# Mr. SymBioMath

High Performance, Cloud and Symbolic Computing in Big-Data Problems applied to Mathematical Modeling of Comparative Genomics

EU FP7 Industry-Academia Partnerships and Pathways
Project Nr. 324554

# Deliverable D3.1
# Fragments identification in Comparative Sequence Procedures

| | |
|---|---|
| Deliverable Number | **D3.1** |
| Deliverable Title | **Fragments identification in Comparative Sequence Procedures** |
| Type of Document | **Report** |
| Dissemination Level | **Public** |
| Workpackage | **WP3: Implementation and Local Testing** |
| Lead Beneficiary | **LRZ** |
| Contractual Delivery Date | **30.04.2015** |
| Actual Delivery Date | 24.04.2015 |
| Editor(s) | Oscar Torreno |
| Author(s) | Oscar Torreno, Oswaldo Trelles |
| Quality Reviewer(s) | Michael Krieger, Alex Upton |

## Document Log

| Version | Authors(s) | Date | Modifications |
|---|---|---|---|
| 0.1 | Paul Heinzlreiter | 12.02.2015 | Initial template |
| 0.2 | Oscar Torreno | 16.04.2015 | First draft |
| 0.3 | Oscar Torreno | 20.04.2015 | Supplementary material |
| 0.4 | Oscar Torreno | 24.04.2015 | Changes based on feedback from quality reviewers |

# Contents

## Executive Summary

In accordance with the technical annex [3] the deliverable D3.1 is focused on describing the fragments identification being developed within the scope of the Mr. SymBioMath project.

The procedures described here have been developed to address the needs of the bioinformatics use case of the Mr. SymBioMath project. The output of the fragments identification procedure serve as input data for further analysis/visualisation modules being developed within the project.

Note: Part of the text in this deliverable is taken from the article “Breaking the computational barriers of pairwise genome comparison”, submitted to the journal BMC Bioinformatics on the 10th April 2015.

# 1 Introduction

The number of genome sequencing projects has grown exponentially, in parallel with a drastic reduction in the cost of sequencing. At the turn of the millennium the cost of sequencing one Mbp of genomic DNA (million DNA base pairs) was about 10 thousand US dollars, compared to around 5 US cents at the time of writing[1]. Scientists are continuing to develop faster and cheaper methods that will allow the routine sequencing of individual patient genomes, thus truly ushering in the era of genetics-based personalised medicine.

It is not just the human genome that is of interest to the research community, the progression of sequencing technology has huge consequences for studies involving the genomes of other organisms. At present, hundreds of different organisms, from all living kingdoms, have been sequenced and thousands more projects are on-going. These developments have put Comparative Genomics into the spotlight in order to provide the tools for studying relationships within this flood of data.

Pairwise sequence comparison algorithms have been implemented since the early days of bioinformatics. Original algorithms for global [20] and local alignments [22] were designed using dynamic programming techniques that result in quadratic calculation time, and memory consumption proportional to the product of the total number of bases analysed.

When sequence analysis jumped from individual genes and proteins to full genomes, new approaches appeared, such as MegaBlast [10], MUMmer [14] and Gepard [12], the latter of which has been reported to be able to compare more than 300Mbp of human chromosome-1 in approximately one hour [12]. These software adopted some ideas introduced by the heuristic sequence database searching algorithms FASTA [21], and later BLAST [1]. These algorithms introduced a computational space reduction strategy based on the fast identification of matching points (hits) that are in turn used as seed points for the extension of local alignments. In FASTA, these matching points are perfect matches between K-mers (words of length k) from each sequence, while BLAST allows certain mismatches, thus enhancing its sensitivity. Other computational space reduction strategies confine the search to the most probable matching space (FASTA), or limit seed extension to regions with a minimal concentration of hits (BLAST). Additionally, such software adopted other ideas coming from the string matching field, such as the Suffix Array data structure [18], which significantly reduces the computational complexity but still keeps a significant use of memory resources.

In general, the reference software was designed to deal with genes, proteins and small genome sequences. Since these are now used for much larger datasets than they were originally designed for, they are now reaching their limit in terms of memory capacity and efficient computation on single-CPU systems. Consequently, there is a pressing need to design new software that tackles the memory consumption problem caused by the analysis of very large genome sequence datasets. A good strategy to deal with this problem is to move data that does not fit into internal memory to external memory (i.e. hard disks), following what is known as an out-of-core strategy [23, 16]. However, since there is a difference of several orders of magnitude in access time between the two memory layers, special care must be taken in order to avoid performance degradation. Some of these approaches have previously been applied to bioinformatics [13], but not specifically for pairwise genome comparison.

In this document we report on GECKO (GEnome Comparison with K-mers Out-of-core), a mod-

[1] http://www.genome.gov/sequencingcosts

ular application designed to identify collections of HSPs (High-scoring Segment Pairs) by pairwise genome comparison procedures, that can then be used to obtain gapped fragments. Our work improves on previous methods by introducing controlled memory usage and a modular design that allows further comparisons to be performed without the need to recalculate intermediate results and thus without sacrificing performance. We have benchmarked the application in terms of both performance and results quality. We designed experiments with datasets ranging from short sequences in the kilobase range, to larger sequences up to 200Mbp in length. This was done in order to compare GECKO against the best currently available software under both unfavourable and favourable conditions respectively. In addition, we performed a massive comparison exercise between mammalian chromosome sequences in order to test one of the key improvements of the application: the avoidance of intermediate result re-calculation. In the tests with short sequences, GECKO was slower compared to existing software, but with long sequences, the results were comparable or superior in terms of performance. The quality of results in both cases, short and long sequences, was superior. Detailed supplementary material and binaries are available at `http://bitlab-es.com/gecko/`.

## 2 Role of D3.1 within WP3

As described in the Mr. SymBioMath Annex I [3] the project design and development phases follow a bottom-up approach. In work package WP1 the state of the art has been described and the requirements for the Mr. SymBioMath software have also been established. In addition, the software interfaces have been defined, including user and web service interfaces.

Based on these requirements the design – and partially the implementation – of the software components has been done in WP2. The results have been reported in deliverable D2.1 [4] covering the areas of client applications describing the jORCA webservice client [19] as well as data compression through the libracio library [4, 15].

Deliverable D2.4 [5] wraps up the efforts within WP2 by providing a description of the full Mr. SymBioMath infrastructure, including the middleware software being deployed to support the bioinformatics applications.

Based on the results from WP2, WP3 now carries out most of the core implementation work of the project. This deliverable describes the fragments identification procedures which have been developed to support the application use case from the bioinformatics domain.

A crucial area where the state of the art has been significantly improved through Mr. SymBioMath is the comparison of large sequences. Compared to previous approaches, the fragments identification procedure reported in this deliverable addresses the memory consumption problem faced by existing methods.

## 3 Methods

To overcome the limitations of existing sequence comparison methods we focused firstly on the application-specific reduction of main memory and computational space usage, and secondly on modularising the process using classical software engineering concepts. We reduced memory usage using an out-of-core strategy designed to manage data structures that are too large

to fit into main memory at one time. Naturally, memory management could be delegated to the Operating System using virtual memory concepts; however poorer data locality can result in performance degradation in memory intensive applications such as large-scale sequence analyses.

To deal with this problem we have applied the following strategies to the design of GECKO (see Figure 1):

### 3.1 Memory consumption and computational space reduction

This section describes our approaches for dealing with the memory usage problem with an out-of-core solution, while compensating for the slower access time of secondary storage devices in several ways:

1. Sensitivity studies involve obtaining results for different $K$ values (word sizes) and require computing word dictionaries for each value. It is easy to realise that a collection of words of length $K$ contains almost all the prefixes with $K\prime < K$ (the last $K - K\prime$ K-mer at the end of each sequence is lost). Regardless of word length, the number of words is practically the same (sequence length $L - K + 1$; with $L >> K$). Therefore the dictionary is calculated only once using a large K value (K=32 by default).
2. Words are compressed on disk with a compression rate of 4 by using 2 bits per letter. This is possible because the K-mers are strictly composed of the $\{$A, C, G, T$\}$ symbols of the DNA alphabet.
3. Larger K values produce a lower number of word matches between sequences, mainly due to less frequent repetitions, and result in a greatly decreased number of potential seed points from which to extend the alignment. On-the-fly dictionary analysis can help in selecting the most appropriate K value. Additionally, a sampling procedure is incorporated to deal with highly redundant words.
4. It is possible to further reduce the number of selected hits by using a proximity criterion, whereby additional seed points must be separated by a minimum distance parameter from other hits in order to be extended.
5. The computed K-word dictionaries remain available for subsequent processing when comparing genomes, which significantly reduces I/O load.

To reduce computational space usage we followed a similar strategy to that used by some existing solutions, which depends on the identification of common K-mers present in both sequences that are then used as seed points for local alignments.

### 3.2 Modular design

As mentioned above, the second major improvement of our design was to modularise the process. The application is designed to be used for multiple genome data analysis, allowing for parameter sensitive studies as well as all-versus-all comparisons of genome collections. With the aim of reducing dependencies and repetitive actions, we organised the application workflow as follows (see Figure 1):

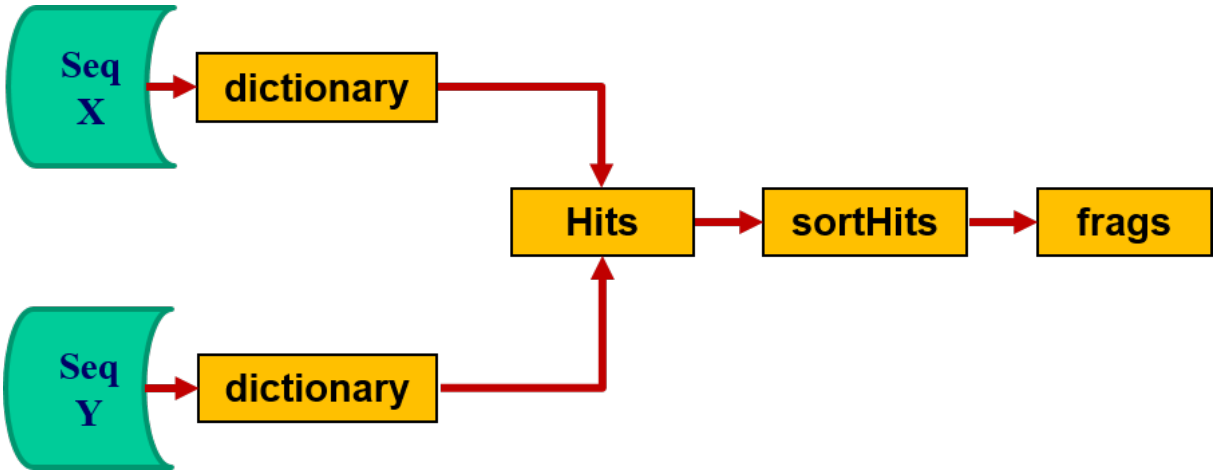



Figure 1: Summary of GECKO's modular design. The branches on the left represent dictionary computation, while the right side shows its detection and fragment extension steps.

1. One-off creation of a K-mer dictionary for each genome or sequence. The dictionary is stored on disk as a hash table, containing the words that appear in the sequence together with their positions.
2. Once calculated, K-mer dictionaries are then used to identify starting points (or hits) that will be used to obtain the HSPs. These seed points correspond to all possible matches produced between dictionary words. It is worth noting that the K value is parameterised at this point, with smaller K values derived as prefixes from the same dictionary.
3. Next, the application produces a local alignment (i.e. the HSPs) based on the calculated starting points, extending them in forward and reverse directions. From this point, all hits covered by a valid HSP are not analysed further.
4. To illustrate possible post-processing steps, several accessory modules have been developed. These include HSP visualisation (equivalent to the Mummerplot application in the MUMmer suite), data format converters to allow the use of other visualisation software packages, and further data analysis tools such as the K-mer frequency analysis program.

