q-gram Based Database Searching Using a Suffix Array (QUASAR)

Stefan Burkhardt [*] Andreas Crauser [*] Paolo Ferragina [†]
Hans-Peter Lenhof [*] Eric Rivals [‡] Martin Vingron [‡]

Abstract

With the increasing amount of DNA sequence information deposited in public databases, searching for similarity to a query sequence has become a basic operation in molecular biology. But even today's fast algorithms reach their limits when applied to all-versus-all comparisons of large databases. Here we present a new database searching algorithm called QUASAR (Q-gram Alignment based on Suffix ARrays) which was designed to quickly detect sequences with strong similarity to the query in a context where many searches are conducted on one database. Our algorithm applies a modification of q-tuple filtering implemented on top of a suffix array. Two versions were developed, one for a RAM resident suffix array and one for access to the suffix array on disk. We compared our implementation with BLAST and found that our approach is an order of magnitude faster. It is, however, restricted to the search for strongly similar DNA sequences as is typically required, e.g., in the context of clustering expressed sequence tags (ESTs).

1 Introduction

Numerous and large databases holding DNA and protein sequences are now readily available over the WEB and are quickly becoming the "lifeblood of molecular biology" [Wal95]. This is due to the combination of the power of biomolecular *sequence comparison* with the ease-of-use of tools for searching such databases for sequences exhibiting *similarity* to a given query sequence. These searches are nowadays routine and vital for molecular biology because they serve to "generate new knowledge" [Doo90]. Whenever a new gene is cloned and sequenced, visiting the appropriate databases is the next step. Different, but equally interesting, applications of these databases are the clustering of similar sequences into sequence families [KV98], and the assembly of sequence fragments in genome sequencing [VAS$^+$98].

Fast programs like the BLAST and FASTA packages [AMS$^+$97, PL88] or iterated Smith-Waterman [SW81] sequence alignments are used for this purpose. Especially BLAST is impressively fast. This program performs a linear scan of the whole collection of sequences searching for a set of words belonging to the neighborhood of some substrings of the query string. The result is a list of candidate hits in the database. Although this algorithm performs single database searches at an amazing speed, today's applications tend to introduce more stringent performance requirements where these algorithms reach their limits. Be it in the comparison of an EST database to itself for the purpose of clustering or in the context of shotgun sequencing, all-against-all comparisons of large amounts of data need to be computed. The problem is further augmented by the current exponential growth in primary sequence data. Thus, database searching tools that read through the entire database may in the near future become too slow to cope with the avalanche of data (see also [Mye92]).

In the field of *exact* string matching, techniques have been developed that preprocess a text in such a way that, upon searching a pattern, only small parts of that text actually need to be explicitly accessed. Since in the applications described above, one imposes a very stringent match criterion, the hope is to draw on these techniques in order to further improve the efficiency of database searching algorithms. Candidates are sophisticated indexing data structures like suffix trees [McC76], suffix arrays [MM93], or Patricia trees [Knu73], that allow to perform queries in time proportional to the length of the query string while being as independent as possible of the size of the searched text [McC76].

Only few attempts have been made to adapt these techniques to the similarity searches needed for biological purposes. Martinez [Mar83], for example, gave the first application of a position tree in molecular biology. This data structure requires about 16 times the space needed to store the original data which clearly may create serious problems when applied to large data collections (see also [Heu97, MH95]). The first sublinear expected time algorithms were proposed by Chang and Lawler [CL94] and Ukkonen [Ukk92]. They use a suffix tree of the query sequence but still scan the database.

Myers [Mye94] suggested a sublinear (in the database size) search algorithm that is centered around an index built on the database sequences. The query sequence is split into small substrings and words from a neighborhood of these are located in the index. The IBM product FLASH [CR93] takes advantage of a large "probabilistic" index where not

[*] MPI für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany
[†] Dipartimento di Informatica, Università di Pisa, Corso Italia 40, 56125 Pisa, Italy
[‡] Deutsches Krebsforschungszentrum, Abt. Theoretische Bioinformatik , INF 280, D-69120 Heidelberg, Germany

RECOMB '99 Lyon France
Copyright ACM 1999 1-58113-069-4/99/04...$5.00

all q-tuples are considered, but only some of them are randomly chosen. They report a 18 GB index for a 100 million residue database which makes such an approach impractical for large databases.

