PoSSuMsearch: Fast and Sensitive Matching of Position Specific Scoring Matrices using Enhanced Suffix Arrays

Michael Beckstette[*][†] Dirk Strothmann[*] Robert Homann[*]
Robert Giegerich[*] Stefan Kurtz[‡]

Abstract: In biological sequence analysis, position specific scoring matrices (PSSMs) are widely used to represent sequence motifs. In this paper, we present a new non-heuristic algorithm, called *ESAsearch*, to efficiently find matches of such matrices in large databases. Our approach preprocesses the search space, e.g. a complete genome or a set of protein sequences, and builds an enhanced suffix array which is stored on file. The enhanced suffix array only requires 9 bytes per input symbol, and allows to search a database with a PSSM in sublinear expected time. We also address the problem of non-comparable PSSM-scores by developing a method which allows to efficiently compute a matrix similarity threshold for a PSSM, given an E-value or a p-value. Our method is based on dynamic programming. In contrast to other methods it employs lazy evaluation of the dynamic programming matrix: it only evaluates those matrix entries that are necessary to derive the sought similarity threshold. We tested algorithm *ESAsearch* with nucleotide PSSMs and with amino acid PSSMs. Compared to the best previous methods, *ESAsearch* show speedups of a factor between 4 and 50 for nucleotide PSSMs, and speedups up to a factor 1.8 for amino acid PSSMs. Comparisons with the most widely used programs even show speedups by a factor of at least 10. The lazy evaluation method is also much faster than previous methods, with speedups by a factor of at least 10.

1 Introduction

Position specific scoring matrices (PSSMs) have a long history in sequence analysis (see [GME87]). A high PSSM-score in some region of a sequence often indicates a possible biological relationship of this sequence to the family or motif characterized by the PSSM. There are several databases incorporating PSSMs, e.g. PROSITE [HSL$^+$04], PRINTS [ABF$^+$03], BLOCKS [HGPH00], or TRANSFAC [MFG$^+$03]. While these databases are constantly improved, there are only few improvements in the programs searching with PSSMs. E.g. the programs *FingerPrintScan* [SFA99], *BLIMPS* [HGPH00], and *MatInspector* [QFWW95] still use a simple straightforward $O(mn)$-time algorithm to search a PSSM of length m in a sequence of length n. The most advanced program in this field is *EMATRIX* [WNB00], which incorporates a technique called lookahead scoring. The lookahead scoring technique is also employed in the suffix tree based method of [DC00]. This method performs a limited depth first traversal of the suffix tree of the set of target sequences. This search updates PSSM-scores along the edges of the suffix tree. Lookahead

[*]Technische Fakultät, Universität Bielefeld, Postfach 100 131, D-33501 Bielefeld, Germany
[†]Corresponding author, Email: mbeckste@techfak.uni-bielefeld.de
[‡]Zentrum für Bioinformatik, Universität Hamburg, 20146 Hamburg, Germany

scoring allows to skip subtrees of the suffix tree that do not contain any matches to the PSSM. Unfortunately, the method of [DC00] has not found its way into a widely available and robust software system.

In this paper, we present a new algorithm for searching PSSMs. The overall structure of the algorithm is similar to the method of [DC00]. However, instead of suffix trees we use enhanced suffix arrays, a data structure which is as powerful as suffix trees (cf. [AKO04]). Enhanced suffix arrays provide several advantages over suffix trees, which make them more suitable for searching PSSMs:

- While suffix trees require about $12n$ bytes in the best available implementation (cf. [Ku99]), the enhanced suffix array used for searching with PSSMs only needs $9n$ bytes of space.
- While the suffix tree is only computed in main memory, the enhanced suffix array is computed once and stored on file. Whenever a PSSM is to be searched with, the enhanced suffix array is mapped into main memory which requires no extra time.
- While the limited depth first traversal of the suffix tree suffers from the poor locality behavior of the data structure (cf. [GK95]), the enhanced suffix array provides optimal locality, because it is sequentially scanned from left to right.

One of the algorithmic contributions of this paper is a new technique that allows to skip parts of the enhanced suffix array containing no matches to the PSSM. Due to the skipping, our algorithm achieves an expected running time that is sublinear in the size of the search space (i.e. the size of the nucleotide or protein database). As a consequence, our algorithm scales very well for large data sizes.

When searching with a PSSM it is very important to determine a suitable threshold for a PSSM-match. Usually, the user prefers to specify a significance threshold (i.e. an E-value or a p-value) which has to be transformed into an absolute score threshold for the PSSM under consideration. This can be done by computing the score distribution of the PSSM, using well-known dynamic programming (DP, for short) methods, e.g. [St89, WNB00, Ra03, RMV03]. Unfortunately, these methods are not fast enough for large PSSMs. For this reason, we have developed a new lazy evaluation algorithm that only computes a small fraction of the complete score distribution. Our algorithm speeds up the computation of the threshold by factor of at least 10, compared to standard DP methods. This makes our algorithm applicable for on-the-fly computations of the score thresholds.

The new algorithms described in this paper are implemented as part of the *PoSSuMsearch* software package. This is available free of charge for non-commercial research institutions. For details, see http://bibiserv.techfak.uni-bielefeld.de/possumsearch/.