With minimal performance losses several software development features have been incorporated into GECKO to enable the development of a set of multi-platform applications. Examples include the usage of generic data types with the same representations in 32 and 64 bit architectures, the implementation of data access functions to read/write binary files in order to avoid Endianness problems and buffering strategies to minimise I/O operations and improve performance.

In the following sections we go into the details of each step performed by the GECKO application in chronological order.

### 3.2.1 Dictionary calculation

The dictionary calculation is based on the well-known binary tree in computer sciences. Each tree node contains a word (key) and its list of occurrences (values). Following the behaviour of a binary tree, left hand side nodes of a given tree come lexicographically before nodes on

the right hand side. To avoid memory consumption problems caused by the huge number of possible words (i.e. a theoretical maximum of $4^K$ different words, without counting repetitions), we decided to split the calculation into $p$ steps (with $p$ being a multiple of 4), thus reducing the amount of memory used by the program by a factor of $p$ (assuming a normal distribution of words). To split the dictionary and conserve its lexicographical order, a prefix of length $\log_4 p$ is used. This strategy requires us to iterate p times over the whole sequence, using a different lexicographically-organized prefix each time to preserve word order. To avoid memory allocation requests for each node, a single memory pool is reserved at the beginning of the process. New memory pools are then only reserved once the currently reserved memory is used up. To obtain the final result we traverse the tree in order, storing the word contained in the node together with the list of occurrences. We considered other strategies for this step, such as a prefix tree and a suffix array, but found that they experience memory consumption issues similar to the problems faced by existing software approaches.

### 3.2.2 Hits determination

The second section of the workflow corresponds to the identification of the starting points or seeds for the local alignment. If a word $w_i$ appears $n$ times in the first sequence at positions $p_j(j = 1...n)$; and the same word $w_i$ appears $m$ times in the second sequence at positions $p_k(k = 1...m)$, a hit will occur in all $(p_j, p_k)$ coordinates producing the following set $h = \{(1,1),...(1,m),(2,1),...(2,m),(n,1),...(n,m)\}$. All these hits are then considered starting positions for possible local alignments. Depending on how similar the sequences are and also on the K value used, the number of resulting hits could be very high. It is highly recommended to mask low complexity regions in order to reduce the hits produced by repetitive sequences. To reduce the number of hits further we have applied a proximity approach, by which those hits on the same diagonal, defined as $d = (p_j - p_k)$, and at a predefined distance are combined. This can be achieved quickly and easily by sorting hits by diagonal (and offset), what is performed using a threaded version of the quicksort algorithm, and then combining the hits that are within the distance parameter value.

### 3.2.3 HSP detection

The last calculation step consists of producing a set of ungapped HSPs that conform to a local alignment. An HSP is defined as a substring matching sequence whose positive accumulated score cannot be increased by extending the fragment at either of its extremes (i.e. until it attains a local similarity maximum between sequences). It starts from a hit with a positive score (the seed points identified in the previous section), and is extended along the sequence such that it accumulates score until the overall HSP score becomes negative or the end of one of the sequences is reached (or both simultaneously). Fragment boundaries are positions that give the highest accumulated score at both ends as HSPs are extended in both directions along the sequence (forward and backward). The algorithm continues searching for HSPs within the next hit in the diagonal or the first one of the next diagonal. If the next hit in the same diagonal has been covered by extension of the previous HSP, it would not be used because it will result in a redundant sub-HSP within the previous one. GECKO outputs a set of identified HSPs that are defined by starting and ending coordinates in both sequences, together with HSP length, score and identity levels.

### 3.2.4 HSP post-processing

Almost all existing methods provide a way of graphically representing local alignments after computation. GECKO incorporates its own visualisation procedure that generates a PNG file as well as the ability to output its analyses in formats that can be processed by the visualisation methods included with existing analysis programs. In addition, GECKO includes post-processing applications that enable tasks such as the ability to apply additional filters to HSP collections or generate gapped alignment constructions based on ungapped ones.

# 4 Results

## 4.1 Dataset

The selected test dataset contains sequences of different sizes in order to thoroughly compare GECKO with other state-of-the-art methods under both favourable (large sequences) and unfavourable (short sequences) situations. Specifically, the dataset is composed of short (virus), medium (bacteria and fly), and large (mammalian) sequences (see Table 1 for sequence names and their GenBank accession numbers). The large mammalian sequences will also be used for an all-versus-all experiment.

| Species | Strain / Chromosome | Accession number | Mbp |
|---|---|---|---|
| Tomato Yellow Leaf Curl Virus | TYLCV | GenBank:AM409201.1 | 0.004 |
| Tomato Yellow Leaf Curl Virus | TYLCV-lr2 | GenBank:EU085423.2 | 0.004 |
| Buchnera aphidicola | APS (Acyrthosiphon pisum) | GenBank:NC_002528.1 | 0.636 |
| Buchnera aphidicola | 5A (Acyrthosiphon pisum) | GenBank:NC_011833.1 | 0.640 |
| Escherichia coli | K-12 | GenBank:NC_000913.2 | 4.596 |
| Escherichia coli | O157:H7 Sakai | GenBank:NC_002695.1 | 5.448 |
| Drosophila melanogaster | chromosome 2R | GenBank:NT_033778.3 | 20.948 |
| Drosophila pseudoobscura | strain MV2-25 chromsome 3 | GenBank:NC_009006.2 | 19.604 |
| Homo sapiens | chromosome 1 | GenBank:NC_000001.11 | 246.600 |
| Pan troglodytes | chromosome 1 | GenBank:NC_006468.3 | 226.172 |

Table 1: Dataset information. From left to right: Species name, strand and/or chromosome of origin, GenBank accession number and size in Mbp.

## 4.2 Infrastructure and reference software

GECKO performance will be compared against equivalent state-of-the-art applications such as Gepard [12], MUMmer [14], Mauve [6], LASTZ [9] and LAST [7, 8, 11]. Either the source code or pre-compiled binaries were downloaded from the sources provided in the corresponding manuscripts. GECKO was compiled using GNU C Compiler (GCC) version 4.8.2, with "-O3" and "-D_FILE_OFFSET_BITS=64" compiling options (in the same way reference software packages were compiled). All the reference software was used in their command line versions in order to do a fair comparison with GECKO which is also executed through the command line (more details about execution parameters in the Section 7.3.2 of the supplementary material).

The tests reported in this document were performed using an Openstack cloud instance configured with 4 Intel Xeon E312xx (Sandy Bridge) 2.0GHz equivalent cores, 8GB of RAM and the Ubuntu 12.04 LTS 64-bit operating system. For storage, a 300GB Openstack volume was used.

The underlying physical disks of the Openstack setup were conventional ones (500GB, 16MB buffer, SATA 3, 7200 RPM). The cloud instance was deployed within the RISC Software GmbH cloud facilities in Hagenberg, Austria. Due to the inability of some current software to run in the mentioned infrastructure with large sequences (see the notes of Table 2), we additionally used Picasso shared memory multiprocessor located at the University of Málaga (Málaga, Spain). It contains 7 nodes, each with eight Intel E7-4870 processors which delivers 96 Gflop/s each, giving a peak performance of 5 Tflop/s. Each node has 2 TB of RAM giving an aggregate memory of 14 TB.

Results shown in this section (Table 2) correspond to sequential (one core) execution of each module except for the hit sorting method that used 8 threads running on one 4 core CPU. Further benchmarks will be included in following deliverables.

| Comparison | Gepard | MUMmer | Mauve | LASTZ | LAST | GECKO |
|---|---|---|---|---|---|---|
| TYLCV-TYLCV-Ir2 | 0.84 | **0.00** | 0.06 | 0.04 | **0.00** | 0.36 |
| BuchneraAPS-BuchneraBp | 2.56 | **0.44** | 6.73 | 0.46 | 46.20 | 1.60 |
| E.colik12-E.coliO157 | 33.12 | 10.63 | 45.92 | **1.83** | 109.00 | 17.20 |
| D.Melanogaster-D.Pseudoobscura | 238.34 | 45.99 | 294.92 | **19.64** | 1593.00 | 48.72 |
| H.Sapiens-Chr1-P.Troglodytes-chr1 | **7084.00**$^{*1}$ | 23226.00$^{*1}$ | >604800.00 | 78360.00 | n.a. | **11848.15** |

Table 2: Execution time in seconds for the comparison of the sequences listed in Table 1 under "Pairwise comparison". The comparison of mammalian chromosomes was also included to test the ability of GECKO and reference software packages to function when analysing very large datasets. The dictionary calculation time is included in the reported times, since the dictionary were not pre-calculated. "n.a." indicates that resource problems prevented analysis execution and the presence of ($^{*1}$) after some execution times indicates that the time was measured in a bigger machine because in such cases they were using more than 8GB of memory.

### 4.3 Results Quality

Although the performance aspects of GECKO's design are crucial, the production of high quality results is equally important. In this section we explain how we evaluated the quality of the results produced by our algorithm versus the other applications using the same parameters. The rationale behind our evaluation was to compare the coverage of the HSPs detected by each algorithm. To avoid biases in the evaluation we decide to obtain a consensus set of reference HSPs. This set is composed of those HSPs reported by at least half of the reference algorithms. The HSPs produced by GECKO were then mapped over the reference HSPs and the percentage of coverage recorded as a measurement of result quality. This means that matching positions reported by the consensus HSP reference and not reported by GECKO will push down the quality and vice versa. There are more sophisticated ways of comparing the results, such as only considering coding regions, or by qualifying and weighting matches depending on sequence type or section. However, we decided not to use these methods as they can incorporate noise or biases into the evaluation.

Following this procedure, the evaluation determined that in our experiments GECKO detected 3% more HSPs than the consensus set. Moreover, GECKO obtained a larger dataset while maintaining identity values over 65%, thus representing the identification of additional statistically-significant HSPs.

### 4.4 Visualisation

Strictly speaking GECKO is not intended for dotplot-like visualisation. However, we provide three alternatives: (1) two different programs able to generate 2D representations, one for single pairwise comparison results, capable of analysing forward and reverse HSPs; and the second for multiple comparisons whereby all comparisons are projected over one of the sequences selected as the reference. Obviously, any of the compared sequences can be used as the reference; (2) small plugins that allow GECKO results to be converted into formats compatible with commonly used visualisation methods; (3) further visualisation procedures developed within the Mr. SymBioMathproject such as 3D-Scover in its desktop and immersive implementations, and also the web visualisation (more details in the deliverable D3.2, sections 3.2.2, 4.2 and 8.3.4 respectively).

## 5 Discussion

Considering that GECKO's implementation was designed primarily for very large sequence comparisons, it compares surprisingly well with the reference software packages when analysing short sequences. It is as fast as Gepard even when the dictionaries were not pre-calculated. Gepard reports 33 seconds for 5Mbp genomes, compared with 17 seconds for our implementation. In the cases of MUMmer, LAST and LASTZ, our execution time was greater, due to the different strategy we are following compared to the suffix array indexing they are using, but still the difference is acceptable since the execution time is not that high. However, for longer sequences, our method strongly outperforms existing methods. GECKO needed less than 2 hours on average to compare chromosome 1 from different species (all possessing more than 120 Mbp) against the 3 hours and a half on average of Gepard and MUMmer and the 29 hours of LASTZ. Since all the reference software packages manage data structures in core memory, their good performance with short sequences was predictable, but this also means that their performance degrades as sequence size grows, entering into starvation when no more computational resources are available. This is due in part to the use of the Suffix Array data structure which on one hand reduces the computational complexity but on the other increases the memory consumption up to 9 times the length of the input sequence in the most efficient implementations.

GECKO's implementation showed real-world performance gains ranging from 133% versus Gepard for TYLCV comparison, to 3269% versus LAST in the case of Drosophila comparison (see Table 2).