In this paper we restrict ourselves to the search for sequences that are strongly similar to a given query sequence. A typical scenario is searching an EST database for sequences that are derived from the same gene as the query sequence. The degree of similarity expected here is high because in essence, one only needs to deal with sequencing errors. Furthermore, we have in mind an application where many searches are run together like, e.g., in an all-versus-all comparison of a set of sequences. Thus, we aim at a search algorithm that is *very fast* on each single search at the expense of (possibly) diminished sensitivity. Given the low performance of current disks compared to internal memory (regarding access time and transfer rate), QUASAR is based on an index data structure to avoid the linear scan of the entire database.

Following a standard approach (see e.g. [JU91, Ukk92, HBD94, PW93, PL88]), we reformulate the problem of searching for database sequences which are very close to the query sequence to the problem of performing a *number of exact* searches for short subsequences of length q (called q-grams). Our approach is based on the following observation: *if two sequences have an edit distance below a certain bound, one can guarantee that they share a certain number of q-grams* [JU91, Ukk92]. This observation allows us to design a filter that selects candidate positions from the database where the query sequence possibly occurs with a high level of similarity. These positions can later be inspected in more depth with a standard alignment algorithm.

The crucial point in the approach we use for QUASAR was therefore the design of the filter which combines some already known ideas in a new way. We *logically* partition the database into equal length blocks of size, say, b (cf. [WM92]). The search of the query sequence S is then decomposed into a certain number of "similarity" searches for subsequences of S of *fixed* length. Such a search is implemented by searching for all the occurrences of its q-grams in the database, counting for each block the number of searched q-grams occuring in it. At this point, a properly chosen bound on the number of shared q-grams allows us to detect blocks which possibly correspond to database portions highly similar to the query sequence S (i.e., candidate hits). An array of counters—one per block—is used to speed up the counting process and a suffix array [MM93] is applied as a space-efficient data structure for quickly retrieving the positions where the searched q-grams occur in the database. We implemented two versions of our data structure and algorithm: one assuming that the whole data collection is resident in the internal memory of the computer, and the other assuming that the index data structure is large and thus has to reside on disk.

The results we achieve with this approach show that our match criterion and filtering approach are successful, especially because our focus is on near-perfect matching for many queries. Our evaluations will focus on such situations showing that our new algorithm is more than one magnitude faster than BLAST while maintaining about the same sensitivity. In the case of an all-against-all comparison, where the running time depends quadratically on the number of sequences compared, such an improvement leads to a substantial increase in overall performance. Similarly, one can envisage that a heavily used web service might take advantage of the speed offered by our approach by collecting queries for a few seconds and then searching for them in the database.

The paper is organized as follows. The Algorithm Section starts by introducing the match criterion we want all our candidate hits to fulfill and explains how it is applied as a filter to select blocks for inspection. It introduces the structure of QUASAR, shows how the suffix array was combined with q-gram matching, and describes the implementation. The Results Section contains the experimental analysis of the internal memory and the secondary memory version of our algorithm. It will elaborate on running time, sensitivity, and also give details on the efficiency of the match criterion we use as a filter.

2 The Algorithm

We have developed and implemented an approximate matching algorithm for determining all sequences in a database D that have a local similarity to a query sequence S. We say that a sequence $d \in D$ is locally similar to S, if there exists at least one pair $(S[i:i+w-1], d')$ of substrings with the following properties:

- $S[i:i+w-1]$ is a substring of S of length w and d' is a substring of d.
- The substrings d' and $S[i:i+w-1]$ have edit distance at most k, i.e., d' can be transformed into $S[i:i+w-1]$ by at most k insert, delete and replace operations.

We call this the *approximate matching problem with k differences and window length w*. For simplicity, we assume that D is one single string of length $|D|$. A pair of substrings with the above properties is called an *approximate match*. Starting with $S[1:w]$ we perform the approximate matching calculations for all possible substrings of S of length w.

