Fast Matching of Transcription Factor Motifs Using Generalized Position Weight Matrix Models

EMANUELE GIAQUINTA¹, SZYMON GRABOWSKI², and ESKO UKKONEN¹

ABSTRACT

The problem of finding the locations in DNA sequences that match a given motif describing the binding specificities of a transcription factor (TF) has many applications in computational biology. This problem has been extensively studied when the position weight matrix (PWM) model is used to represent motifs. We investigate it under the feature motif model, a generalization of the PWM model that does not assume independence between positions in the pattern while being compatible with the original PWM. We present a new method for finding the binding sites of a transcription factor in a DNA sequence when the feature motif model is used to describe transcription factor binding specificities. The experimental results on random and real data show that the search algorithm is fast in practice.

Key words: algorithms, automata, sequence analysis, strings.

1. INTRODUCTION

T HE BINDING OF TRANSCRIPTION FACTOR PROTEINS to their target DNA sequences is a fundamental process in the transcription of genetic information. Two main computational problems that arise from the study of these interactions are: i) finding an accurate model to describe the binding specificities of a given transcription factor; and ii) given a DNA sequence and a motif that describes the binding specificities of a given transcription factor, finding all the binding sites in the sequence that match the motif. The traditional model used to represent transcription factor motifs is the position weight matrix (PWM) (Stormo et al., 1982; Schneider et al., 1986; Gribskov et al., 1987). In this model, a motif of length *m* is represented using a matrix of size $m \times |\Sigma|$, where Σ is an alphabet of size 4 (DNA) in this context; each cell of the matrix contains the position-specific weights for each symbol in the alphabet, that is, the cell m_{ij} contains the weight contributed by an occurrence of the *j*-th symbol at the *i*-th position. This model assumes that there is no correlation between positions in the sites, that is, the contribution of a nucleotide at a given position to the total affinity does not depend on the other nucleotides that appear in other positions. The problem of matching the locations in DNA sequences at which a given transcription factor protein binds to is well studied under the PWM model (Pizzi and Ukkonen, 2008). One application where this problem becomes relevant is, for example, described in Hallikas et al. (2006).

Many more advanced models have been proposed to overcome the independence assumption of the PWM; among them are the maximal dependence decomposition (MDD) (Burge and Karlin, 1997), the generalized weight matrix (GWM) (Zhou and Liu, 2004), the permuted variable length Markov models

¹Department of Computer Science, University of Helsinki, Helsinki, Finland.

²Institute of Applied Computer Science, Lodz University of Technology, Lódź, Poland.

(PVLMM) (Zhao et al., 2005), feature motif model (FMM) (Sharon et al., 2008), the dinucleotide weight matrix (DWM) (Siddharthan, 2010), and the tree-based position weight matrix (TPWM) (Bi et al., 2011). While some of these models are not compatible with the original PWM representation, others are actually a generalized version of it. In this article, we deal with this latter type of model as we believe that they have a greater chance to replace the standard PWM model. In particular, we focus on the FMM (Sharon et al., 2008) since, to our knowledge, it is the most general one. However, our approach can also be applied, with small changes, to other similar models.

In the FMM model, the TF-binding specificities are described with so-called *features*, that is, rules that assign a weight to a set of associations between symbols and positions. In particular, a PWM can be expressed as a set of features that include a single association; in this respect, it is a special case of this more general model. Given a DNA sequence, a set of features and a motif length m, the matching problem consists of computing the score of each site (substring) of length m in the sequence, where the score of a site is the sum of the weights of all the features that occur in the site.

To our knowledge, the problem of matching transcription-factor motifs using the FMM model has not been investigated yet. In this work, we present a fast method to match FMM motifs. Our method is based on a reduction of this problem to the one of matching a set of gapped q-gram patterns in a given sequence, where a q-gram pattern is a sequence of q symbols, over a finite alphabet Σ , such that there is a gap of fixed length between each two consecutive symbols. In particular, we are interested in computing the list of matching patterns for each position in the sequence. This problem is a specific instance of the *variable length gaps problem* (VLG problem) for multiple patterns. Note that, although the number of associations in a feature in the FMM model can be up to m, the experimental evaluation presented in Sharon et al. (2008) allows for up to two associations. In this work, we focus on the case of two associations only, that is, the features are 2 grams with a fixed-length gap between the two symbols.

We then present a framework for the problem of matching FMM motifs that uses such an algorithm as a component and describes novel simple algorithms for the problem of matching a set of gapped q-gram patterns in the q = 2 case. The proposed algorithms are fast when the gap lengths in the patterns are constrained, which is a reasonable assumption in this context since the motifs are usually short. Our method is considerably faster than a naive search of all the features for each site in the sequence.