### 5.1 In- or out-of-core implementations and modularity

Traditionally, bioinformatics programs, in common with conventional software development practices, are designed to perform calculations with the data loaded in main memory in order to take advantage of the difference in access time between main and external memory. However, the growth rate of available data has surpassed the growth of the typical amount of RAM memory available, making it impractical to keep all the data in core. Consequently there is a pressing need to re-design trusted software packages, as well as to develop brand new software strategies to tackle this problem.

The out-of-core implementation used in GECKO has the following advantages:

1. It removes any dependence on K-mer size, giving rise to the possibility of using small prefixes for short sequences and bigger values for larger ones. We have identified 32 as a maximum K value that gives the exact matches that are useful for this type of application, especially while comparing distantly-related sequences. Greater K values did not produce enough seed points for a meaningful comparison (even with chromosome or genome-sized datasets).
2. Working in disk allows word dictionaries computed by previous program instances to be preserved in secondary storage, thus reducing the time required for multiple comparison studies.
3. The modular implementation of GECKO stores intermediate results to disk, which facilitates the development of small and simple software components that allow the exhaustive analysis of the program's final output, as well as intermediate data such as word frequencies, word structure, comparative studies, extreme frequency analysis, functional genomics annotation and data visualisation. This method for organising execution even allows interactive analysis, with the possibility of re-executing specific parts of the analysis with different parameters.

### 5.2 K-mer size parameter

It is not difficult to deduce from all of the above that the time needed to complete each analysis is determined by word size (K), and strongly affected by both noise and the algorithm's seed point detection sensitivity. K-mers are stored as K=32 to avoid having a large collection of dictionaries for each K value. K=32 contains all the K-mers for $K\prime < K$ with no additional processing, values that are especially useful to obtain enough exact matches for distant sequences. The software is designed such that it can be used with K values greater than 32 in case future sequences and/or applications require such a change. Using an incorrect K value will degrade performance due to the large number of K-mers repetitions. To avoid starvation GECKO uses a sampling scheme for very common repetitions.

## 6 Conclusions

This document presents GECKO, a pairwise genome comparison application based on an enhanced reduction of memory consumption and computational space, combined with a modular out-of-core implementation with several important advantages, including K value independence, complexity reduction, high performance and high results accuracy.

Additionally, software components can be easily added to this application to extend its capabilities in the spirit of software developer collaboration. New modules can be added without needing any change to the current architecture. Example programs currently available include: K-mer frequency calculation, analysis of over- and under-represented word sets, pre-visualisation monitoring tools and full construction of local ungapped fragments including their alignment.

A set of benchmarks demonstrates the effectiveness of GECKO's implementation, even on a single CPU machine.

GECKO does not require custom software or libraries to run. It can be executed within a variety of computing environments, from simple desktop PCs to more complex architectures such as clusters.

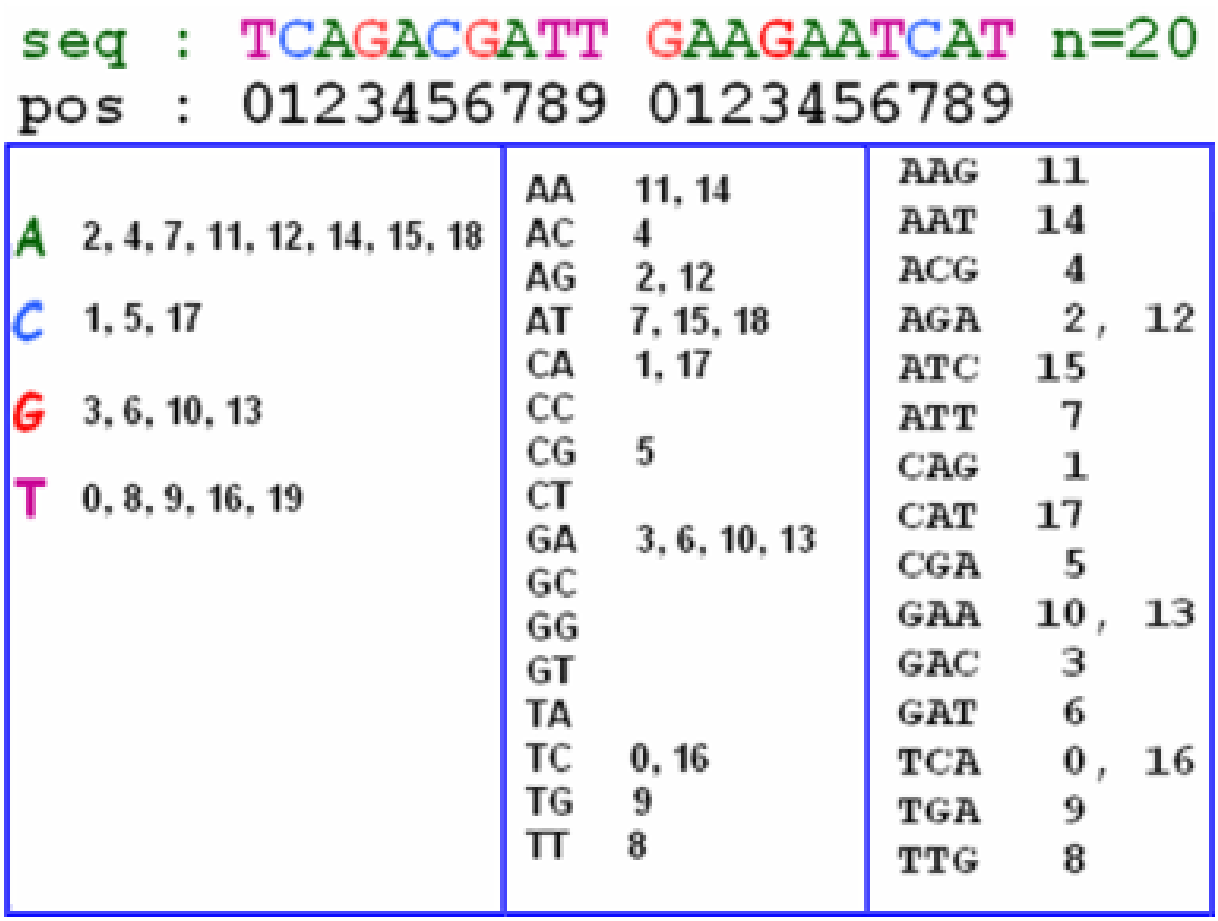
This software aims to facilitate massive comparisons of genome-sized sequences, as well as more complex evolutionary studies. Currently the output provided by this program is being used to identify evolutionary events such as inversions, transpositions and gene duplications. These studies have already provided new insights into evolutionary models of populations and species [2], as well as comparative metagenomic studies [17].

# 7 Supplementary material

## 7.1 Computational space reduction

### 7.1.1 Words

To reduce the computational space and accelerate data processing most of the proposed strategies uses some kind of pre-processing step. K-mers are used as prefixes for fast identification of matching words to be used as seed points from where to extend the local alignment.

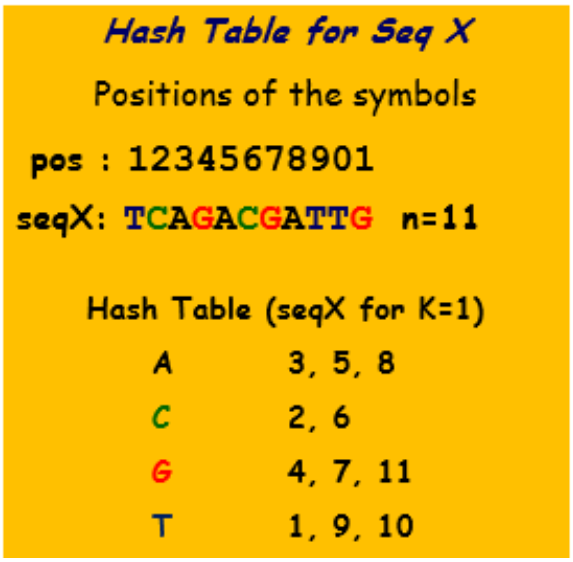


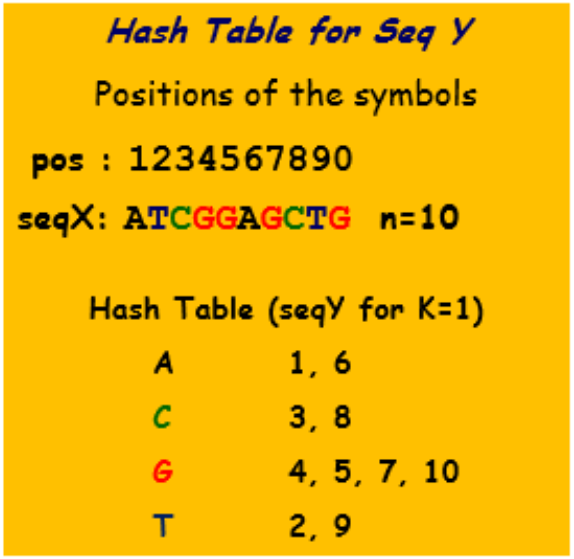

In the image a hash table using different “prefix” length (K=1, 2, 3). The header contains the “word” and the table contains the positions in which that word appears in the sequence. The longer the prefix is, the shorter the number of “occurrences” is. In this case, the hash is built-up by “full-identity”, thus all the words are the same for a given header entry (this is a tradeoff between sensitivity and memory requirements). The number of putative hash-headers is 4K where K is the word length, however NOT all the combinations are present in the sequence. The exact number of words is L-K+1 being L the sequence length.

### 7.1.2 Hits

Hits are word matches to be used as seed points. The number of hits depends on the number of matching-word repetitions, and it depends also on the word size (K).

Example:

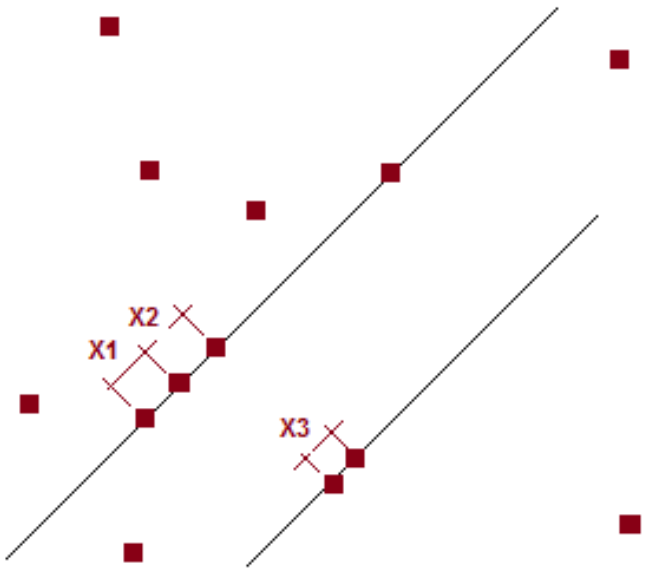


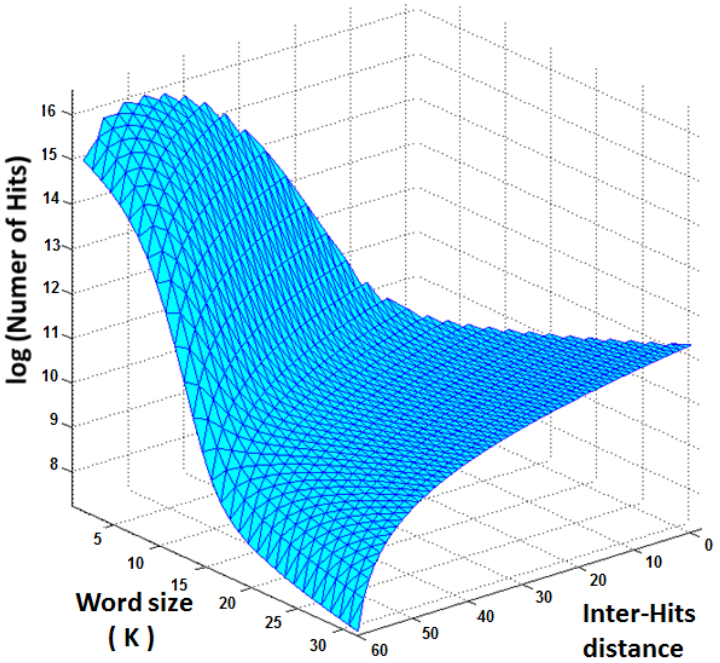

Identical words produce hits in the coordinates they appear (diag= h - v, when xh matches yv) (A) Produces hits in : (3, 1), (3,6), (5, 1), (5, 6), (8, 1) and (8,6) (C) Produces hits in : (2, 3), (2, 8), (6, 3) and (6,8)

### 7.1.3 Big-hits

To reduce the computational space even further, the hits (i.e. matches of words coming from the sequences under comparison) are joined together based on their proximity.