For simplicity we now consider the first substring $S[1:w]$ of length w and describe how to determine all approximate matches of $S[1:w]$ with substrings of D. Our general idea is the following: We modify an idea introduced by [OM88] and [JU91] to solve approximate matching by reducing it to exact matching of short substrings of length q (called *q-grams*). It relies on the following lemma:

Lemma 1 [JU91] Let an occurrence of $S[1:w]$ with at most k differences end at position j in D. Then at least $w+1-(k+1)q$ of the q-grams in $S[1:w]$ occur in the substring $D[j-w+1:j]$.

This lemma gives a necessary condition for a subsequence of D to be a candidate for an approximate match with $S[1:w]$: At least $t = w+1-(k+1)q$ of the q-grams contained in $S[1:w]$ occur in a substring of D with length w. Substrings of D with this property are potential approximate matches and will later be checked with an alignment algorithm.

2.1 Suffix Array as Index Data Structure

In order to find the potential approximate matches of $S[1:w]$ in D, for each q-gram Q in $S[1:w]$ we have to efficiently retrieve its list of occurrences in D (we call this a *hitlist*). By using an index data structure for all q-grams in D we hope to direct the search for Q towards small portions of D and thus to avoid scanning the whole database. Since q is a parameter in our approach, we decided to use a full-text indexing data structure so that it is not necessary to rebuild the index if we change q. We use the suffix array as introduced by Manber and Myers [MM93]. The suffix array A

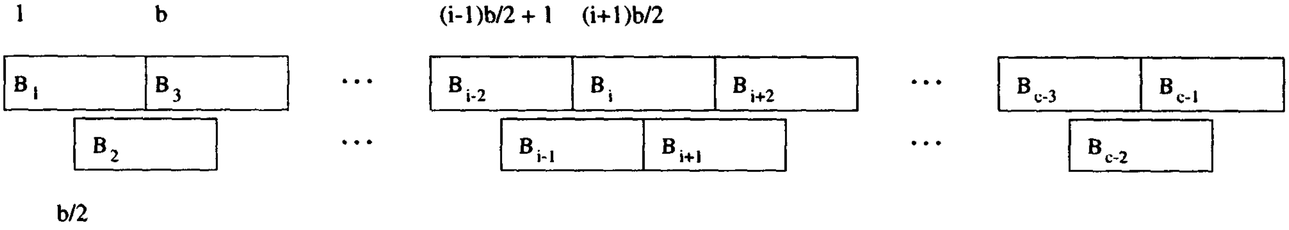

Figure 1: Partition of the database D into overlapping blocks of size b.

built on database D is an array of length $|D|$ storing the lexicographically ordered sequence of all suffixes of D. Entry $A[j]$ contains the text position where the j-th smallest suffix of D starts. Therefore A requires storing exactly one pointer per text position (usually 4 bytes). The suffix array for D is constructed in a preprocessing step. As we are only interested in the occurrences of q-grams, it is not necessary to use the search procedure introduced by Manber and Myers. Instead, we precompute the positions of the hitlists in the suffix array A for all possible q-grams and store them in an auxiliary search array of size $|\Sigma|^q$, where Σ is the alphabet. For a given query q-gram, this allows us to find the start position of its hitlist in constant time. If we want to change q, we only have to precompute the start positions again via a sequential scan of A.

2.2 Block Addressing

In order to find all approximate matches between $S[1:w]$ and D, we have to identify all the substrings in D that share at least t q-grams with $S[1:w]$. A simple approach would be to assign a counter to each substring of length w in D ($|D| - w + 1$ counters) and to increment all the counters of blocks containing q-grams from $S[1:w]$. After processing the hitlists of the query q-grams all substrings with counter values greater than or equal to t are potential approximate matches. A main drawback of this solution is the space required to store the counters. Therefore we combine several substrings of length w into a block and assign only one counter to this block [JU91, WM92]. This strategy has two effects: it reduces the amount of counters required, but it can also lead to more false positives. It therefore introduces a tradeoff between space requirements and the number of falsely identified candidates.