The rest of the article is organized as follows. In Section 2, we recall some preliminary notions and elementary facts. In Section 3, we present our method to find transcription-factor motifs using the FMM model. In Section 4, we present new algorithms for the problem of matching gapped q-gram patterns, which we use as a component in our method. In Section 5, we present experimental results on random and real data of our new algorithms and of the full method. Finally, we briefly discuss our conclusions in Section 6.

2. BASIC NOTIONS AND DEFINITIONS

Let $\Sigma = \{c_1, c_2, \dots, c_{\sigma}\}$ denote an alphabet of size σ . Let Σ^* denote the Kleene star of Σ , and Σ^m the set of all possible sequences of length *m* over Σ . |S| is the length of string *S*, S[i], $i \ge 0$, denotes its (i + 1)-th character, and $S[i, \dots, j]$ its substring between the (i + 1)-st and the (j + 1)-st characters (inclusive). A gapped q-gram pattern is of the form

$$(p_1 \cdot j_1 \cdot p_2 \cdot \ldots \cdot j_{q-1} \cdot p_q)$$

where $\langle p_1, p_2, \ldots, p_q \rangle$ is a sequence of q symbols and $\langle j_1, j_2, \ldots, j_{q-1} \rangle$ is the associated sequence of gaps, that is, $j_k \in N$ is the length of the gap between symbols p_k and p_{k+1} . We say that a gapped q-gram pattern $(p_1 \cdot j_1 \cdot p_2 \cdot \ldots \cdot j_{q-1} \cdot p_q)$ occurs in S at ending position i if

$$S[i-\sum_{i=k}^{q-1}(j_i+1)]=p_k,$$

for k = 1, ..., q. Given a set \mathcal{P} of gapped q-gram patterns, we denote with $g_{\text{span}}(\mathcal{P})$ the number of distinct gap lengths in \mathcal{P} .

The RAM model is assumed, with machine word size $w = \Omega(\log n)$. We use some bitwise operations following the standard notation as in C language: &, $|, \sim, \ll$ for and, or, not, and left shift, respectively. The function to compute the position of the most significant set bit of a word x is $|\log_2(x)|$.

3. METHOD

The FMM (Sharon et al., 2008) is a relatively new model to represent the interactions between transcription factors and their target DNA sequences. In the FMM model, the TF-binding specificities are described with so-called *features*, that is, rules that assign a weight to a set of associations between symbols and positions. Given a DNA sequence T, a set of features and a motif length m, the matching problem consists in computing the score of each site (substring) of length m in the sequence T, where the score of a site is the sum of the weights of all the features that occur in the site. Figure 1, taken from the Segal Lab website, shows an example of FMM motif, visualized as a sequence logo. The motif is of length 23 and consists of 20 2-gram features and 4×23 1-gram features. The 1-gram features taken together represent a PWM. The height of each letter gives the weight of the corresponding feature.

Formally, a *feature* can be denoted as

$$\{(a_1, i_1), \ldots, (a_a, i_a)\} \rightarrow w$$

where w is the affinity contribution of the feature and $a_j \in \{A, C, G, T\}$ is the nucleotide that must occur at position i_j , for j = 1, ..., q and $1 \le i_j \le m$. It is easy to transform these rules into new rules where the left side is a gapped q-gram pattern: if $i_1 < i_2 < ... < i_q$, we can induce the following gapped q-gram pattern rule

$$(a_1 \cdot (i_2 - i_1 - 1) \cdot \ldots \cdot (i_q, i_{q-1} - 1) \cdot a_q) \rightarrow (i_q, w).$$

This transformation has the advantage that the resulting pattern is position independent. Moreover, after this transformation, different features may share the same gapped q-gram pattern. For example, the features $\{(a, 1), (c, 3)\}$ and $\{(a, 3), (c, 5)\}$ map onto the same pattern $\{a \cdot 2 \cdot c\}$. Then, the matching problem can be decomposed into two components: the first component identifies the occurrences of the groups of features by searching for the corresponding gapped q-gram patterns while the second component computes the score for each candidate site using the information provided by the first component.

For a motif of length *m*, the second component can be easily implemented by maintaining the score for *m* sites simultaneously with a circular queue of length *m*. Each time a group of features with an associated set of position/weight pairs $\{(i_1, w_1), \ldots, (i_r, w_r)\}$ is found at position *j* in the sequence, the algorithm adds the weight w_k to the score of the alignment that ends at position $j + m - i_k$ in the sequence, if $j \ge i_k$.