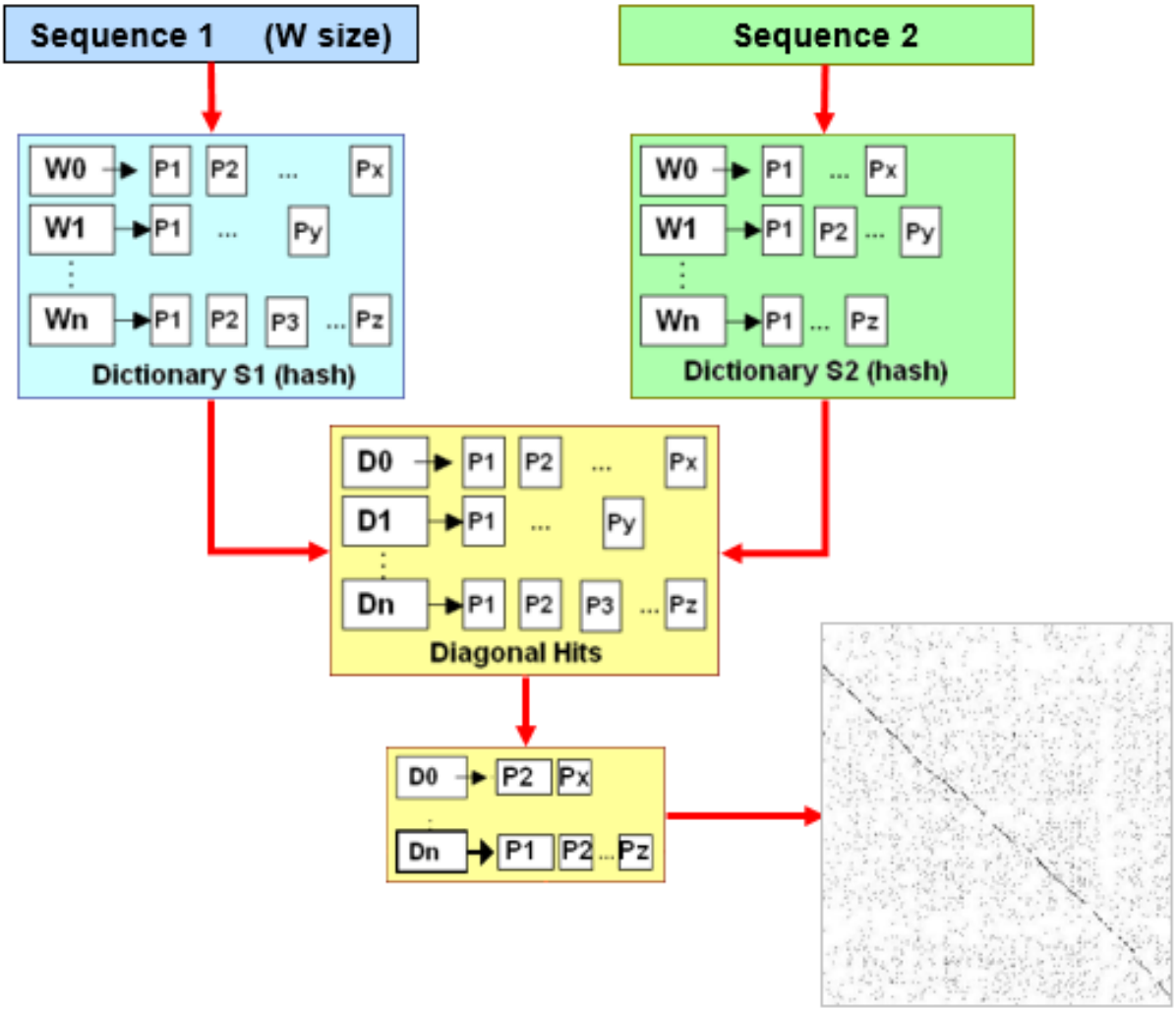

Hits in the same diagonal at a distance shorter that a parameter D will be joined to form a "big-Hit" (in the image the three hits in the first upper diagonal (at distances $X1, X2 < D$) and the two hits in the second diagonal $(X3 < D)$ are joined to form a "3BigHit" and "2BigHit".




Behavior of the seed-points computational space as a function of K (word length) and the inter-hits distance parameter used to group neighbor hits. Real data (chromosomes X from several species) have been used in the simulation.

## 7.2 Modular design

### 7.2.1 The global idea




**Pre-processing**

Masking low complexity regions.

**Hashing**

Including sub-processes such as sorting, grouping, etc.

**HITS by diagonal**

Including BigHits detection and pruning.

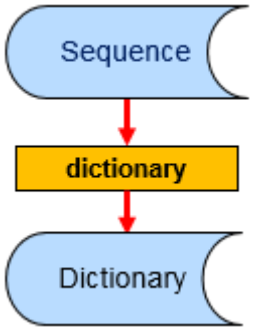
**HITS Extension**

Search for similarities using hits as seed points.

**Post-processing**

Visualization, frequencies, words distributions, etc.

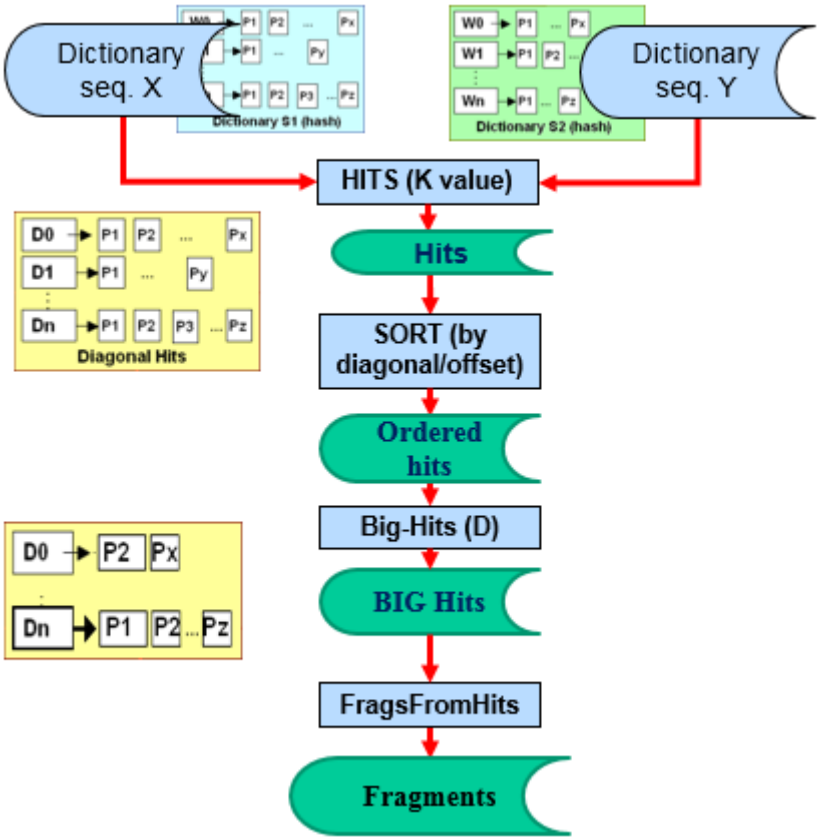
### 7.2.2 First step: Building the dictionary




The dictionary calculation is based on the well-known binary tree in computer sciences. Each tree node contains a word (key) and its list of occurrences (values). Following the behaviour of a binary tree, left hand side nodes of a given tree come lexicographically before nodes on the right hand side. To avoid memory consumption problems caused by the huge number of possible words (i.e .a theoretical maximum of $4^k$ different words, without counting repetitions), we decided to split the calculation in p steps (with p being a multiple of 4), thus reducing the amount of memory used by the program by a factor of p (assuming a normal distribution of words). To split the dictionary and conserve its lexicographical order, a prefix of length $log_4 p$ is used. This strategy requires us to iterate p times over the whole sequence, each time using a different prefix in lexicographical order to preserve word order. Another improvement is the reservation of a memory pool at the beginning of the process to avoid memory allocation requests occurring for each node. Instead, we request for a pool of memory and new memory pools are then only reserved once currently reserved memory is used up. To obtain the final result we traverse the tree in order, storing the word contained in the node together with the list of occurrences. We considered other strategies in this step, such as a prefix tree, but found that they experience memory consumption issues similar to the problems faced by existing software approaches.

It is worth noting that this process needs to be done only once for each sequence (or twice if the reverse complementary is going to be also analyzed). The dictionary is computed for $K = 32$ which contains all the prefixes for $k\prime < K$.

### 7.2.3 Second step: Alignment from hits




Starting: dictionaries of the sequences to compare

Hits: hits production based on identical words (the size of K can be redefined) to increse sensitivity, which is particularly interesting for remote homologs sequences.

Sort: by diagonal and offset in the diagonal

Big-Hits: join hits by proximity

FragsFromHits: The main procedure. Extends the seed-points to produce the local ungapped alignment

| Module | Description | Input/outputs |
|---|---|---|
| Dictionary | Building a k-mers dictionary.<br>- Words scanning<br>- Organize words in a hash table (disk) | Sequences / Sequence dictionaries |
| Hits | The same word in both seqs will produce a hit | Dictionaries for each sequence / Hits (diag, X,Y) |
| BigHits | Seeds identification<br>- Order the collection of hits (sortHits)<br>- Identify consecutive hits as big-hits<br>- [filtering of isolated hits] | Hits collection (diag, posX, posY) /<br>Reduce Big-Hits collection (diag, posX, posY) |
| Fragments | Un-gapped fragment detection by extension of seed points | Big-Hits collection / Un-gapped fragments |
| Post-Process | Post processing (available)<br>- Words frequencies<br>- Fragments distribution (Length, Score)<br>- Dotplot visualization<br>- Detailed fragment composition | Several of intermediate files / Several outputs |

Other: several tools can be used for postprocessing

## 7.3 Benchmarking

### 7.3.1 Dataset

| Species | Strain / Chromosome | Accession number | Mbp |
|---|---|---|---|
| Tomato Yellow Leaf Curl Virus | TYLCV | GenBank:AM409201.1 | 0.004 |
| Tomato Yellow Leaf Curl Virus | TYLCV-Ir2 | GenBank:EU085423.2 | 0.004 |
| Buchnera aphidicola | APS (Acyrthosiphon pisum) | GenBank:NC_002528.1 | 0.636 |
| Buchnera aphidicola | 5A (Acyrthosiphon pisum) | GenBank:NC_011833.1 | 0.640 |
| Escherichia coli | K-12 | GenBank:NC_000913.2 | 4.596 |
| Escherichia coli | O157:H7 Sakai | GenBank:NC_002695.1 | 5.448 |
| Drosophila melanogaster | chromosome 2R | GenBank:NT_033778.3 | 20.948 |
| Drosophila pseudoobscura | strain MV2-25 chromsome 3 | GenBank:NC_009006.2 | 19.604 |
| Homo sapiens | chromosome 1 | GenBank:NC_000001.11 | 246.600 |
| Pan troglodytes | chromosome 1 | GenBank:NC_006468.3 | 226.172 |

Table 3: Dataset information. From left to right: Species name, strand and/or chromosome of origin, GenBank accession number and size in Mbp.