In detail our *block addressing scheme* works as follows: The database D is conceptually divided into blocks of fixed size b ($b \geq 2w$). We assign a counter to each block. This counter will be incremented whenever a search for a q-gram Q reports an occurrence inside the block. After processing all q-grams in $S[1:w]$, the counter of a certain block indicates how many q-grams from $S[1:w]$ are contained in this segment of the database. These counter values are stored in an array of size $|D|/b$. Using this array we can find all interesting portions of the database, i.e., all blocks that contain t or more occurrences of q-grams from $S[1:w]$ (see Lemma 1). These blocks have to be checked for approximate matches using an alignment algorithm.

If we look in more detail at the simple block addressing scheme, we see that we will miss candidates for approximate matches that cross block boundaries. In a worst case scenario, the occurrences of q-grams from $S[1:w]$ are spread among two adjacent blocks and none of these block counters reaches the threshold t. In order to avoid this problem, we use a second block decomposition of the database, i.e., a second block array. The second block decomposition is shifted by half the length of a block (b/2) (see Figure 1). It is obvious that, if a situation as described above occurs for blocks B_1 and B_3 in Figure 1, then block B_2 contains the potential candidate.

At the end of the search procedure for the q-grams of $S[1:w]$ the blocks containing approximate matches to $S[1:w]$ have a counter value of at least t.

2.3 Window Shifting and Alignment

So far we have discussed our approach to find all approximate matches for the window $S[1:w]$. In order to determine the approximate matches for the next window $S[2:w+1]$, we only have to consider the "old" q-gram $S[1:q]$ and the "new" q-gram $S[w-q+2:w+1]$. We have to reconsider the hitlist of the q-gram $S[1:q]$. We decrement the counter values of all blocks that contain copies of this q-gram and that have not reached the threshold t. If a counter for a block has already reached t, we leave it unmodified. In this way we mark all candidate blocks already found. Then we use the suffix array to search for all occurrences of the "new" q-gram $S[w-q+2:w+1]$ and increment the corresponding block counters. We shift the window of length w over the string S until we reach its end.

After computing the list of blocks containing potential hits, BLAST [AGM$^+$90] is used to scan all these blocks. The version we integrated into our code is a modification of NCBI BLAST 2.0.3. A database in BLAST format is built in main memory which is then passed to the BLAST search engine. The construction of this database requires a significant amount of time and introduces unnecessary overhead.

2.4 Secondary Memory Version

The size of larger databases may prohibit storing the suffix array in internal memory. Therefore we developed a *secondary memory* version of QUASAR where the suffix array is stored in secondary memory (hard disk). As hard disks are mechanical devices, the time to access data is dominated by moving the disk head to the location where the data resides. This seek time is not a linear function of distance, i.e., seeks over short distances are much faster than seeks over long distances [RW94]. Additionally, modern disk drives are optimized to be fast on sequential operations. This is achieved by read-ahead strategies and command-buffering. Therefore it is necessary to avoid random disk accesses whenever possible. The approach described above would introduce random disk accesses to read the hitlists of the q-grams out of the suffix array. Our simple solution to circumvent this problem is as follows. We group together several query sequences. Then we generate a list of all q-grams contained in these sequences and sort them with respect to the start positions of their hitlists in the suffix array. This allows us to access the suffix array sequentially, reducing the number of random disk accesses and taking advantage of read-ahead strategies.

2.5 Complexity

The preprocessing-step (the construction of the suffix array and the precomputation of the search array) can be done in $O(|D| \log |D|)$ time [MM93]. Searching for a specific q-gram requires constant time but the number of reported occurrences can be linear in $|D|$. As there are $O(|S|)$ q-grams, our approach takes $O(|S| \cdot |D|)$ time. If at the end c blocks reach the threshold t, the alignment with BLAST takes further $O(c \cdot b \cdot |S|)$ time. The space complexity of our algorithm is dominated by the space used for the suffix array. At construction time we need $9|D|$ space. At run time our internal algorithm only consumes $5|D|$ space.

3 Results

We evaluated the performance and sensitivity of QUASAR and compared it to NCBI BLAST 2.0.3. In the following, we describe the setup of our experiments, compare the lists of similar sequences found by BLAST and QUASAR, and analyze the running times and the efficiency of our filter. As QUASAR was designed for processing multiple queries, we used test sets of 1000 queries (ESTs from the databases themselves). When evaluating BLAST we also used 1000 queries at a time. This allows BLAST to keep the database in main memory, so it only has to load it once at the start of the test run. The time for loading was included in our calculation for the average time per search because we could not measure it separately, but we estimate its impact on the measurement to be far below 1 percent. For QUASAR we did not include the load time which was 30 seconds for the mouse EST data base and 114 seconds for the human EST data base.

