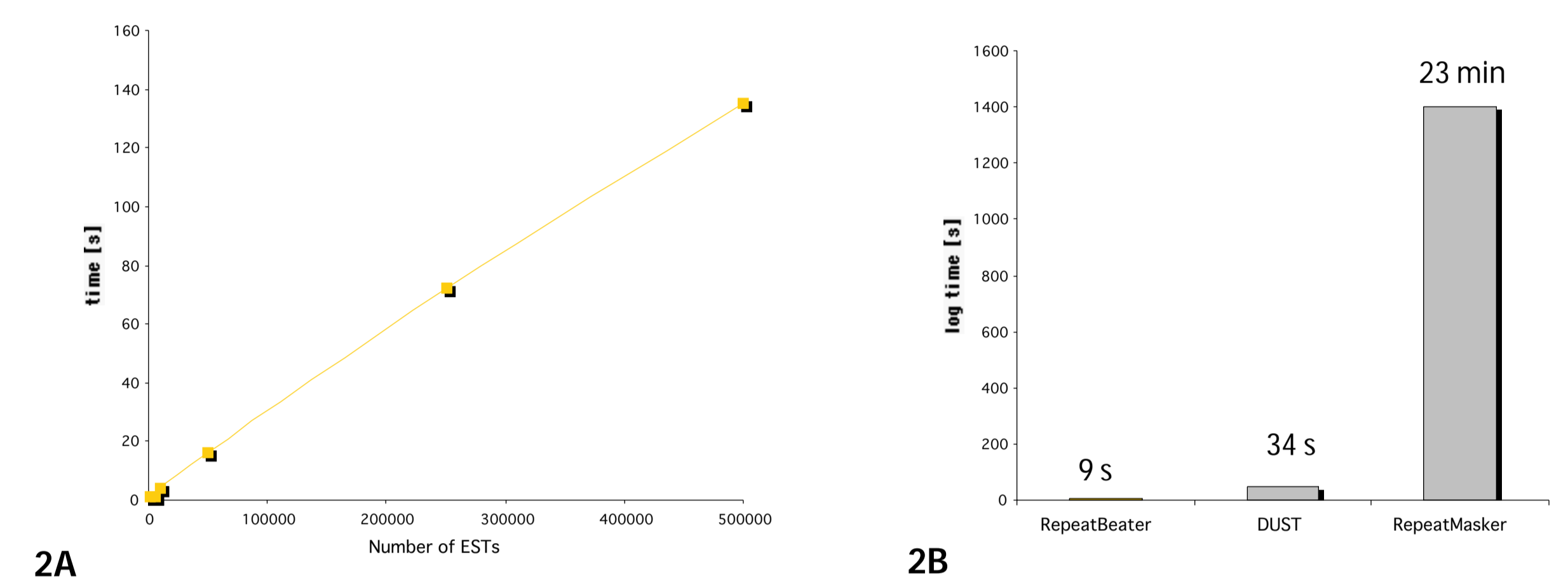


**Figure 1. Flowchart of the clustering:** 1. Masking the sequences with the RepeatBeater. 2. Reading the sequences until the memory is exceeded. 3. Matching all sequences against the 11-mers. 4. Aligning all matching pairs. 5. Repeating steps 2 to 4 until all sequences are matched. 6. Building the clusters

## Masking the repeats with the RepeatBeater

The RepeatBeater was developed for masking repeats within DNA sequences. The algorithm recognizes single repeats, 2-6-mers and low-complexity regions. It screens the sequences separately and finds the region to mask by counting the occurring bases. The RepeatBeater does not rely on alignments, therefore only identical matches can be detected. Cut-off values can be set to determine when to mask, which allows the sensitivity of the RepeatBeater to be altered without changing the running time.

We have compared the results of the RepeatBeater with the results produced by the RepeatMasker(TM) program and the DUST program (used by *blastn*, Tatusov and Lipmann). All three programs mask mono- to pentanucleotides in the same way, only DUST and the RepeatBeater mask hexanucleotides, the RepeatMasker does not recognize hexanucleotide repeats. **Figure 2A** shows how long it takes to mask a particular number of sequences with the RepeatBeater. **Figure 2B** shows a comparison of the performance of the three programs.