2 PSSMs and lookahead scoring: *LAsearch*

A PSSM is a representation of a multiple alignment of related sequences. We define it as a function $M : [0, m-1] \times \Sigma \to \mathbb{R}$, where m is the length of M and Σ is a finite alphabet. Usually M is represented by an $m \times |\Sigma|$ matrix, see Table 1 for an example. Each row of the matrix reflects the frequency of occurrence of each amino acid or nucleotide at the

A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y	th_d	σ_d
-19	**92**	-45	-49	-30	-36	-38	-12	-41	-21	-22	-40	-46	-44	-44	-30	-25	16	-35	-34	**2**	**398**
5	-17	17	**22**	-28	-15	-7	-23	-8	-27	-21	26	18	-7	-13	-9	9	-19	-33	-25	**24**	**376**
7	-8	-29	-28	2	-25	-10	25	-23	-4	-5	-25	-32	-26	-25	-18	13	22	-11	**36**	**60**	**340**
-29	**99**	-55	-61	-42	-45	-47	-31	-52	-34	-36	-49	-56	-55	-55	-38	-35	-29	-44	-46	**159**	**241**
-14	-22	14	**22**	-28	9	-8	-26	15	-27	-20	-7	-26	-3	31	-13	5	-23	-30	-24	**181**	**219**
-25	-34	-25	-16	-37	-30	-15	-36	45	-34	-26	-18	-35	-9	**49**	-25	-26	-33	-39	-31	**230**	**170**
7	-8	-25	-24	-19	-23	-22	4	-15	-10	-8	-19	-29	-21	11	-13	**31**	31	-31	-22	**261**	**139**
-34	-27	-44	-43	60	-41	-8	-16	-38	-14	-17	-39	-51	-40	-36	-39	-35	-21	-1	**56**	**317**	**83**
7	**40**	-16	-14	-9	-14	-6	-17	14	-20	-15	-10	-24	-11	12	15	9	-13	-16	20	**357**	**43**
-7	**43**	16	-7	-27	-15	-9	-24	-5	-26	-18	-6	-25	25	13	25	-8	-21	-30	-24	**400**	**0**

Table 1: Amino acid PSSM of length $m = 10$ of a zinc-finger motif. If the score threshold is $th = 400$, then only substrings beginning with C or V can match the PSSM, because all other amino acids score below the intermediate threshold $th_0 = 2$. That is, lookahead scoring will skip over all substrings which begin with amino acids different from C and V.

corresponding position of the alignment. From now on, let M be a PSSM of length m. We define $sc(w, M) = \sum_{i=0}^{m-1} M(i, w[i])$ for a sequence $w \in \Sigma^m$ of length m. $sc(w, M)$ is the *match score of* w w.r.t. to M. Given a sequence S of length n over alphabet Σ and a score threshold th, the *PSSM searching problem* is to find all positions $j \in [0, n-m]$ in S s.t. $sc(S[j..j+m-1], M) \geq th$.

A simple algorithm for the PSSM searching problem slides along the sequence and computes $sc(w, M)$ for each $w = S[j..j+m-1]$, $j \in [0, n-m]$. The running time of this algorithm is $O(mn)$. It is used e.g. in the programs *FingerPrintScan* [SFA99], *BLIMPS* [HGPH00], *MatInspector* [QFWW95], and *MATCH* [KGR+03].

In [WNB00], lookahead scoring is introduced to improve the simple algorithm. Lookahead scoring allows to stop the calculation of $sc(w, M)$ when it is clear that the given overall score threshold th cannot be achieved. To explain the method, define $pfxsc_d(w, M) = \sum_{h=0}^{d} M(h, w[h])$, $\max_d = \max\{M(d, a) \mid a \in \Sigma\}$, and $\sigma_d = \sum_{h=d+1}^{m-1} \max_h$ for any $d \in [0, m-1]$. $pfxsc_d(w, M)$ is the *prefix score of depth* d. σ_d is the maximal score that can be achieved in the last $m-d-1$ positions of the PSSM. Let $th_d = th - \sigma_d$ be the *intermediate threshold* at position d. It is easy to prove that $sc(w, M) \geq th$ implies $pfxsc_d(w, M) \geq th_d$ for all $d \in [0, m-1]$. This gives a necessary condition for a PSSM-match which can easily be exploited: When computing $sc(w, M)$ by scanning w from left to right, one checks for $d = 0, 1, \ldots$, if the intermediate threshold th_d is achieved. If not, the computation can be stopped. See Table 1 for an example. The lookahead scoring algorithm (called *LAsearch*) runs in $O(kn)$ time, where k is the average number of PSSM-positions per sequence position actually evaluated. In the worst case, $k \in O(m)$, which leads to the worst case running time of $O(mn)$, not better than the simple algorithm. However, k is expected to be much smaller than m, leading to considerable speedups in practice.