### 7.3.2 Reference software

The reference software used for comparison purposes are:

1. Gepard (version 1.30) In this case we used the command line version (gepardcmd.sh). An execution line example:

   ```
   gepardcmd.sh -seq1 S1 -seq2 S2 -matrix matrix.mat -outfile plot.png
   ```

2. MUMmer (version 3.23) In this case we ask MUMmer to report all the matches (as GECKO is doing), because with the default options it is reporting only the maximal unique matches (MUMs). We ask MUMmer also to use compare only (A, C, G, T) with "-n" option and to do it with both sequence strands "-b". An execution line example:

   ```
   mummer -maxmatch -n -b S1 S2
   ```

3. Mauve (version 2.3.1) In this case we did not use Mauve graphical user interface but the command-line program progressiveMauve with "—mums" to report the maximal unique matches as MUMmer is doing, "—skip-gapped-alignment" parameter is used because GECKO is calculating ungapped HSPs, and also specifying the score matrix. An execution line example:

   ```
   progressiveMauve –mums –skip-gapped-alignment –substitution-matrix=matrix.mat
   –output=mauve.txt S1 S2
   ```

4. LASTZ (version 1.02.00) In this case we ask LASTZ to use overlapping words during the search ("—step=1"), also to calculate ungapped HSPs ("–nogapped") and everything in both strands ("—strand=both"). An execution line example:

   ```
   lastz S1 S2 –step=1 –nogapped –strand=both –format=maf –scores=matrix.mat
   ```

5. LAST (version 545) LAST first calculates a database (similar step to the one GECKO is performing) with the following command:

```
lastdb S1DB S1
```

And then it performs the comparison. Specific parameters change the maximum number of initial matches per query position (-m 1000) in order to make it comparable to the limit of also 1000 used in GECKO. It was executed using the same scoring scheme ("-r" and "-q"), for both strands ("-s 2"), calculating local alignment ("-T 0"), with a minimum initial match of 32 ("-l") and with an overlapping window along the query sequence of step 1 ("-k", similar to the "–step=1" of LASTZ). An execution line example:

```
lastal S1DB S2 -m 1000 -r 4 -q 4 -s 2 -T 0 -l 32 -k 1
```

All the programs are using the following scoring matrix present in LAST program in order to do a fair comparison (or equivalent match/mismatch parameters in the cases they are not accepting a scoring matrix parameter):

| | A | C | G | T |
|---|---|---|---|---|
| A | 4 | -4 | -4 | -4 |
| C | -4 | 4 | -4 | -4 |
| G | -4 | -4 | 4 | -4 |
| T | -4 | -4 | -4 | 4 |

### 7.3.3 Execution time

| Comparison | Gepard | MUMmer | Mauve | LASTZ | LAST | GECKO |
|---|---|---|---|---|---|---|
| TYLCV-TYLCV-Ir2 | 0.84 | **0.00** | 0.06 | 0.04 | **0.00** | 0.36 |
| BuchneraAPS-BuchneraBp | 2.56 | **0.44** | 6.73 | 0.46 | 46.20 | 1.60 |
| E.colik12-E.coliO157 | 33.12 | 10.63 | 45.92 | **1.83** | 109.00 | 17.20 |
| D.Melanogaster-D.Pseudoobscura | 238.34 | 45.99 | 294.92 | **19.64** | 1593.00 | 48.72 |
| H.Sapiens-Chr1-P.Troglodytes-chr1 | **7084.00**$^{*1}$ | 23226.00$^{*1}$ | >604800.00 | 78360.00 | n.a. | **11848.15** |

Table 4: Execution time in seconds for the comparison of the sequences listed in Table 1 under "Pairwise comparison". The comparison of mammalian chromosomes was also included to test the ability of GECKO and reference software packages to function when analysing very large datasets. The dictionary calculation time is included in the reported times, since the dictionary were not pre-calculated. "n.a." indicates that resource problems prevented analysis execution and the presence of ($^{*1}$) after some execution times indicates that the time was measured in a bigger machine because in such cases they were using more than 8GB of memory.

The execution time in the case of Gepard contains "*" because the program suffered memory consumption problems in our test infrastructure. This does not correspond with what they report, see the table below extracted from: "Krumsiek, Jan, et al. (2007); "Gepard: a rapid and sensitive tool for creating dotplots on genome scale"; Bioinformatics Vol. 23 no. 8"

| Sequence length | DOTTER | Gepard | Gepard pre-SA |
|---|---|---|---|
| 10 000 bp | 2 s | <1 s | <1 s |
| 50 000 bp | 30 s | <1 s | <1 s |
| 100 000 bp | 2 min 4 s | <1 s | <1 s |
| 1 000 000 bp | 2 h 10 min | 5 s | 4 s |
| 5 000 000 bp | 52 h 38 min[a] | 47 s | 40 s |
| Human chrom. I | 382 years[a] | 61 min | 53 min |

We confirmed this by executing Gepard in a bigger machine. The execution reported the use of almost 50GB of memory and a total time of 1 hour, 58 minutes and 4 seconds. We think the difference in execution time with GECKO resides in the fact that Gepard is not reporting a list of the resulting HSPs (at least we didn't find the parameters to do it), what in this case of a big file seriously reduce the execution time. Instead, Gepard reports a 'png' image which provides a visual overview of the similarities shared by the sequences under comparison.

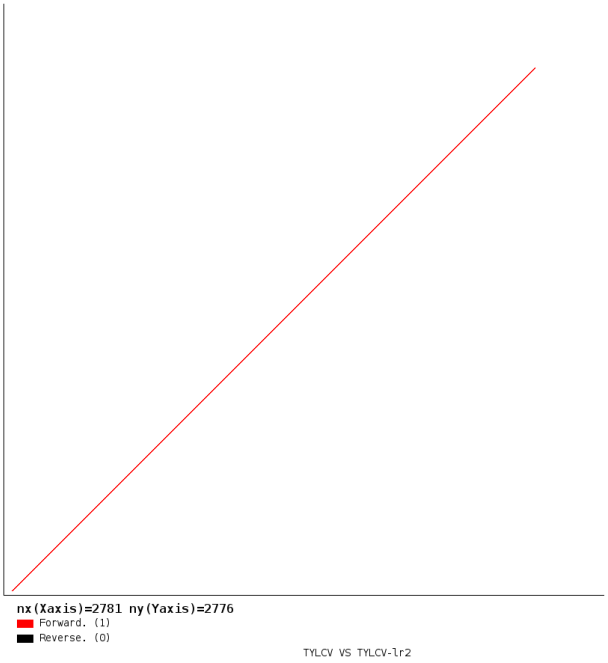
In the case of Mummer, the table contains "*" because it also suffered memory consumption problems in our 8GB infrastructure. We confirmed this fact with a later execution in a bigger machine. Anyway the execution time is greater than the one reported by GECKO (6 hours, 27 minutes and 6 seconds compared to 3 hours, 17 minutes and 24 seconds calculated as dictionary calculation time plus comparison time of GECKO).

In the case of LAST, the table contains "n.a." because it suffered memory consumptions problems in our 8GB infrastructure. In this case the execution was even not possible in a bigger machine because of the more than 300GB used in the maximum resident size). This is in contrast with the statement they make in their webpage (`http://last.cbrc.jp`): "Human vs. mouse. This took about 1 day on 1 CPU, and less than 2 GB of RAM." We believe it is because of the parameters we are using, but in order to do an apples-to-apples comparison with GECKO the parameters need to be fixed to such values. In any case, the execution time reported until it crashed it is much greater compared to the one of GECKO (39 hours 57 minutes and 57 seconds in LAST compared to 3 hours, 17 minutes and 24 seconds calculated as dictionary calculation time plus comparison time of GECKO).

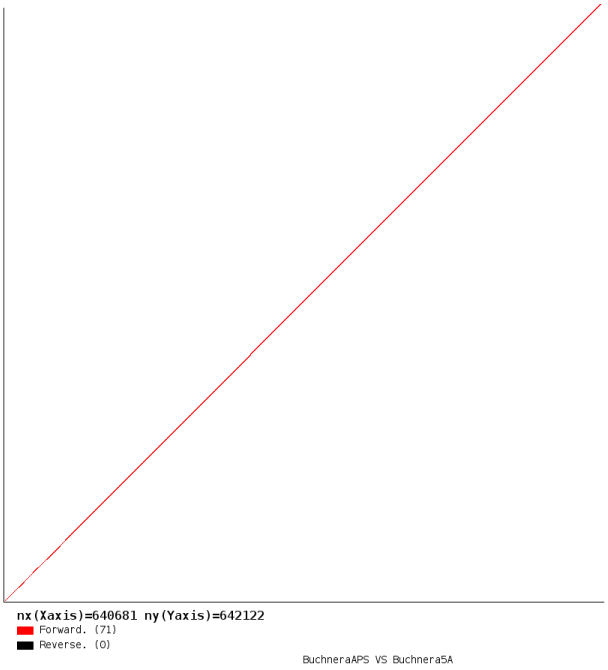
In the case of Mauve (i.e. progressiveMauve) in the table appears "$> 604800.00$" what means that the execution time is greater than a week (the maximum execution time of a job in our bigger system). Since the job was cancelled we did not obtain the memory consumption, that is why the table contains "n.a.".