**Figure 2A. Performance test:** Increasing amounts of ESTs were processed by the RepeatBeater. **Figure 2B. Comparison with other masking software:** We have masked 28,014 ESTs (*A. thaliana*) with different software for masking repeats on an Intel-based PentiumIII machine. The RepeatMasker (Version 07/13/2002) was used with the option '-int'.

## Clustering of the masked ESTs

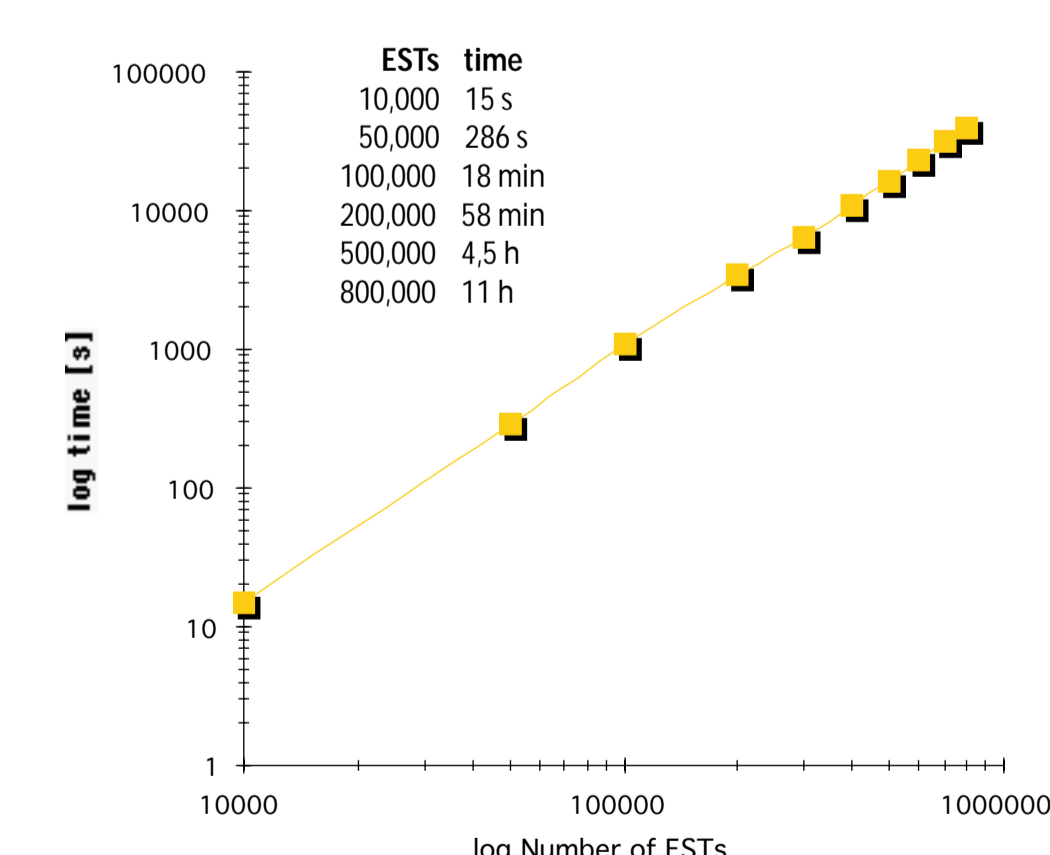
After repeat masking, the sequences are clustered with our new cluster algorithm. This algorithm is based on the Hashed-Position-Tree (1996, Heumann et al). We have modified this algorithm to increase the input size and the speed, and decrease the amount of memory which must be available for clustering.

The new Clusterer divides all sequences in 11-mers and stores them in an array, where the buckets represent all possible 11-mers in the given data set. This array could be seen as a prefix-suffix-tree. It allows a very quick and easy way to find exact and even nearly exact matches between sequences in  $O(n)$  time. This also eliminates aligning sequences which would result in low scoring. The information created through the array is used to determine whether to align two sequences or not. The flowchart in **Figure 1** shows the particular steps of the clustering which are explained in more detail in the following sections.

