Fast alignment of large genome databases: A Demonstration [*]

Tamer Kahveci Ambuj K. Singh
Department of Computer Science University of California
Santa Barbara, CA 93106
{tamer,ambuj}@cs.ucsb.edu

1 Motivation

The growth in the amount of genomic information has spurred increased interest in large scale comparison of genetic strings. Identification of the genetic code of the deadly E.coli bacteria, or genetic clues for fibrodysplasia ossificans progressiva (FOP), a disease that affects muscle and skeleton growth, and the vital proteins for the bone growth, or identification of the genes that hasten the healing of some venous ulcers are only a few of the achievements obtained recently. Conditions such as skin and colon cancer can already be classified into much finer categories than before and soon the same approach may be possible for heart disease, schizophrenia and many other conditions. Using this kind of genetic information helps to target the treatments at the precise form of the illness that has been diagnosed, thus helping patients and doctors weigh the risk and benefits of different treatments.

One important emerging application, called *Comparative Genomics*, analyzes and compares the genetic material of different species. It is the most reliable way to identify genes and predict their functions. The functions of the higher level organisms, like humans, can be revealed by comparing to their counterparts in similar or lower level organisms. Such genome analysis involve comparison of huge strings, as large as the whole genome of a species.

Phylogenetics and evolutionary studies are other important applications that use complete genetic information of different species. Phylogenetics is used to infer the ancestral relationships among different species. This sort of relations can be captured by comparing the genetic code of the whole genomes.

The amount of biological data has been growing exponentially. This growth is on a collision course with current homology search and database query techniques and presents new challenges to biological database design. Queries (as mentioned above) will be large and complex, databases will be huge, some data will be on disk, and significant portions of datasets will be local because of networking bottleneck and proprietary data. For example, human chromosome 21, one of the shortest human chromosomes, contains more than 33 M base pairs. Fast and sensitive homology search algorithms are needed to 1) answer large queries, 2) handle huge databases stored on disk, and 3) interact with multiple datasets seamlessly.

We demonstrate an efficient algorithm for alignment of large genome strings. Our algorithm constructs a boolean *Match Table* for a given query string and database string with the help of the MRS index structure [2]. The size of the MRS index structure is approximately 1-2% of that of database. Each entry of the Match Table corresponds to a query/database substring pair. An entry in the Match Table is marked as *True* if the corresponding query substring and database substring potentially contain similar patterns. It is marked as *False* otherwise. The size of the Match Table is negligible compared to that of database (typically 0.1% of the database.).

Once the Match Table is computed, we build hash tables on these strings as follows. First, we find the number of marked rows and columns by projecting all the *True* entries of the Match Table to its rows and columns. If the number of marked columns is more than the number of marked rows, we choose a vertical slice of the Match Table, and construct the hash table on the string that corresponds to rows of the Match Table. Otherwise, we choose a horizontal slice, and construct the hash table on the string that corresponds to columns of the Match

[*]Work supported partially by NSF under grants EIA-0080134, IIS-9877142, and DBI-0213903.

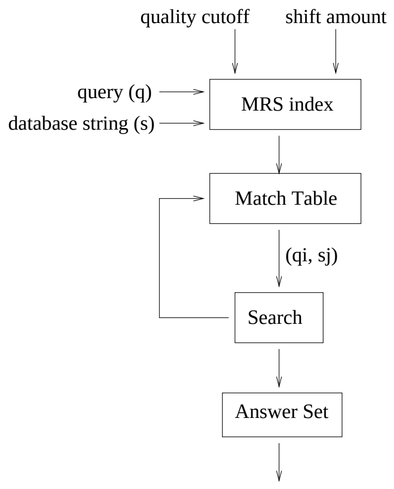

Figure 1. Control flow within MAP. q_i and s_j correspond to query and database substrings respectively.

Table. The width of these slices is restricted by available memory: the size of the hash table for a slice is no more than the available memory.

Once the hash table of a string for a slice is constructed, the marked substrings of the other string are read sequentially and exactly matching substrings (i.e. seeds) of the prespecified size (i.e. 11) are found using this hash table. The seeds are then extended in both directions to find better matches. Later, the results are reported in a descending score order. We call this technique *MAP* (MAtch table based Pruning).

The experimental results show that, MAP runs up to 97 times faster than BLAST [1] without decreasing the output quality. Furthermore, MAP can work well even for small memory sizes. This drastic reduction in CPU and I/O cost has the potential of making homology searches viable on desktop PCs. The filtering and scheduling techniques of MAP can easily be used to speedup and reduce the memory requirements of any of the current genome alignment tools. MAP also provides the user a coarse grain visualization of the similarity pattern between the strings prior to actual search. Currently, we are building a web server which provides genome alignment using MAP.