### 7.3.4 Resulting dotplots

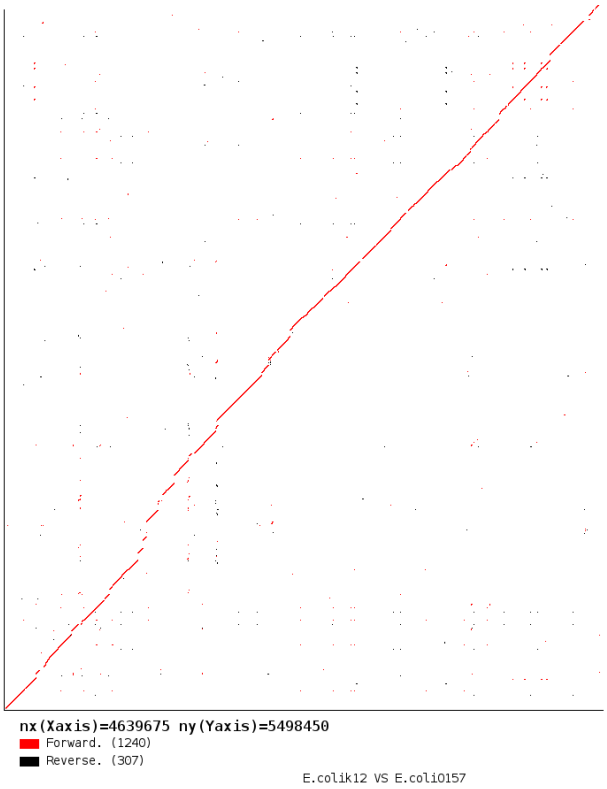
**TYLCV vs. TYLCV-Ir2**



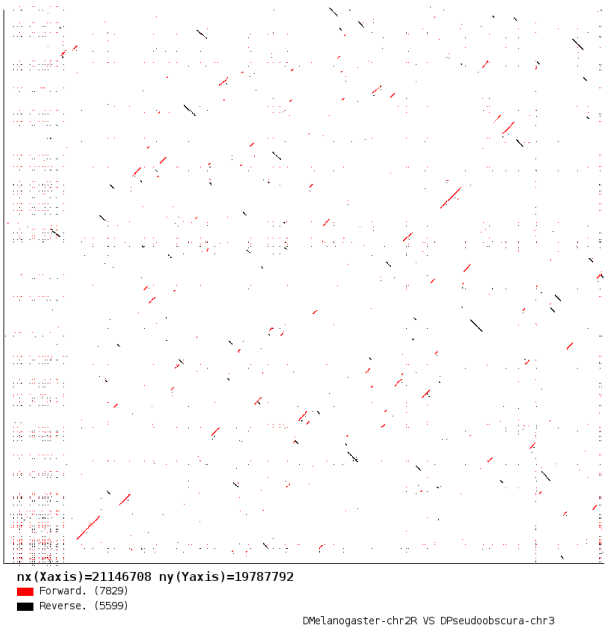

### BuchneraAPS vs. Buchnera5A



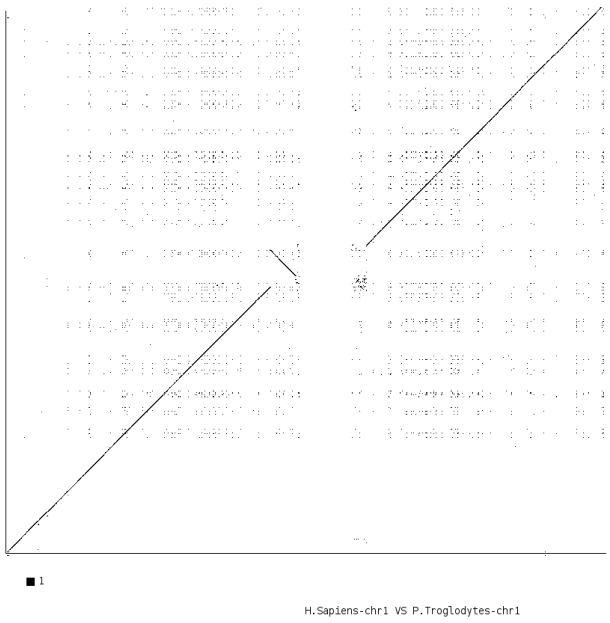

### E.colik12 vs. E.coliO157




### D.Melanogaster-chr2R vs. D.Pseudoobscura-chr3




### H.Sapiens-chr1 vs. P.Troglodytes-chr1




## 7.4 Programs usage

This is a basic usage explanation, a guided exercise is available at `http://chirimoyo.ac.uma.es/gecko/documents/GuidedExercise-fromGENOMES2Visualization-reduced-v0.1.pdf`.

### 7.4.1 Dictionary creation

Usage: dictionary seq.fasta prefixSize prefixOutfile

Parameters:

- Seq.fasta: input sequence from which the words to be stored in the dictionary will be extracted.
- prefixSize: this parameters indicate the number of iterations that the program will do. The number of iterations is $4^p$. Example with $p = 1$ the first iteration will look for the words starting by 'A', the second iteration the words starting with 'C', the third with 'G' and the fourth with 'T'. This is used to avoid entering into starvation due to the high amount of memory to be used.
- prefixOutFile: this parameter indicates the output file name of this program "prefixOutFile.dict"

We have un-successfully used $dustmask$ (from Blast distribution). Thus, we consider low complexity regions (LCR) must be identified before using this procedure.

Dictionary considerations:

1. Accept "ACGT" as valid symbols to conform.
2. The program interprets low case symbols as non-valid characters to conform a word, so they are skipped (this is the usual way used by maskers to represent a LCR).

3. Remove non-coding ASCII characters. We have found, in particular the '\r\n' used by MS-DOS coding to represent the new line are not well processed by Linux-like environments (an alternative is to use the dos2unix command).

Output file format:

```
typedef struct {
  //Word compressed in binary format
  word w;
  //Number of ocurrences inside the
  //sequence. This is used to know the
  //number of locations stored after this
  //struct
  uint64_t num;
} hashentry;

typedef struct {
  //Ocurrence position in the sequence
  uint64_t pos;
  //For multiple sequence files this var
  //reflects in what sequence occurs the
  //word
  uint64_t seq;
} location;
```

structs–dict.h

### 7.4.2 Hits

Usage: hits prefixNameX prefixNameY Outfile Ksize

Parameters:

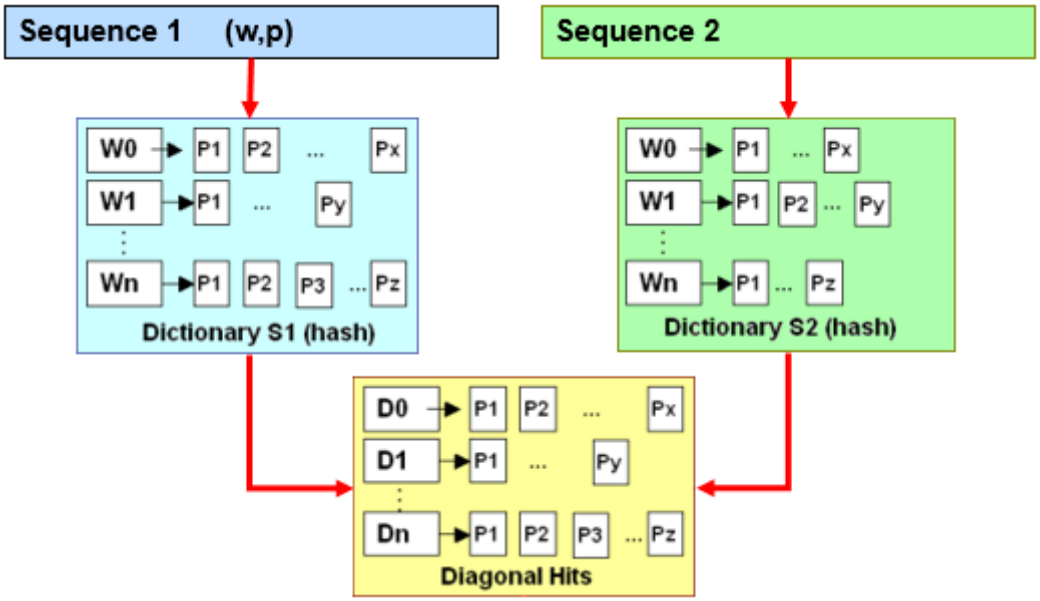
- prefixNameX & Y :refers to *.dict: index of words-InitPosition-NumberRepetitions; and
- Outfile is in the form of [Diagonal][position in X][sequence in X (for multiple sequence fasta files)][position in Y][ sequence in Y (for multiple sequence fasta files)]
- Ksize: prefix matching size (in this way, the dictionary is be comnputed only once for 32-mers but the matches can be computed for smaller values using word prefixes)

Output file format:

```
typedef struct {
  //Diagonal where the hit is located
  //This value is calculated as:
  //posX - posY
  int64_t diag;
  //Ocurrence position in sequence X
  uint64_t posX;
  //Ocurrence position in sequence Y
  uint64_t posY;
```

```
10    //For multiple sequence files this var
11    //reflects in what sequence of X file
12    //occurs the word
13    uint64_t seqX;
14    //For multiple sequence files this var
15    //reflects in what sequence of Y file
16    //occurs the word
17    uint64_t seqY;
18 } hit;
```

structs–hits.h




### 7.4.3 SortHits

Usage: sortHits bufferSize nThreads inputFile Outfile

Parameters:

- bufferSize indicates the number of hits to be stored in memory in order to be sorted. Bigger numbers of this parameter give more performance. Recommended value: 10000000
- nThreads indicates the number of parallel POSIX threads that will sort the hits input file. Our experience executing the program tells us that a good value is 32, because in the meanwhile some threads are doing I/O the others are sorting.
- Outfile is ordered first by [Diagonal] and then by [position in X].

### 7.4.4 FilterHits

Two hits will be grouped if they occur at a given distance (less than the K-mer size parameter). In this way we reduce the number of hits to be extended. It is also possible to remove all

”isolated” hits.

Usage: filterHits fileIn fileOut KmerSize filterIsolatedHits

Parameters:

- fileIn refers to the hits file coming from the sortHits procedure. This file is sorted first by diagonal and then by position in the sequence X.
- fileOut indicates the output file of this program
- KmerSize indicates the length of the matches in order to know when one hit is starting at a position already covered by a previous hit.
- filterIsolatedHits indicates if a hit that is isolated in one of the diagonal is filtered or not. Usually when two sequences are similar it is normal that in every diagonal they have more than one hit.

Note: here there is a possibility to include in the output how many hits conform each Big-Hit and then compute fragments using only Big-Hits formed by ”at least” n simple hits. However, during experiments we verify that for K=32, working with individual hits is fast enough. The main reason for this behaviour is that when hits are ordered, the fragments (next) program is able to collect hits on the same diagonal and then jump all the hits that were incorporated into a given fragment.

The Double-Hit approach used by Blast has been demonstrated to be very powerful to compare short sequences such as genes and proteins. However when working with genomic sequences it is expected large strings of identical fragments (not only by the longer sequences under analysis but other genomic characteristics such as operons, suggest this possibility).


Overlapped hits points for K = 5

Although we have mentioned that the fragment detection program will avoid consecutive hits to be extended, there are some situations (such parallel implementation) in which is better to fusion the consecutive hits to avoid spurious or duplicated results.

Let’s analyze this fact. For fragments with a large identical section several overlapping words will be found (see previous illustration). All these hits are considered as potential seeds or starting point of fragments. Next step is to extend the seed to identify the real fragments. Under this situation (a) either several seeds will be extended to produce the same fragment; or (b) a clever strategy will discard a-posteriori the consecutive seed once the first one integrate all the seed in one fragment. Potentially, in a parallel implementation, this fact will produce a false estimation of the computational load to be distributed to the different processors, and seeds assigned to different processors will be no identified as close hits.

We have analyzed the possibility to extend the seed by using the 2-words matches imported from Blast; or enhancing the seed by let the signal to detect their own boundaries that could be named k-words self-bounded matches.

As it was previously mentioned, the 2-words hits used with high success in program such Blast and FASTA have this good behavior due to the length of the sequences under analysis (genes or proteins). But in the case of genomic sequences, the probability of consecutive words increase so much and distort the estimation of the computational load. There are more seeds than needed since several of these seeds are formed by overlapped words.

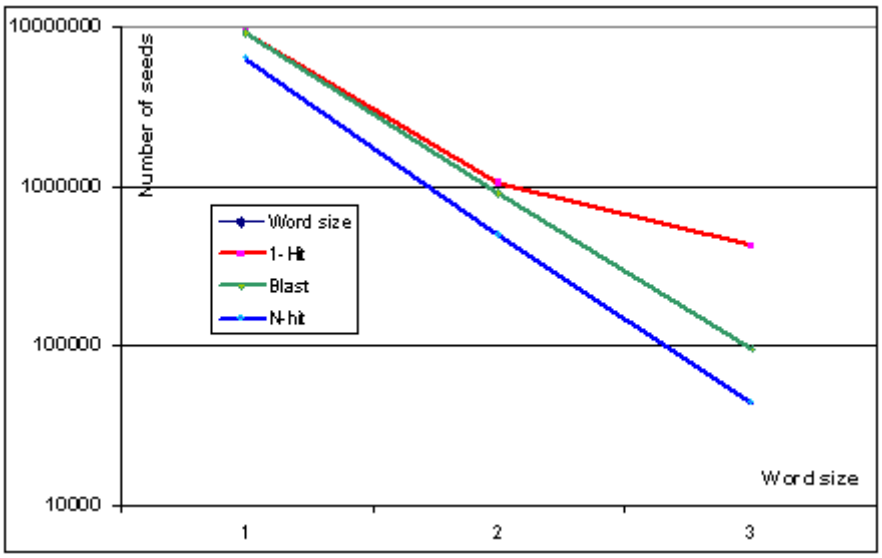
In the table and the corresponding graph, the number of 1-word hits is displayed together with the number of 2-words blast hits used as seed in the well-known application. The self-bounded (N-hits) approach get a notorious reduction of the number of seeds.

| Word size | 8 | 10 | 12 |
|---|---|---|---|
| 1-Hit | 9116866 | 1053657 | 419068 |
| Blast | 9114584 | 893780 | 96933 |
| N-hit | 6314262 | 492683 | 44458 |

Here a (numbers) table with the reduction of the number of seed using both approaches.