### Creating a prefix-suffix-array (Figure 1.2)

First as many sequences as possible are read into the array until the memory is exceeded. Each position of every sequence is treated as the beginning of an 11-mer. Each 11-mer created in this way is stored in an array in its 11-mer specific bucket with its position in the sequence and a sequence identification number. Memory control is dynamic and, therefore, can stop sorting the 11-mers when the algorithm reaches a predefined amount of megabytes.

Sequences which are not sorted in the array in this first step (or *rotation*) will be considered in the next rotation.



**Figure 3. Performance test of the Clusterer:** An increasing amount of ESTs (*H. sapiens*) were clustered on a single Intel-based PentiumIII machine for which 1500 MB were assigned to the process.

## Doing a blast-like database search with our Clusterer

Besides of the simple alignment algorithm which is sufficient for clustering sequences we have implemented a dynamic programming algorithm for producing a local alignment with a query sequence and a database. It runs with the same parameters as *blastn* and produces comparable results which are presented in XML format. Currently, *blastn* is a little faster than our Clusterer in the blast-modus. However, we plan to develop a parallel version of our Clusterer, i. e. the rotation steps will be done in parallel on several CPUs instead of one after the other on the same CPU. This will very significantly increase the performance of our Clusterer.

### Creating a list of matching pairs (Figure 1.3)

Two sequences are called a "matching pair" if they share at least one common 11-mer. All sequences (those already in the array and all others) are again divided in 11-mers to find all common 11-mers within the sequences sorted in the array. While collecting matching 11-mers, the algorithm checks for overlaps and regions with short local distances. They are recognized and merged to one common region. Even regions with only one to four nucleotides between the matching 11-mers are merged. Therefore, the regions are scored not only by length but also by similarity. The resulting score is called the match score.

The structures needed to store the matching pairs are allocated at the beginning of the algorithm and kept during the whole run time.

### Aligning all matching pairs with a certain match score (Figure 1.4)

After collecting all common 11-mers, a simple alignment algorithm aligns all sequence pairs sharing a block of overlapping (or nearly overlapping) 11-mers with a sufficient match score.

The resulting alignments include score, length and positions. This score is used in the final step. At the end of this rotation step the alignment results are written to hard disk to keep the memory available for the array and the matching structures. If there are still sequences which have not been sorted in the prefix-suffix-array the algorithm starts processing them in the next rotation (see **Figure 1.5**).

### Building the clusters (Figure 1.6)

At the end of the last rotation, the result file is read to build the clusters. Each sequence begins its own cluster and the final clustering is constructed through a series of iterations that can be described as transitive closure clustering. This term refers to the property that any two sequences with a given level of similarity (the alignment score) will be in the same cluster even if they share no similarity but there exists a sequence C with enough similarity to both A and B.

Cluster size	Number of Clusters		
	our Clusterer	CAP3	TIGR
1	5871	5790	7220
2	824	822	893
3	231	266	203
4	75	82	39
5	27	26	6
6	10	9	1
7	2	3	0
8	1	1	0
9	0	3	0
10	1	0	0
Number of non singletons	1402	1212	1142

Type of rearrangement	compared with CAP3	compared with TIGR	CAP3 comp. with TIGR Ass.
equivalent clusters	990	749	741
single split	8	34	35
simple merge	9	0	0
complex	378	619	436

4A

4B

**Figure 4 Comparison of our Clusterer with other programs:** We have clustered 10,000 ESTs (*A. thaliana*) with our Clusterer, with the CAP3 program (1995, Sutton et al.) and with the TIGR Assembler software (1999, Huang et al.). **4A.** Cluster size distribution for the three different programs. **4B.** Distribution of cluster events.

## Outlook

Our new developed RepeatBeater and the new clustering algorithm will be implemented in the new version of the HarvESTer(TM) EST Clustering and Assembly System. The HarvESTer system is a sophisticated high-throughput clustering and assembly tool for processing ESTs. Most of the modules of the HarvESTer system are already running in parallel on several nodes. Optimizing the new Clusterer for running in parallel as well will improve the speed of this system.