3.1 Experiments

Three data sets were used for the tests. Firstly, the set of mouse EST sequences having no homology to known mouse mRNAs was used (198323 sequences, 75.2 Million base pairs (Mbps)). Since we are also engaged in clustering mouse ESTs, we produced a version of this database which was modified for clustering purposes (195671 sequences, 73.5 Mbps). In this set we removed the sequences containing typical mouse repeat sequences and preprocessed the data to limit the amount of "contextual" repeats. We removed sequence tails containing poly-X (X= $\{A, C, G, T, N\}$) which are frequently introduced by sequencing inaccuracies and are usually "clipped" by the sequencing centers. In fact, some sequences contained large poly-X strands of up to 300 bp. We also clipped the poly-A and poly-T signals at the beginning or end of the sequences. The largest data set we used for testing is the set of human ESTs as contained in the NCBI Human Unigene database (723675 sequences, 279.5 Mbps). This data set was preprocessed analogously to the mouse EST data, of course without removing common mouse repeat sequences.

BLAST was executed with the E-value threshold set to $E = 10^{-5}$, all other parameters were left at their default values. We ran QUASAR with a window length of $w = 50$ bps, q-grams of length 11 and a threshold t which guarantees to find windows with at most 6% difference (i.e., an edit distance of 3). All tests were conducted on one processor (SUN Ultra SparcII, 333 Mhz) of a dedicated Sun Enterprise 10000 with 4 GB of main memory and a local disk array.

3.2 Sensitivity

For the comparison between BLAST and QUASAR in terms of sensitivity, one would ideally compare E-values. However, as pointed out in the Algorithms Section, we postprocess the matches between the query and selected blocks using BLAST. We call this version QUASAR-BLAST to keep things apart. QUASAR-BLAST scans only a filtered subset of the entire database. Consequently for a pair of sequences, QUASAR-BLAST and BLAST E-values are not comparable because they refer to databases of different sizes.

Nevertheless, if a sequence is recorded as a hit by BLAST and QUASAR-BLAST it will yield the same alignment and the same score in both cases. This allows us to compare the lists of matches produced by QUASAR-BLAST and BLAST. Two outcomes are possible. If both lists contain the same hit-sequences, they will be in the same order with the exception of sequences with equal E-values. We report this as *identical results*. If the two lists differ, we report the BLAST E-value of the best match that QUASAR missed.

In the column *Identical results* in Table 1, one can see that in most cases (91.4% and 97.1%), QUASAR finds exactly the same hits as BLAST (with $E = 10^{-5}$). When the results differ (column *False negative*), the average E-values of the first missed match are 10^{-12} and 10^{-14} respectively. Note that we averaged the logarithms of the E-values, not the E-values themselves. The overall minimum E-values for false negatives are 10^{-34} and 10^{-31}. Given that we are guaranteed to have passed every block containing at least one window of more than 94% identity to QUASAR-BLAST and that a real match of ESTs typically has near-zero E-value, this assures us that we did not miss any sequences we intended to find. The cutoff of 10^{-5} for the BLAST E-value is not a value one would use to search for very conservative matches. Thus the large number of cases where QUASAR achieves the same sensitivity as BLAST shows that QUASAR is also able to find evolutionary divergent homologues of a given query.

The level of sequence identity in some window to which QUASAR is guaranteed to find a match was set to 6%. This level is for the "worst case", i.e., a match where differences are regularly spaced along the matching sequence. In practice we achieve better results: for mouse ESTs the maximum percentage of differences reported by QUASAR averaged over all 1000 queries is 12%. The overall maximum is 21%, with these numbers being computed on all matches having more than 6% differences. We are aware of the fact that theoretical and empirical analyses are necessary to aid the user in setting the parameters to achieve the level of sensitivity he wants. For mouse ESTs, we also evaluated the sensitivity for block sizes between 512 and 4096. The comparison with BLAST as described above yielded exactly the same results.