Our reformulation of lookahead scoring and its implementation is the basis for improvements and evaluation in the subsequent sections.

i	suf[i]	lcp[i]	skp[i]	$S_{suf[i]}$
0	1		12	aaaaccacac$
1	2	3	2	aaaccacac$
2	3	2	3	aaccacac$
3	7	1	6	acac$
4	4	2	6	accacac$
5	9	2	6	ac$
6	0	0	12	caaaaccacac$
7	6	2	9	cacac$
8	8	3	9	cac$
9	5	1	11	ccacac$
10	10	1	11	c$
11	11	0	12	$

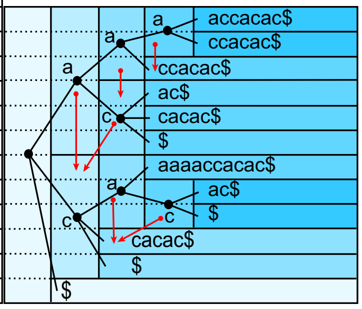

Figure 1: The enhanced suffix array and the suffix tree for sequence $S = \texttt{caaaaccacac}$. Some skp entries are shown in the tree as red arrows: If $\mathsf{skp}[i] = j$, then an arrow points from row i to row j.

3 PSSM searching using enhanced suffix arrays: *ESAsearch*

The enhanced suffix array for a given sequence S of length n consists of three tables suf, lcp, and skp. Let $ be a symbol in Σ, larger than all other symbols, which does not occur in S. suf is a table of integers in the range 0 to n, specifying the lexicographic ordering of the $n+1$ suffixes of the string $S\$$. That is, $S_{\mathsf{suf}[0]}, S_{\mathsf{suf}[1]}, \ldots, S_{\mathsf{suf}[n]}$ is the sequence of suffixes of $S\$$ in ascending lexicographic order, where $S_i = S[i..n-1]\$$ denotes the ith nonempty suffix of the string $S\$$, for $i \in [0, n]$. suf can be constructed in $O(n)$ time and requires $4n$ bytes.

lcp is a table in the range 0 to n such that $\mathsf{lcp}[i]$ is the length of the longest common prefix of $S_{\mathsf{suf}[i-1]}$ and $S_{\mathsf{suf}[i]}$, for $i \in [1, n]$. Table lcp can be computed in linear time from suf. For PSSM searching we do not access values in table lcp larger than 255, and hence we can store it in n bytes.

skp is a table in the range 0 to n s.t. $\mathsf{skp}[i] = \min(\{n+1\} \cup \{j \in [i+1, n] \mid \mathsf{lcp}[i] > \mathsf{lcp}[j]\})$. In terms of suffix trees, $\mathsf{skp}[i]$ denotes the lexicographically next leaf that does not occur in the subtree below the branching node corresponding to the longest common prefix of $S_{\mathsf{suf}[i-1]}$ and $S_{\mathsf{suf}[i]}$. Fig. 1 shows this relation. Array skp can be computed in $O(n)$ time given suf and lcp. For the algorithm to be described we assume that the enhanced suffix array for S has been precomputed.

In a suffix tree, all substrings of S of a fixed length m can be scored with a PSSM by a depth first traversal of the tree. Using lookahead scoring, one can skip certain subtrees that do not contain matches to the PSSM. Since suffix trees have several disadvantages (see the introduction), we use enhanced suffix arrays to search PSSMs. Like in other algorithms on enhanced suffix arrays (cf. [AKO04]), one simulates a depth first traversal of the suffix tree by processing the arrays suf and lcp from left to right. To incorporate lookahead scoring while searching we must be able to skip certain ranges of suffixes in suf. To facilitate this, we use table skp. We will now make this more precise.

For $i \in [0,n]$, let $v_i = S_{\mathsf{suf}[i]}$, $l_i = \min\{m, |v_i|\} - 1$, and $d_i = \max(\{-1\} \cup \{d \in [0,l_i] \mid \mathit{pfxsc}_d(v_i, M) \geq \mathit{th}_d\})$. Now observe that $d_i = m-1 \Leftrightarrow \mathit{pfxsc}_{m-1}(v_i, M) \geq \mathit{th}_{m-1} \Leftrightarrow \mathit{sc}(v_i, M) \geq \mathit{th}$. Hence, M matches at position $j = \mathsf{suf}[i]$ if and only if $d_i = m-1$. Thus, to solve the PSSM searching problem, it suffices to compute all $i \in [0,n]$ satisfying $d_i = m-1$. We compute d_i along with $C_i[d] := \mathit{pfxsc}_d(v_i, M)$ for any $d \in [0, d_i]$. d_0 and C_0 are easily determined in $O(m)$ time. Now let $i \in [1,n]$ and suppose that d_{i-1} and $C_{i-1}[d]$ are determined for $d \in [0, d_{i-1}]$. Since v_{i-1} and v_i have a common prefix of length $\mathsf{lcp}[i]$, we have $C_i[d] = C_{i-1}[d]$ for any $d \in [0, \mathsf{lcp}[i]-1]$. Consider the following cases:

- If $d_{i-1}+1 \geq \mathsf{lcp}[i]$, then compute $C_i[d]$ for $d \geq \mathsf{lcp}[i]$ while $d \leq l_i$ and $C_i[d] \geq \mathit{th}_d$. We obtain $d_i = d$.
- If $d_{i-1}+1 < \mathsf{lcp}[i]$, then let j be the minimum value in the range $[i+1, n+1]$ such that all suffixes $v_i, v_{i+1}, \ldots, v_{j-1}$ have a common prefix of length $d_{i-1}+1$ with v_{i-1}. Due to the common prefix we have $\mathit{pfxsc}_d(v_{i-1}, M) = \mathit{pfxsc}_d(v_r, M)$ for all $d \in [0, d_{i-1}+1]$ and $r \in [i, j-1]$. Hence $d_{i-1} = d_r$ for $r \in [i, j-1]$. If $d_{i-1} = m-1$, then there are PSSM matches at all positions $\mathsf{suf}[r]$ for $r \in [i, j-1]$. If $d_{i-1} < m-1$, then there are no PSSM matches at any of these positions. That is, we can directly proceed with index j. We obtain j by following a chain of entries in table skp: compute a sequence of values $j_0 = i$, $j_1 = \mathsf{skp}[j_0], \ldots, j_k = \mathsf{skp}[j_{k-1}]$ such that $d_{i-1}+1 < \mathsf{lcp}[j_1], \ldots, d_{i-1}+1 < \mathsf{lcp}[j_{k-1}]$, and $d_{i-1}+1 \geq \mathsf{lcp}[j_k]$. Then $j = j_k$.

These case distinctions lead to the program *ESAsearch*. In the worst case, *ESAsearch* has to evaluate m positions for each suffix of S. This leads to a running time of $O(mn)$. However, if the score threshold is stringent, then often large ranges of suffixes can be skipped, leading to a sublinear running time.

4 Finding an appropriate threshold for PSSM searching: *LazyDistrib*

4.1 Probabilities and Expectation values

The results of PSSM searches strongly depend on the choice of an appropriate threshold value th. A small threshold may produce a large number of false positive matches without any biological meaning, whereas meaningful matches may not be found if the threshold is too stringent. PSSM-scores are not equally distributed and thus scores of two different PSSMs are not comparable. It is therefore desirable to let the user define a significance threshold instead. The expected number of matches in a given random sequence database (E-value) is a widely accepted measure of the significance. We can compute the E-value for a known background distribution and length of the database by exhaustive enumeration of all substrings. However, the time complexity of such a computation is $O(|\Sigma|^m m)$ for a PSSM of length m. If the values in M are integers within a certain range $[r_{min}, r_{max}]$ of width $R = r_{max} - r_{min} + 1$, then DP methods (cf. [St89, WNB00, Ra03, RMV03]) allow to compute the probability distribution (and hence the E-value) in $O(m^2 R |\Sigma|)$ time.

While recent publications focus on the computation of the complete probability distribution, what is required specifically for PSSM matching, is computing a partial cumulative distribution corresponding to an E-value resp. p-value specified by the user. Therefore we have developed a new "lazy" method to efficiently compute only a small fraction of the complete distribution.

We formulate the problem we solve w.r.t. to E-values and p-values: Given a user specified E-value η, find the minimum threshold $\mathit{Tmin}_E(\eta, M)$, such that the expected number of matches of M in sequence S is at most η. Given a user specified p-value π, find the minimum threshold $\mathit{Tmin}_{\mathcal{P}}(\pi, M)$, such that the probability that M matches a random string of length m is at most π.

The threshold $\mathit{Tmin}_E(\eta, M)$ can be computed from $\mathit{Tmin}_{\mathcal{P}}(\pi, M)$ according to the equation $\mathit{Tmin}_E(\pi n, M) = \mathit{Tmin}_{\mathcal{P}}(\pi, M)$. Hence we restrict on computing $\mathit{Tmin}_{\mathcal{P}}(\pi, M)$.

Since all strings of length m have a score between $sc_{\min}(M) = \sum_{d=0}^{m-1} \min\{M(d, a) \mid a \in \Sigma\}$ and $sc_{\max}(M) = \sum_{d=0}^{m-1} \max\{M(d, a) \mid a \in \Sigma\}$, we conclude $\mathit{Tmin}_{\mathcal{P}}(1, M) = sc_{\min}(M)$ and $\mathit{Tmin}_{\mathcal{P}}(0, M) > sc_{\max}(M)$. To explain our lazy evaluation method, we first consider existing methods based on DP.

4.2 Evaluation with dynamic programming

We assume that at each position in sequence S, the symbols occur independently, with probability $f(a) = (1/n) \cdot |\{i \in [0, n-1] \mid S[i] = a\}|$. Thus a substring w of length m in S occurs with probability $\prod_{i=0}^{m-1} f(w[i])$ and the probability of observing the event $sc(w, M) = t$ is $\mathbb{P}[sc(w, M) = t] = \sum_{w \in \Sigma^m, sc(w,M)=t} \prod_{i=0}^{m-1} f(w[i])$. We obtain $\mathit{Tmin}_{\mathcal{P}}(\pi, M)$ by a lookup in the distribution:

$$\mathit{Tmin}_{\mathcal{P}}(\pi, M) = \min\{t \mid sc_{\min}(M) \leq t < sc_{\max}(M) + 1, \mathbb{P}[sc(w, M) \geq t] \leq \pi\}.$$