**IMPORTANT:** since the 1-word matches are ordered by diagonal, the typical array (of length $N + M - 1$) for identification of 2-words matches is no more necessary.

Thus **NOT memory boundaries!!!**




Number of seeds using the 1-word, 2-words-blast and n-hits (log scale).

Here, the accuracy of results is evaluated. The full set of local alignments between two E.coli genomes was computed ($L1 \& L2 \approx 5Mpb$). The full search space is $L1xL2$, which is reduced to linear space by identifying 1-word hits ($L1 + L2$ search space) and performing 1.035.923 fragment extensions.

Using a word of length 12; a small identical fragment of length 15 will potentially produce 3x2x1 2-hits seeds which represent the factorial of the difference between the fragment length and

the word size, which is stratospheric for words as large as 30.

However, the strategy of self-bounded seeds (Big-Hits) reduces the search space to 30%.

More important, a 99.98% of the final real fragments are identified (99.99% counting fragments with more than 20 residues length).

| Length | number of 1-word hits | n-Hits seeds | Real fragments | Diff. |
|---|---|---|---|---|
| 12 | 1035923 | 722265 | 722283 | 18 |
| 13 | | 218030 | 218058 | 28 |
| 14 | | 65993 | 66011 | 18 |
| 15 | | 20164 | 20183 | 19 |
| 16 | | 6193 | 6209 | 16 |
| 17 | | 1923 | 1937 | 14 |
| 18 | | 605 | 619 | 14 |
| 19 | | 237 | 239 | 2 |
| 20 | | 75 | 75 | 0 |
| 21 | | 58 | 58 | 0 |
| 22 | | 22 | 22 | 0 |
| 23 | | 22 | 22 | 0 |
| 24 | | 17 | 31 | 14 |
| 25 | | 31 | 44 | 13 |
| 26 | | 14 | 14 | 0 |
| 27 | | 14 | | -14 |
| 28 | | 15 | 14 | -1 |
| 29 | | | 14 | 14 |
| 30 | | 18 | 14 | -4 |
| 32 | | 14 | | -14 |
| 33 | | 15 | 15 | 0 |
| 34 | | 2 | 2 | 0 |
| 25 | | | 14 | 14 |
| 36 | | 2 | | -2 |
| 39 | | 14 | | -14 |
| 44 | | 14 | | -14 |
| 46 | | 14 | 14 | 0 |
| 47 | | 14 | | -14 |
| 48 | | 14 | | -14 |
| 49 | | 14 | 14 | 0 |
| 63 | | 14 | 14 | 0 |
| 2031 | | 1 | 1 | 0 |
| 3150 | | 1 | 1 | 0 |
| **TOTAL** | **1035923** | **313564** | **313639** | **75** |
| | | **30,27%** | **99,98%** | |

Table of common words of length 12 for the two E.coli genomes used in the study. This represent the number of seed point that would be processed to identify fragments. A self-bounded strategy to identify longer seeds by overlapping of single hits allow a reduction to 30% of the seeds needed to examine. The extension of seed to identify fragments (local alignments) perfectly reproduce the results produced by exhaustive strategies.

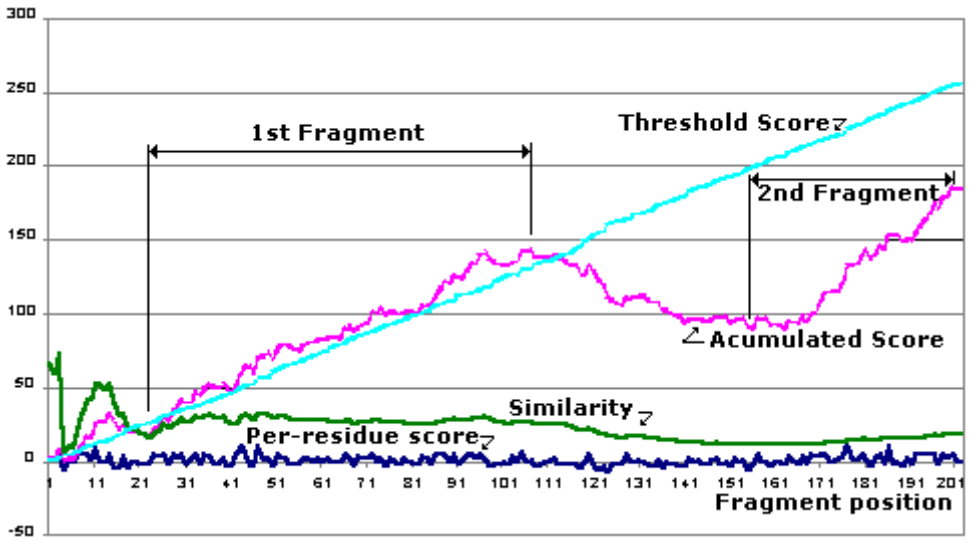
### 7.4.5 FragHits

A fragment is a sub-string present in both sequences whose quality (score) cannot be increased by extending the string in any of both directions. Quality is measured using a scoring scheme. Currently this scoring scheme is 4 for identical symbols, -1 for N-N matches and -4 for different symbols, but we have also available a version which uses a user-specified scoring matrix. The program also accounts the number of identical symbols in the fragment.

It is easy to deduce that a fragment starts and ends with an identical symbol and eventually grows until reach a maximum quality or the sequences ends.

To avoid poorly scored fragments, a filtering strategy is used, based on length or similarity. Thus, when the borders of the fragment are detected, the maximum score is computed in a level of similarity is obtained. In other cases, the p-value is computed for the corresponding fragment.

Noteworthy to observe, the fragment with higher score is not necessarily the best fragment from the similarity point of view, being possible –and really frequent- that a discarded long fragment could contains one or more shorter but well-conserved sub-fragments that satisfy filtering requirements.




Typical fragment detection algorithms that apply quality assessment at the end of fragment detection. In the picture (bottom part) the per-residue score and the similarity level are shown. The growing line represents the accumulated score that should be needed at each point by the fragment to be accepted as an interesting fragment (over the similarity threshold). The hill-shaped line correspond to the real score the fragment obtain at each point. This fragment should be discarded if a similarity threshold is used, because at the end of the fragment although the score is maximum, the similarity does not. Computing at each point the criterion that will be used to filter the fragment should produce two shorter but well-conserved fragments. (A length based threshold score can also be used). Although a slightly increase in CPU-time is needed, better results are obtained.

In this case, hits are used as seed points to extend the fragment. A backward step is also included to extend the fragment in the opposite direction from the hit.

Usage: FragHits SeqX.file SeqY.file HitsFile Out.file Lmin SimThr WordSize fixedLen strand

Parameters:

- SeqX.file and SeqY.file are the sequences in FASTA format.
- HitsFile (binary, ordered and filtered).
- Out file (binary) will save a fragment with the format:

```
1  struct FragFile {
2    //Diagonal where the frag is located
3    //This value is calculated as:
4    //posX - posY
5    int64_t diag;
6    //Start position in sequence X
7    uint64_t xStart;
8    //Start position in Sequence Y
9    uint64_t yStart;
10   //End position in Sequence X
11   uint64_t xEnd;
12   //End position in Sequence Y
13   uint64_t yEnd;
14   //Fragment Length
15   //For ungaped aligment is:
16   //xEnd-xStart+1
17   uint64_t length;
18   //Number of identities in the
19   //fragment
20   uint64_t ident;
21   //Score of the fragment. This
22   //depends on the score matrix
23   //used
24   uint64_t score;
25   //Percentage of similarity. This
26   //is calculated as score/scoreMax
27   //Where score max is the maximum
28   //score possible
29   float similarity;
30   //sequence number in the 'X' file
31   uint64_t seqX;
32   //sequence number in the 'Y' file
33   uint64_t seqY;
34   //synteny block id
35   int64_t block;
36   //'f' for the forward strain and 'r' for the reverse
37   char strand;
38 };
```

structs–frags.h

- Lmin: minimal fragment length.
- SimThr: Similarity Threshold (to obtain all fragments use a low value, or zero).
- WordSize indicates the K value used to calculate the hits (seed points) based on the computed sequence dictionaries.
- FixedLen indicates whether the length parameter should be taken into account as the actual value or as the percentage of the length of the input sequences. This parameter has more sense when comparing multiple sequence fasta files. Allowed values: 1 (actual value), 0 (to be considered as percentage).
- Strand indicates whether we are computing the forward or the reverse strand fragments. Allowed values: f (forward), r (reverse).

## 7.5 Results quality

Although the performance aspects of GECKO's design are crucial, the production of high quality results is equally important. In this section we explain how we evaluated the quality of the results produced by our algorithm versus the other applications using the same parameters. The rationale behind our evaluation was to compare the coverage of the HSPs detected by each algorithm. To avoid biases in the evaluation we decide to obtain a consensus set of reference HSPs. This set is composed of those HSPs reported by at least half of the reference algorithms. The HSPs produced by GECKO were then mapped over the reference HSPs and the percentage of coverage recorded as a measurement of result quality. This means that matching positions reported by the consensus HSP reference and not reported by GECKO will push down the quality and vice versa. There are more sophisticated ways of comparing the results, such as only considering coding regions, or by qualifying and weighting matches depending on sequence type or section. However, we decided not to use these methods as they can incorporate noise or biases into the evaluation.

Following the previously described procedure with the results of the pairwise test described in the paper, the evaluation determined that in our experiments GECKO detected 3% more HSPs on average than the consensus set. Moreover, GECKO obtained a larger dataset while maintaining identity values over 65%, thus representing the identification of additional statistically-significant HSPs.

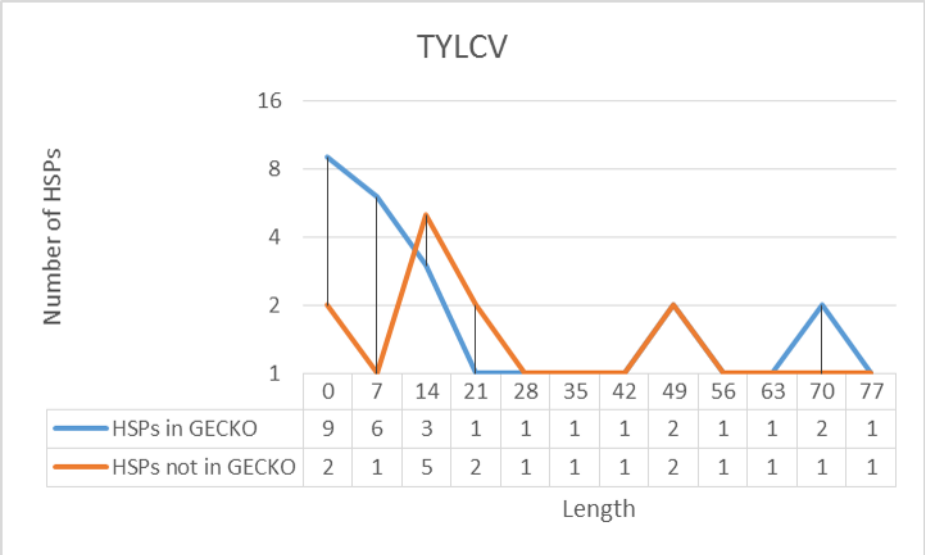
In the following sections we will summarize some examples of the performed experiments showing the HSPs we report and the other don't and vice versa. The graphs only pretend to illustrate how we are comparing. We will show also the alignment of those we are not taking into account to point out the alignment is not good enough and the same for the alignments not reported by the others to point out that they are good enough.