3.3 Performance

The last three columns of Table 1 summarize the average running times in seconds for QUASAR, QUASAR running with external memory (QUASAR-E) and BLAST. In the mouse and human EST databases, QUASAR searches

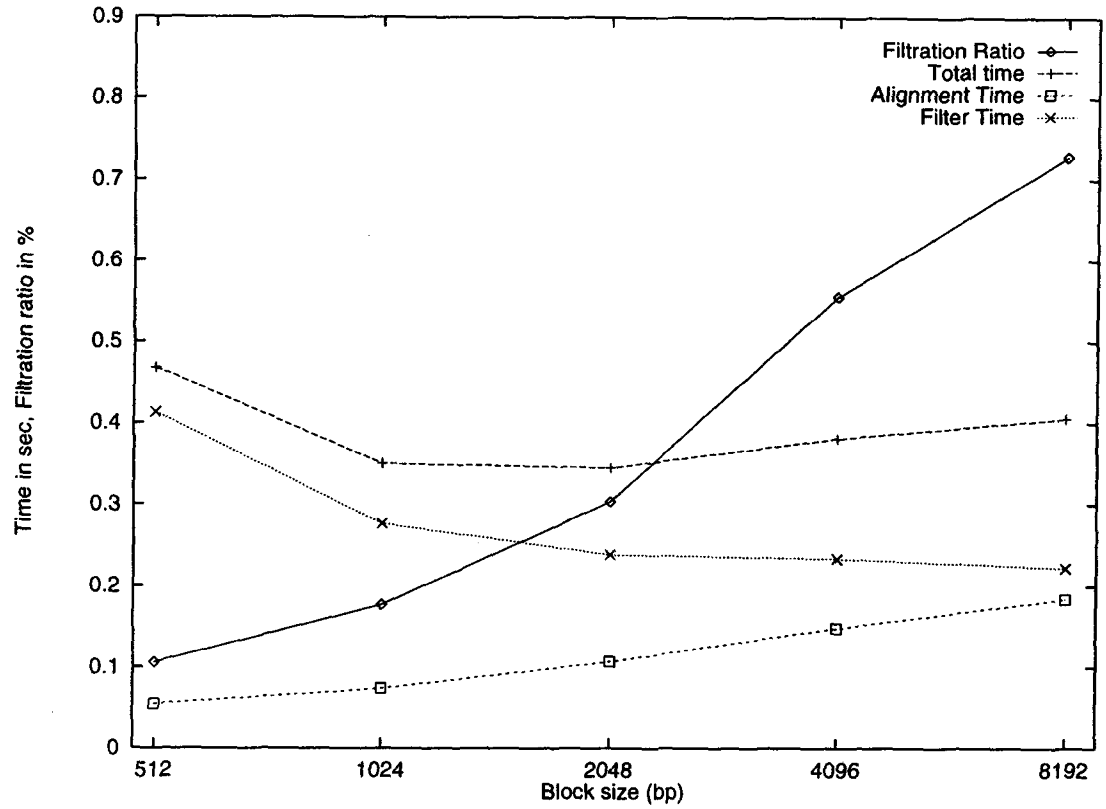

Figure 2: Running times and filtration ratio for various block sizes.

DB	Size	Queries	Identical	False Neg	Filtration	CPU times in seconds		
	Mbps	Size(bps)	Results	E-value	Ratio	QUASAR	QUASAR-E	BLAST
Mouse	73.5	368	91.4 %	10^{-12}	0.24%	0.123	0.128	3.37
Human	279.5	393	97.1 %	10^{-14}	0.17%	0.38	0.39	13.27

Table 1: *Sensitivity and running times of searches in mouse and human EST databases with block size of 1024bps. From left to right: database and its size, average query size, percentage of searches giving identical results, for searches yielding different results the minimum and average E-values of the first missed sequence, the filtration ratio and the CPU times of* QUASAR, QUASAR-E *and* BLAST.

through 73.5 Mpbs and 279.5 Mbps in 0.12 respectively 0.38 seconds. QUASAR-E with the suffix array on the disk achieves nearly the same performance with 0.13 and 0.39 seconds. Both QUASAR and QUASAR-E are approximately 30 times faster than BLAST. This improvement is especially significant because there has already been a lot of work and fine-tuning on BLAST.