2 System Overview

Figure 1 illustrates the different steps of this technique. The user submits a query along with two search parameters: quality cutoff and shift amount. These parameters affect the sensitivity and the performance of search. A boolean *Match Table* is then constructed by using the MRS index structure on the datasets. Later, pairs of blocks of the datasets are read using this Match Table and searched using BLAST.

As shown in Figure 2(a), the Match Table M for a query string q and a database string s, is a boolean matrix. It is constructed by performing a search on the MRS index structure. Each column of the match table corresponds to the substrings contained in an MBR of the MRS index structure. For simplicity, we set the box capacity to the page size. A number of subqueries are constructed by sliding a window of length w on query q with a shift amount of Δ. Each row of M corresponds to a set of consecutive subqueries of total size equal to a page size. A score value is computed for each database/query block pair using the MBR approximation of the sequences in the MRS index structure. If the score value for a pair is larger than the given threshold, then the corresponding entry in the Match Table is set to *True*. It is set to *False* otherwise.

Once the match table is constructed, only the query/database page pairs that are marked as *True* need to be searched. Figure 2 illustrates the slices obtained from a sample Match Table by MAP. In this figure, we assume that the available memory can store the hash table for 3 pages at most.

The decision for split direction is made as follows. Let r and c be the number of marked rows and columns of the match table. If $r < c$ then the Match Table is split vertically. Otherwise, the Match Table is split horizontally. A hash table is then constructed on the marked rows if it is a vertical split. Otherwise, the hash table is built on the marked columns of the split region. The other dataset is sequentially scanned and the same procedure is run for the rest of the Match Table. For example, for the Match table in Figure 2(a), $r = 5$ and $c = 9$. Therefore, it is split vertically and the hash table is built on the rows of this cut. The size of a slice is determined based on memory size: The size of the hash table for the marked substrings of slice can not be more than the available memory. This is determined by iteratively incrementing the split location.

In our experiments, we used the BLAST [1] for comparison. According to our experimental results, MAP runs 20 to 97 times faster than BLAST. Furthermore, MAP can work well even for small memory sizes. Unlike the massive data structures used by current techniques, the size of the MRS index structure is only 1-2% of the database.

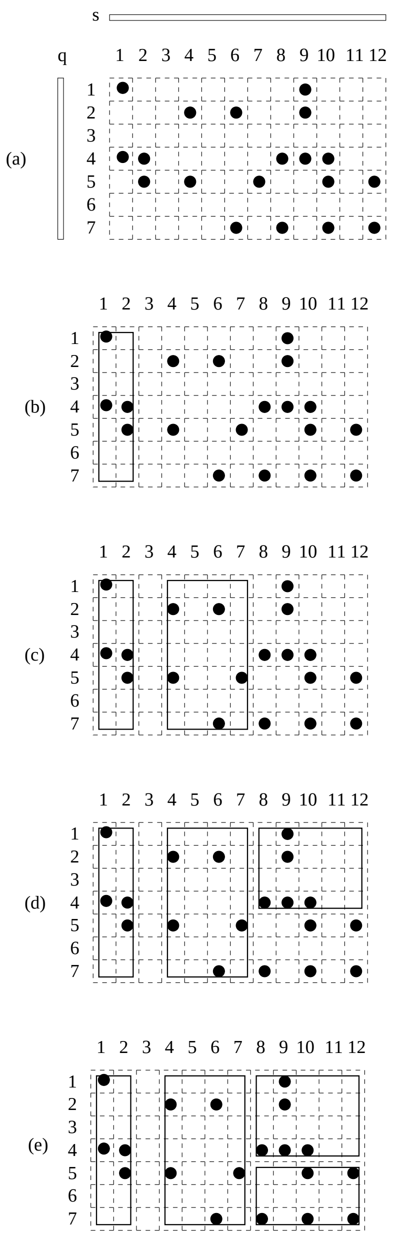

Figure 2. (a) An illustration of Match Table on query q and database s. The black dots correspond to *True* entries. (b-e) The slices determined by the MAP algorithm. we assume that the available memory can store the hash table for 3 pages at most.

MAP achieves these performance improvements while keeping quality of the resulting answer set very close to that of BLAST. Detailed experimental evaluation of MAP is available in [3].

MAP is a very general technique, in the sense that its Match Table based pruning and dynamic splitting scheme can be used to improve any of the current string search tools. Hence, one can view MAP as a technique that improves the available techniques instead of as a competitor to these techniques.

References

[1] S. Altschul and W. Gish. Basic local alignment search tool. *J. Molecular Biology*, 1990.

[2] T. Kahveci and A. Singh. An efficient index structure for string databases. In *VLDB*, pages 351–360, Roma, Italy, September 2001.

[3] T. Kahveci and A. Singh. MAP: Searching large genome databases. In *PSB*, 2003. to appear.