If the values in the PSSM M are integers in a range of width R, dynamic programming allows to efficiently compute the probability distribution (see Figure 2). The dynamic programming aspect becomes more obvious by introducing for each $k \in [0, m-1]$ the *prefix* PSSM $M_k : [0, k] \times \Sigma \to \mathbb{N}$ defined by $M_k(j, a) = M(j, a)$ for $j \in [0, k]$ and $a \in \Sigma$. Corresponding distributions $Q_k(t)$ for $k \in [0, m-1]$ and $t \in [sc_{\min}(M_k), sc_{\max}(M_k)]$, and $Q_{-1}(t)$, are defined by

$$Q_k(t) = \begin{cases} \text{if } t = 0 \text{ then } 1 \text{ else undefined} & \text{if } k = -1 \\ \sum_{a \in \Sigma} Q_{k-1}(t - M(k, a)) f(a) & \text{otherwise} \end{cases}$$

We have $\mathbb{P}[sc(w, M) = t] = Q_{m-1}(t)$. The algorithm computing Q_k determines a set of probability distributions for $M_0, \ldots, M_k$. Q_k is evaluated in $O(sc_{\max}(M)|\Sigma|)$ time from Q_{k-1}, summing up to $O(sc_{\max}(M)|\Sigma|m)$ total time.

If we allow for floating point scores that are rounded to ϵ decimal places, the time and space requirement increases by a factor of 10^{ϵ}. Conversely, if all integer scores share a greatest common divisor z, the matrix should be canceled down by z.

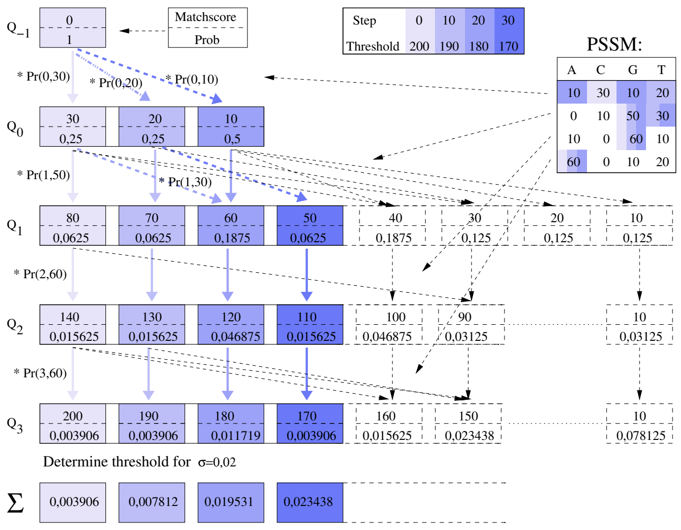

Figure 2: The simple DP scheme of section 4.2 computes all probability vectors Q_0, Q_1, Q_2, Q_3 completely. Scores that cannot occur at all (like 199) are omitted in the figure. Since $sc_{\max}(M) = 200$, sums over $Q(200), Q(199), Q(198), \ldots$ are computed until the given p-value $\pi = 0.02$ is exceeded. The sought score threshold is $Tmin_{\mathcal{P}}(\pi, M) = 171$. In contrast to the simple scheme, the restricted computation only evaluates the colored parts of the probability vectors. In step 0 $Q(200)$ is computed, as indicated by the same color for all concerned components. For threshold $t = 180$ we obtain the intermediate thresholds $th_2 = 120$, $th_1 = 60$, and $th_0 = 10$. After step 30, in which $Q(170)$ is computed, the rest of the computation can be skipped.

4.3 Restricted probability computation

In order to find $Tmin_{\mathcal{P}}(\pi, M)$ it is not necessary to compute the whole codomain of the distribution function $Q = Q_{m-1}$. We propose a new method only computing a partial distribution by summing over the probabilities for decreasing threshold values $sc_{\max}(M)$, $sc_{\max}(M) - 1, \ldots$, until the given p-value π is exceeded.

In step d we compute $Q(sc_{\max}(M) - d)$ where all intermediate scores contributing to $sc_{\max}(M) - d$ have to be considered. In analogy to lookahead scoring, in each row j of M we avoid all intermediate scores below the intermediate threshold th_j because they do not contribute to $Q(sc_{\max}(M) - d)$. The algorithm stops if the cumulated probability for threshold $sc_{\max}(M) - d$ exceeds the given p-value π and we obtain $Tmin_{\mathcal{P}}(\pi, M) = sc_{\max}(M) - d + 1$.