The performance of a q-gram based approach like ours might be influenced by repeats or low-complexity regions. The running times on the original mouse EST database that still contains repeats, poly-A, etc. show that this effect is moderate. A search on this database required on average 0.25 seconds which is still considerably faster than BLAST. Furthermore, the external memory version QUASAR-E is nearly as fast as QUASAR. This is due to the sorting of the q-grams before accessing their hitlists on the disk and to much smaller data structures in main memory. In practice this results in a substantial increase in speed over BLAST and allows us to work on larger problems. For instance, we were able to run an all-versus-all comparison of the modified mouse ESTs (195671 queries) on an Ultra Sparc 2 with 1 GB of main memory in less than 11 hours. We estimated the time required to conduct these searches with BLAST to be larger than 10 days on the same machine.

3.4 Dependency of the filtration ratio on the block size

To evaluate the quality of our filtration, we measure what we call *filtration ratio*. It is defined as the percentage of the database which is selected by the filtration and passed to the alignment step. We compute the filtration ratio as follows:

$$\frac{\#\text{ filtered sequences}}{\#\text{ sequences in database}}$$

Since the length of ESTs does not vary much, this is approximately the same as counting numbers of base pairs instead of numbers of sequences. For block sizes between 512 and 8192, Figure 2 shows the total running time of QUASAR, the time spent on the filtration and the time spent in the alignment phase. In the same graph, we show the filtration ratio in percent.

The filtration ratio increases with the block size: for larger block sizes the portion of the database passed to the alignment step grows. Since for each interesting block all sequences covering that block are searched in the alignment phase, the total amount of sequences processed in this phase grows roughly linearly with the block size. In contrast to this, the time required to filter the database is reduced due to the smaller number of counters that have to be incremented in this phase and to the reduced size of the block counter array. It seems surprising that the time required for the alignment phase does not grow proportionally to the filtration ratio. This is caused by the overhead of calling the alignment algorithm and the fixed number of matches that are found in the alignment phase which are independent of the chosen block size.

It should be pointed out that the filtration step is extremely fast and accounts for less than one third of the total running time (for a block size of 2048bps). Thus, the speed of QUASAR would benefit most from improvements in the alignment phase. We therefore plan to improve the interface between the filtration and alignment step. The curve also shows that the optimal block size lies between 1024 and 4096 bps.

4 Discussion

The work presented here arose from a collaboration concerning the clustering, assembly, and analysis of EST sequences. Our focus on near-perfect matching of many queries stems from the interest in this problem. Correspondingly, it is not our intention to produce a substitute for BLAST or other current database searching methods. Our approach is intended as a complement to existing methods for applications where high similarity is expected and where many searches are performed together.

Our results show that our new approach is more than a magnitude faster than BLAST in identifying strongly similar sequences. In the case of an all-against-all comparison, where the running time is proportional to the squared number of sequences in the database, such an improvement leads to a substantial increase in overall performance. Similarly, one can envisage that a heavily used Web service might take advantage of the speed offered by our approach by collecting queries for a few seconds and then processing them as a group.

The good running times of QUASAR are due to the use of the precomputed suffix array. However, the algorithm to compute the suffix array, although of complexity $O(n\log(n))$, requires large amounts of internal memory and takes rather long in absolute time. Furthermore, for large data sets internal memory is likely to be insufficient for the construction and secondary memory versions are required to keep its running time within reasonable limits. This is an area of future research.

In further ongoing work we are studying in more depth the dependence of the running times of QUASAR on the size of the hit lists and the size of the database. We are working on the parallelization of our algorithm and investigating possibilities to improve the sensitivity. We are currently applying QUASAR to run all-against-all comparisons of human ESTs, mouse ESTs, and Arabidopsis ESTs. Based on this output, mouse ESTs have already been clustered, assembled, and representative clones have been selected for design of expression arrays.