**TLCV vs. TYLCV-Ir2**

As explained before the result of the comparison based on coverage is the following:

| | |
|---|---|
| Gecko cov. | 87.45% |
| Consensus cov. | 83.17% |
| Diference | 4.28% |

The following plots have as X-axis the length of the zones in which HSPs reported by GECKO differ with the ones reported by the consensus set, and the other way around. The maximum X value corresponds to the longer difference. Then we group the values from 0 to the maximum value in 12 classes which are the one represented in the axis. The Y-axis contains the number of segments with the given difference in length. From the plot shown below we can extract that the coverage difference contributing to GECKO's better result are concentrated in the zone of 70 residues and also in short segments. Given the fact of the short sequence length, and also that the HSPs are non-overlapping, we believe the previously mentioned segments are the ones contributing to the better coverage.



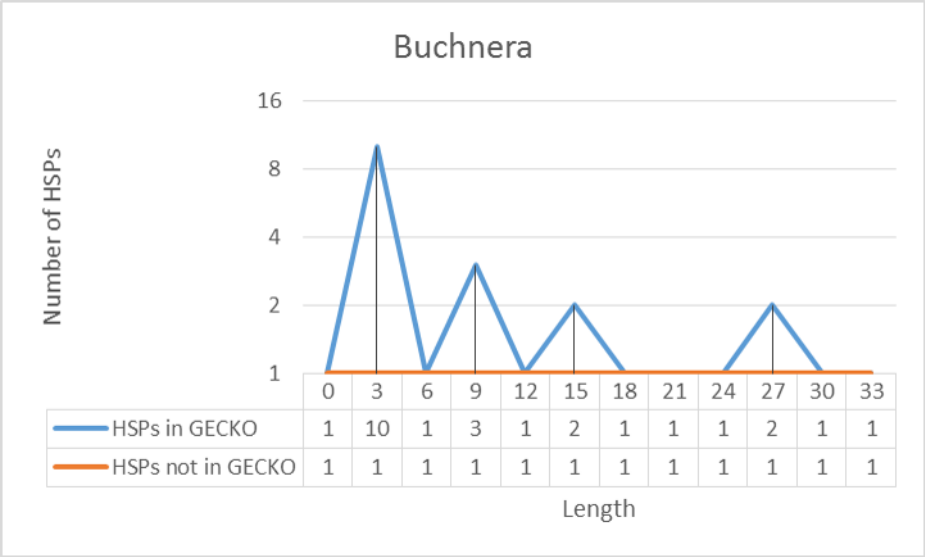

| | 0 | 7 | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 | 70 | 77 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HSPs in GECKO | 9 | 6 | 3 | 1 | 1 | 1 | 1 | 2 | 1 | 1 | 2 | 1 |
| HSPs not in GECKO | 2 | 1 | 5 | 2 | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 1 |

**Buchnera APS vs. Buchnera 5A**

As explained before the result of the comparison based on coverage is the following:

| | |
|---|---|
| Gecko cov. | 100.00% |
| Consensus cov. | 99.99% |
| Diference | 0.01% |

In this comparison the difference in coverage is minimal so the plot of the HSPs present in GECKO and not in the others is not so significant since the lengths in the X axis are very short.

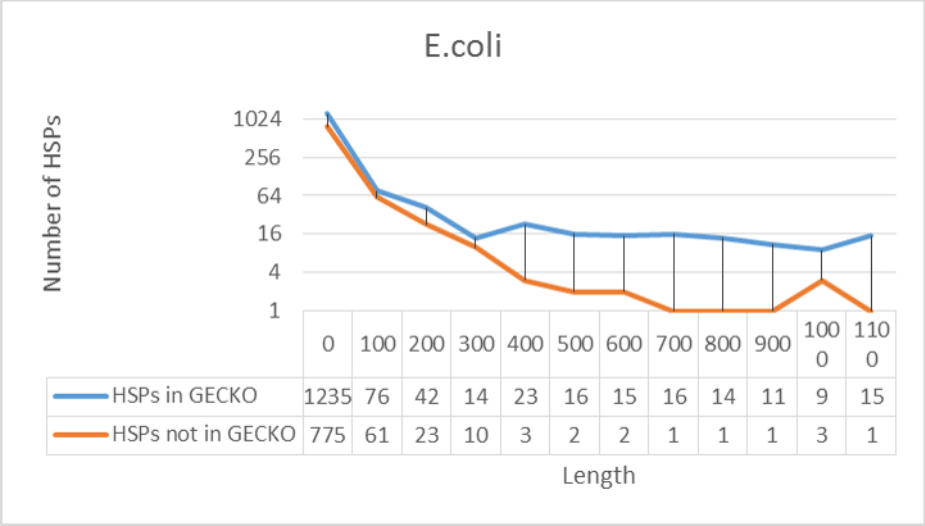



**E.colik12 vs. E.coliO157**

As explained before the result of the comparison based on coverage is the following:

| | |
|---|---|
| Gecko cov. | 90.19% |
| Consensus cov. | 88.93% |
| Diference | 1.26% |

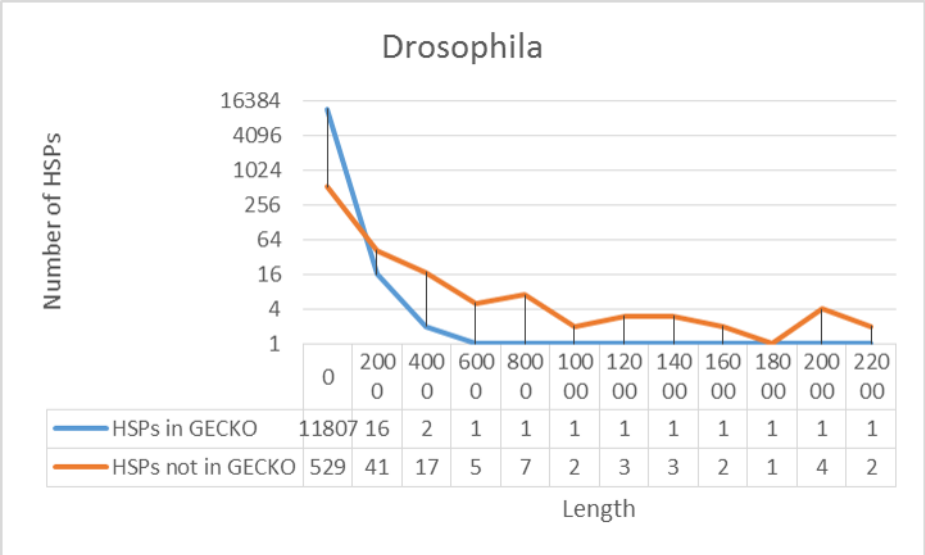
The frequencies of the HSPs of different lengths found in one party and not in the other can be visualized in the next figure (Y axis in logarithmic scale).




**D. Melanogaster chr2R vs. D. Pseudoobscura chr3**

In this comparison with larger sequences the state of the art software is joining small HSPs to conform long alignments. This is the explanation of why GECKO is reporting more segments only in the first range of the plot (0-1999) and the rest software has longer segments.

| | |
|---|---|
| Gecko cov. | 90.19% |
| Consensus cov. | 88.93% |
| Diference | 1.26% |




| | 0 | 2000 | 4000 | 6000 | 8000 | 10000 | 12000 | 14000 | 16000 | 18000 | 20000 | 22000 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HSPs in GECKO | 11807 | 16 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| HSPs not in GECKO | 529 | 41 | 17 | 5 | 7 | 2 | 3 | 3 | 2 | 1 | 4 | 2 |

## References

[1] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[2] Jose Arjona-Medina, Oscar Torreno Tirado, and Oswaldo Trelles. Software for featuring genome evolution, 2014. Poster presented at the European Conference on Computational Biology (ECCB), Sept 7–10, Strasbourg conference center, Strasbourg, France.

[3] Mr. SymBioMath Consortium. Mr. SymBioMath Annex I: Description of Work. Grant Agreement Number 324554, FP7-PEOPLE-2012-IAPP, October 2012.

[4] Mr. SymBioMath Consortium. Big data I/O management algorithms. Mr. SymBioMath Deliverable D2.1, 2014. http://www.mrsymbiomath.eu/workpackages/wp2.

[5] Mr. SymBioMath Consortium. Cloud and hpc-based software. Mr. SymBioMath Deliverable D2.4, 2014. http://www.mrsymbiomath.eu/workpackages/wp2.

[6] Aaron E Darling, Bob Mau, and Nicole T Perna. progressivemauve: multiple genome alignment with gene gain, loss and rearrangement. *PloS one*, 5(6):e11147, 2010.

[7] Martin C Frith, Michiaki Hamada, and Paul Horton. Parameters for accurate genome alignment. *BMC bioinformatics*, 11(1):80, 2010.

[8] Martin C Frith, Raymond Wan, and Paul Horton. Incorporating sequence quality data into alignment improves dna read mapping. *Nucleic acids research*, 38(7):e100–e100, 2010.

[9] RS Harris. Improved pairwise alignment of genomic dna. 2007. *PhD diss., The Pennsylvania State University*, 2007.

[10] Yue Huang and Ling Zhang. Rapid and sensitive dot-matrix methods for genome analysis. *Bioinformatics*, 20(4):460–466, 2004.

[11] Szymon M Kiełbasa, Raymond Wan, Kengo Sato, Paul Horton, and Martin C Frith. Adaptive seeds tame genomic sequence comparison. *Genome research*, 21(3):487–493, 2011.

[12] Jan Krumsiek, Roland Arnold, and Thomas Rattei. Gepard: a rapid and sensitive tool for creating dotplots on genome scale. *Bioinformatics*, 23(8):1026–1028, 2007.

[13] Vamsi K Kundeti, Sanguthevar Rajasekaran, Hieu Dinh, Matthew Vaughn, and Vishal Thapar. Efficient parallel and out of core algorithms for constructing large bi-directed de bruijn graphs. *BMC bioinformatics*, 11(1):560, 2010.

[14] Stefan Kurtz, Adam Phillippy, Arthur L Delcher, Michael Smoot, Martin Shumway, Corina Antonescu, and Steven L Salzberg. Versatile and open software for comparing large genomes. *Genome biology*, 5(2):R12, 2004.

[15] Iris Leitner and Oswaldo Trelles. Intuitive library for efficient access of compressed genome sequences. In *Poster Proceedings of the 13th European Conference on Computational Biology*, 2014.

[16] Felipe A Louza, Guilherme P Telles, and Cristina Dutra De Aguiar Ciferri. External memory generalized suffix and lcp arrays construction. In *Combinatorial Pattern Matching*, pages 201–210. Springer, 2013.

[17] Francielle Maboni, Ana Tereza Ribeiro de Vasconcellos, Arnaldo Zaha, Alex Upton, Priscill Orue Esquivel, Oscar Torreno, and Oswaldo Trelles. Pig metagenome analysis using gecko. Technical Report 2014-001, Department of Computer Architecture, University of Málaga, Campus de Teatinos, Málaga, September 2014.

[18] Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. *siam Journal on Computing*, 22(5):935–948, 1993.

[19] Victoria Martín-Requena, Javier Ríos, Maximiliano García, Sergio Ramírez, and Oswaldo Trelles. jORCA: easily integrating bioinformatics web services. *Bioinformatics*, 26(4):553–559, 2010.

[20] Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

[21] William R Pearson and David J Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8):2444–2448, 1988.

[22] Temple F Smith and Michael S Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[23] Jeffrey Scott Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing surveys (CsUR)*, 33(2):209–271, 2001.