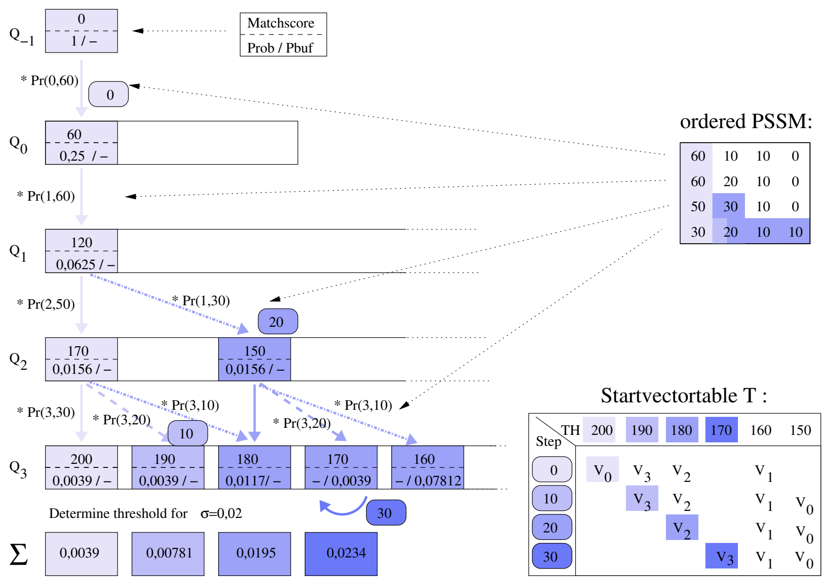

Figure 3: In this example we use the same PSSM as in Figure 2. However, in each row the scores are sorted in descending order, and the rows are sorted with the most discriminant row coming first. For every threshold th, table T contains the index i, where the probability vectors $Q_i, Q_{i+1}, \ldots, Q_{m-1}$ have to be updated for the computation of $Q(th)$. Vectors $Q_0, \ldots, Q_{i-1}$ can be skipped. In step 0 $Q(200)$ is computed, resulting in the update of Q_0, Q_1, Q_2, Q_3. In step 20, the buffers of $Q_3(170)$ and $Q_3(160)$ are filled to avoid re-access of $M(2, C) = \max_2 - 20 = 30$. In step 30, the buffer of $Q_3(170)$ is added to $Q_3(170)$ and the computation is finished.

4.4 Lazy evaluation of the permuted matrix

The restricted computation strategy performs best if there are only few iterations (i.e. $Tmin_{\mathcal{P}}(\pi, M)$ is close to $sc_{\max}(M)$) and in each iteration step the computation of $Q_k(t)$ can be skipped in an early stage, i.e. for small values of k. The latter occurs to be more likely if the first rows of M contain strongly discriminative values leading to the exclusion of the small values by comparison with the intermediate thresholds. An example of this situation is given in Table 1. Since $Q_k(t)$ is invariant to the permutation of the rows of M, we can sort the rows of M such that the most discriminative rows come first. We found that the difference between the largest two values of a row is a suitable measure for the level of discrimination since a larger difference increases the probability to remain below the intermediate threshold. Since the rows of M are scanned several times, we save time by initially sorting each row in order of descending score.

We divide the computation steps where the step d computes $Q(sc_{\max}(M) - d)$: In step 0 only the maximal scores $\max_i$, $i \in [0, m-1]$ in each row have to be evaluated.

In step $d > 0$ all scores $M(i,a) \geq \max_i - d$ may contribute to $Q(sc_{\max}(M) - d)$. Since in general a score value $M(i,a) \geq \max_i - d$ also gives contribution to $Q(sc_{\max}(M) - l), l > d$ we can save time by storing $Q_i(\max_i - l), l > d$ in step d in a buffer and reusing the buffer in steps $d+1, d+2, \ldots$. This allows for the computation of $Q_k(sc_{\max}(M) - d)$ only based on the buffer and scores $M(i,a) = \max_i - d$ while scores $M(i,a) > \max_i - d$, $i \in [0, m-1]$ can be omitted. We therefore have developed an algorithm *LazyDistrib* employing lazy evaluation of the distribution: Given a threshold t, the algorithm only evaluates parts of the DP vectors necessary to determine $Q_k(t)$ and simultaneously saves subresults concerned with score t in an additional buffer matrix $Pbuf$ (instead of recomputing them later) (see Figure 3). This is described by the following recurrence:

$$\begin{aligned} Q_k(t-d) &= Pbuf_{M_k}(t-d) + \\ &\quad \sum_{a \in \Sigma, M(k,a) \geq \max_k - d} Q_{k-1}(t - d - M(k,a)) f(a) \\ Pbuf_{M_k}(t-d) &= \sum_{a \in \Sigma, M(k-1,a) < \max_k - d} Q_{k-1}(t - d - M(k-1,a)) f(a) \end{aligned}$$

In the present implementation, the algorithm assumes independently distributed symbols. The algorithm can be extended to an order d-Markov model (w.r.t. the background alphabet distribution). This increases the computation time by a factor of $|\Sigma|^d$.

5 Implementation and Results

We implemented *LAsearch*, *ESAsearch*, and *LazyDistrib* in C as part of our program *PoSSuMsearch*. The program was compiled with the GNU C compiler (version 3.1, optimization option `-O3`). All measurements were performed on a 8 CPU Sun UltraSparc III computer running at 900MHz, with 64GB main memory (using only one CPU and a

	Experiment 1	Experiment 2	Experiment 3	Experiment 4
# searched sequences	59021	15668	5000	1 (*H.s.* Chr. 6)
total length	20.2 MB	17.9 MB	2.0 MB	162.9 MB
sequence source	see [DC00]	DBTSS 3.0	SwissProt 42.8	Sanger V1.4
sequence type/PSSM type	protein	DNA	protein	DNA
# PSSMs	4034	219	10931	576
PSSM source	see [DC00]	MatInspector	PRINTS 36	TRANSFAC Prof. 6.2
avg. length of PSSMs	29.74	14.21	17.37	13.33
index construction (sec)	41	32	2.3	586
mdc (min)	129	–	480	–
MatInspector		×		
FingerPrintScan			×	
DN00	×			
LAsearch	×	×	×	×
ESAsearch	×	×	×	×