5 Acknowledgements

This work was partially supported by the EU ESPRIT LTR Project N. 20244 (ALCOM-IT), WP 3.2. Stefan Burkhardt gratefully acknowledges financial support from a Graduiertenkolleg graduate fellowship of the Deutsche Forschungsgemeinschaft (DFG). Martin Vingron and Eric Rivals gratefully acknowledge financial support from BMBF within the German Human Genome Project.

6 Contacting the Authors

The authors can be contacted by sending email to `stburk@mpi-sb.mpg.de`.

References

[AGM+90] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[AMS+97] S.F. Altschul, T.L. Madden, A.A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman. Gapped Blast and Psi-Blast: a new generation of protein database search programs. *Nucleic Acids Res.*, 25:3389–3402, 1997.

[CL94] W. I. Chang and E. L. Lawler. Sublinear approximate string matching and biological applications. *Algorithmica*, 12(4/5):327–344, 1994.

[CR93] A. Califano and I. Rigoutsos. FLASH: A fast look-up algorithm for string homology. In L. Hunter, D. Searls, and J. Shavlik, editors, *Proc. of the 1st International Conf. on Intelligent Systems for Molecular Biology*, pages 56–64, 1993.

[Doo90] R. F. Doolittle. *What we have learned and will learn from sequence databases*, pages 21–31. Addison-Wesley, 1990.

[HBD94] W. Hide, J. Burke, and D. B. Davison. Biological evaluation of d2, an algorithm for high-performance sequence comparison. *J. Comp. Biol.*, 1:199–215, 1994.

[JU91] P. Jokinen and E. Ukkonen. Two algorithms for approximate string matching in static texts. In *Proc. of the 16th Symposium on Mathematical Foundations of Computer Science*, volume 520 of *Lecture Notes in Computer Science*, pages 240–248, 1991.

[Knu73] D. E. Knuth. *The Art of Computer Programming (Volume III): Sorting and Searching.* Addison-Wesley, Reading, MA, 1973.

[KV98] A. Krause and M. Vingron. A set-theoretic approach to database searching and clustering. *Bioinformatics*, 14:430–438, 1998.

[Mar83] H. M. Martinez. An efficient method for finding repeats in molecular sequences. *Nucleic Acids Research*, 11(13):4629–4634, 1983.

[McC76] E. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23(2):262–272, 1976.

[Heu97] K. Heumann. *Biologische Sequenzdatenanalyse großer Datensätze basierend auf Positionsbaumvarianten.* PhD thesis, 1997.

[MH95] H. W. Mewes and K. Heumann. Genome analysis: Pattern search in biological macromolecules. In *Combinatorial Pattern Matching*, Lecture Notes in Computer Science, vol. 937, Springer-Verlag, pages 261–285, 1995.

[MM93] U. Manber and E. W. Myers. Suffix Arrays: A new method for on-line string searches. *SIAM Journal on Computing*, 22(5): 935–948, 1993.

[Mye92] E. W. Myers. Algorithmic advances for searching biosequence databases. In Sándor Suhai, editor, *Computational Methods in Genome Research*, pages 121–135. Plenum Press, New York, 1992.

[Mye94] E. W. Myers. A sublinear algorithm for approximate keyword searching. *Algorithmica*, 12(4/5):345–374, 1994.

[OM88] O. Owolabi and D. R. McGregor. Fast approximate string matching. *Software Practice and Experience*, 18(4):387–393, 1988.

[PL88] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. *PNAS*, 85:2444–2448, 1988.

[PW93] P. A. Pevzner and M. S. Waterman. Multiple filtration and approximate pattern matching. *Algorithmica*, 13:135–154, 1995.

[RW94] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, pages 17–28, 1994.

[SW81] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[Ukk92] E. Ukkonen. Approximate string-matching with q-grams and maximal matches. *Theoretical Computer Science*, 92(1):191–211, 1992.

[VAS+98] J. C. Venter, M. D. Adams, G. G. Sutton, A. R. Kerlavage, H. O. Smith, and M. Hunkapillar. Shotgun sequencing of the human genome. *Science*, 280:1540–1542, 1998.

[Wal95] M. M. Waldrop. On-line archives let biologists interrogate the genome. *Science*, 269:1356–1358, 1995.

[WM92] S. Wu and U. Manber. Fast text searching allowing errors. *Communications of the ACM*, 35(10):83–91, 1992.