Table 2: Overview of the sequences and PSSMs used in the performed experiments. For the experiments that use p-value or E-value cutoffs, we precomputed the cumulative score distributions and stored them on file. *mdc* is the time needed for this task. In Experiment 1 we measured the running time of the Java-program from [DC00], referred to by *DN00*. We ran *DN00* with a maximum of 2 GB memory assigned to the Java virtual machine. *DN00* constructs the suffix tree in main memory and then performs the searches. For a fair comparison, we therefore measured the total running time, and the time for matching the PSSMs (without suffix tree construction). For Experiment 2, we implemented the matrix similarity scoring scheme (MSS) of *MatInspector* and matched the PSSMs against both strands of the DNA sequences with different MSS cutoff values. Instead of using the reverse strand we use the reverse complement $\overline{M}$ of the PSSM M, defined by $\overline{M}(i,a) = M(m-1-i,\overline{a})$ for all $i \in [0, m-1]$ and $a \in \Sigma$, where $\overline{a}$ is the Watson Crick complement of nucleotide a. This allows to use the same enhanced suffix array for both strands. The last five rows show which programs were used in which experiment.

small fraction of the memory). Enhanced suffix arrays were constructed with the program *mkvtree*, see http://www.vmatch.de/.

We performed four experiments comparing different programs for searching PSSMs. Table 2 gives more details on the experimental input. In these experiments *ESAsearch* performed very well, especially on nucleotide PSSMs, see Experiments 2 and 4. It is faster than *MatInspector* by a factor between 11 and 697, depending on the stringency of the given thresholds. The commercial advancement of *MatInspector*, called *MATCH* was not available for our comparisons, but based on [MFG$^+$03] we presume a running time comparable to *MatInspector*. Compared to *LAsearch*, *ESAsearch* is faster by a factor between 4 and 50. In the experiments using protein PSSMs, *ESAsearch* is faster than the method of [DC00] by a factor between 1.5 and 1.8 (see Experiment 1). This is due to the better locality behavior of the enhanced suffix array compared to a suffix tree. For larger p-values *LAsearch* performs slightly better than *ESAsearch*. Increasing the stringency, the performance of *ESAsearch* increases, resulting in a speedup of factor 1.5 for a p-value of 10^{-40}. We explain this behavior by the larger alphabet size, resulting in shorter common prefixes and therefore smaller skipped areas of the enhanced suffix array. With increasing stringency of the threshold, the number of positions in each suffix to score decreases, resulting in smaller values for d_i, and finally larger skipped areas of the enhanced suffix array. Compared to the *FingerPrintScan* program, *ESAsearch* achieves a speedup factor between 55

Experiment 1: 4034 PSSMs in 21.2 MB protein sequences				
p-value	*DN00* (total time)	*DN00* (search)	*LAsearch*	*ESAsearch* $+41$ sec.
10^{-10}	65808	64939	39839	41813
10^{-20}	38773	37706	23786	24378
10^{-30}	21449	20362	14111	13084
10^{-40}	9606	8533	8067	5374

Experiment 2: 219 PSSMs in 18.8 MB DNA			
MSS	*MatInspector*	*LAsearch*	*ESAsearch* $+32$ sec.
0.80	6440	2239	553
0.85	6344	1826	278
0.90	6303	1542	143
0.95	6283	1261	62
1.00	6273	452	9

Experiment 3: 10931 PSSMs in 2 MB protein sequences			
E-value	*FingerPrintScan*	*LAsearch*	*ESAsearch* $+2.3$ sec.
10^{-10}	13521	1922	242
10^{-20}	13521	882	20
10^{-30}	13516	814	12

Experiment 4: 576 PSSMs in 169 MB DNA		
MSS	*LAsearch*	*ESAsearch* $+586$ sec.
0.85	38432	1217
0.90	34127	815
0.95	28705	596
1.00	11016	452

Table 3: Experiment 1: Running times in seconds of the different PSSM searching methods at different levels of stringency, when searching for 4034 amino acid PSSMs in 59021 sequences (21.2 MB) from SwissProt. These are the same PSSMs and sequences used in the experiments of [DC00]. Experiment 2: Running times in seconds of *MatInspector*, *LAsearch*, and *ESAsearch*, when searching 219 PSSMs on both strands of 18.8 MB DNA sequence data at different matrix similarity score (MSS) cutoffs. Experiment 3: Running times in seconds of *FingerPrintScan*, *LAsearch*, and *ESAsearch* when searching all 10931 PSSMs from the PRINTS database in the first 5000 sequences of SwissProt for different E-values. Experiment 4: Running times in seconds of *LAsearch* and *ESAsearch* when searching 576 PSSMs in H. sapiens chr. 6 at different matrix similarity score (MSS) cutoffs. The additional time needed for the construction of the enhanced suffix array is shown for each experiment in the head of the *ESAsearch* column.

and 1126, see Experiment 3. In a final experiment, we compared algorithm *LazyDistrib* with the DP-algorithm computing the complete distribution. *LazyDistrib* shows a speedup factor of 16 on our test set (see Table 4).

p-value	simple DP	*LazyDistrib*	speedup factor
10^{-10}	4800	479	10.0
10^{-20}	4800	320	15.0
10^{-30}	4800	304	15.8
10^{-40}	4800	298	16.1

Table 4: Running times in seconds when computing score thresholds for all 10931 PSSMs from the PRINTS database, given different p-values.

6 Conclusion

We have presented a new non-heuristic algorithm for searching PSSMs, achieving expected sublinear running time. It shows superior performance over the most widely used programs, especially for DNA sequences. The enhanced suffix array, on which the method is based, requires only $9n$ bytes. This is a space reduction of more than 45 percent compared to the $17n$ bytes implementation of [DC00]. Our second main contribution is a new algorithm for the efficient calculation of score thresholds from user defined E-values and p-values. The algorithm allows for accurate on-the-fly calculations of thresholds, and has the potential to replace formerly used approximation approaches.

References

[ABF+03] Attwood, T. K., Bradley, P., Flower, D. R., Gaulton, A., Maudling, N., Mitchell, A. L., Moulton, G., Nordle, A., Paine, K., Taylor, P., Uddin, A., and Zygouri, C.: PRINTS and its automatic supplement, prePRINTS. *Nucl. Acids Res.* **31**(1):400–402. 2003.

[AKO04] Abouelhoda, M., Kurtz, S., and Ohlebusch, E.: Replacing Suffix Trees with Enhanced Suffix Arrays. *Journal of Discrete Algorithms*. **2**:53–86. 2004.

[DC00] Dorohonceanu, B. and C.G., N.-M.: Accelerating Protein Classification Using Suffix Trees. In: *in Proc. of the International Conference on Intelligent Systems for Molecular Biology*. pp. 128–133. Menlo Park, CA. 2000. AAAI Press.

[GK95] Giegerich, R. and Kurtz, S.: A Comparison of Imperative and Purely Functional Suffix Tree Constructions. *Science of Computer Programming*. **25**(2-3):187–218. 1995.

[GME87] Gribskov, M., McLachlan, M., and Eisenberg, D.: Profile Analysis: Detection of Distantly Related Proteins. *Proc. Nat. Acad. Sci. U.S.A.* **84**:4355–4358. 1987.

[HGPH00] Henikoff, J., Greene, E., Pietrokovski, S., and Henikoff, S.: Increased Coverage of Protein Families with the Blocks Database Servers. *Nucl. Acids Res.* **28**:228–230. 2000.

[HSL+04] Hulo, N., Sigrist, C., Le Saux, V., Langendijk-Genevaux, P. S., Bordoli, L., Gattiker, A., De Castro, E., Bucher, P., and Bairoch, A.: Recent improvements to the PROSITE database. *Nucl. Acids Res.* **32**:134–137. 2004.

[KGR+03] Kel, A., Gößling, E., Reuter, I., Cheremushkin, E., Kel-Margoulis, O., and Wingender, E.: MATCH: a tool for searching transcription factor binding sites in DNA sequences. *Nucl. Acids Res.* **31**:3576–3579. 2003.

[Ku99] Kurtz, S.: Reducing the Space Requirement of Suffix Trees. *Software—Practice and Experience*. **29**(13):1149–1171. 1999.

[MFG+03] Matys, V., Fricke, E., Geffers, R., Gößling, E., Haubrock, M., Hehl, R., Hornischer, K., Karas, D., Kel, A. E., Kel-Margoulis, O. V., Kloos, D.-U., Land, S., Lewicki-Potapov, B., Michael, H., Munch, R., Reuter, I., Rotert, S., Saxel, H., Scheer, M., Thiele, S., and Wingender, E.: TRANSFAC(R): transcriptional regulation, from patterns to profiles. *Nucl. Acids Res.* **31**(1):374–378. 2003.

[QFWW95] Quandt, K., Frech, K., Wingender, E., and Werner, T.: MatInd and MatInspector: new fast and versatile tools for detection of consensus matches in nucleotide data. *Nucl. Acids Res.* **23**:4878–4884. 1995.

[Ra03] Rahmann, S.: Dynamic programming algorithms for two statistical problems in computational biology. In: *Proc. of the 3rd Workshop of Algorithms in Bioinformatics (WABI)*. pp. 151–164. LNCS 2812, Springer Verlag. 2003.

[RMV03] Rahmann, S., Müller, T., and Vingron, M.: On the Power of Profiles for Transcription Factor Binding Site Detection. *Statistical Applications in Genetics and Molecular Biology*. **2**(1). 2003.

[SFA99] Scordis, P., Flower, D., and Attwood, T.: FingerPRINTScan: intelligent searching of the PRINTS motif database. *Bioinformatics*. **15**(10):799–806. 1999.

[St89] Staden, R.: Methods for calculating the probabilities for finding patterns in sequences. *Comp. Appl. Biosci.* **5**:89–96. 1989.

[WNB00] Wu, T., Nevill-Manning, C., and Brutlag, D.: Fast Probabilistic Analysis of Sequence Function using Scoring Matrices. *Bioinformatics*. **16**(3):233–244. 2000.