4. ALGORITHMS

In this section, we present two simple algorithms to search for the occurrences of a set \mathcal{P} of gapped 2-gram patterns in a text T of length n, defined over a finite alphabet Σ of size σ . This problem is a specific instance of the VLG problem for multiple patterns. In the VLG problem, a pattern is a concatenation of strings and of variable length gaps. In the specific scenario of gapped q-gram patterns, where all the strings in the patterns have unitary length, the available more general algorithms for this problem do not scale well in the case of multiple patterns. In particular, to our knowledge, the best practical methods for the VLG problem are those introduced in Bille et al. (2010) and Haapasalo et al. (2011). In our problem, the first

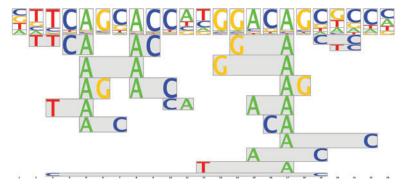


FIG. 1. Feature motif model (FMM) of the neuron-restrictive silencer factor (NRSF) protein.

algorithm has $O(n + \alpha)$ -time complexity, where α is the number of occurrences of the pattern strings (key words) in the text *T*. Note that the number of occurrences of a key word that occurs in *r* patterns and in *l* positions in the text is equal to $r \times l$. Instead, the second algorithm has $O(n + \alpha')$ -time complexity, where α' is the number of occurrences in the text of pattern prefixes ending with a key word. In the case of multiple patterns, these results are not ideal because, if we assume a uniform distribution for the patterns, we have on average that α and α' are $\Omega(n\frac{len(\mathcal{P})}{\alpha})$ and $\Omega(n\frac{|\mathcal{P}|}{\alpha})$, respectively, where $len(\mathcal{P})$ is the sum of the number of symbols in each pattern.

Our first algorithm for q = 2, based on the "Four Russians" technique (Arlazarov et al., 1970), has $O(n + ng_{\text{span}}(\mathcal{P})/h + occ)$ -time complexity, where $g_{\text{span}}(\mathcal{P})$ is the number of distinct gap lengths in \mathcal{P} , and h is a parameter. The second algorithm, based on bit-parallelism (Baeza-Yates and Gonnet, 1992), has $O(n[\ell/w] + occ)$ -time complexity, where $g_{\text{size}}(\mathcal{P}) \le \ell \le \sigma g_{\text{size}}(\mathcal{P})$ and $g_{\text{size}}(\mathcal{P})$ is the size of the variation range of the gap lengths in \mathcal{P} . The second algorithm is good when σ and $g_{\text{size}}(\mathcal{P})$ are small. Otherwise, the first algorithm is preferable.

4.1. The GQ1 algorithm

Given a set of gapped 2-gram patterns

$$\mathcal{P} = \{(a_1 \cdot j_1 \cdot b_1), (a_2 \cdot j_2 \cdot b_2), \dots, (a_l \cdot j_l \cdot b_l)\}$$

let $g_{\min}(\mathcal{P}) = \min \langle j_1, j_2, \dots, j_l \rangle$, $g_{\max}(\mathcal{P}) = \max \langle j_1, j_2, \dots, j_l \rangle$ and $g_{\text{size}}(\mathcal{P}) = g_{\max}(\mathcal{P}) - g_{\min}(\mathcal{P}) + 1$.

We assume that all the patterns in \mathcal{P} are distinct, that is, for any $c_1, c_2 \in \Sigma$, and $j \in N$, there exists at most one *i* such that $a_i = c_1$, $b_i = c_2$, and $j_i = j$. Let *G* be the set of all the distinct gap lengths in the patterns. For each $j \in G$, we maintain a lookup table (LUT) L_j of size σ^{2h} , where $1 \le h \le \lfloor \log n/\log \sigma \rfloor$ is a parameter. For any two strings S_1 and S_2 of length *h*, we have that $L_j(S_1, S_2)$ contains the list of all the pairs (i, k) such that

$$S_1[k] = a_i \wedge S_2[k] = b_i \wedge j = j_i$$

for $1 \le i \le l$ and $0 \le k < h$. For h = 1, the LUT definition can be simplified to

$$L_j(c_1, c_2) = \begin{cases} i & \text{if } c_1 = a_i \wedge c_2 = b_i \wedge j = j_i \\ 0 & \text{otherwise} \end{cases}$$

The first algorithm, named GQ1, works as follows: first, *T* is packed to *T'*, so that each symbol occupies only $[\log \sigma]$ bits. This phase is performed in O(n) time. For h = 1 it is possible to skip this pass and set T' = T. Let $T_i = T[i - h + 1..i]$ be the substring of *T* of length *h* ending at position *i*. The packed encoding of any such substring can be computed in constant time from the packed text using simple bitwise operations. Then, the algorithm scans *T'* in chunks of *h* symbols, for each position $i = h \times k - 1$, $1 \le k \le n/h$, and computes the list of matching patterns ending in one of the symbols of the substring T_i . To this end, it checks for each distinct gap length $j \in G$, the LUT entry $L_j(T_{i-j-1}, T_i)$, where T_{i-j-1} is the substring of length *h* at distance j + 1 from T_i . It traverses the associated list of patterns and reports each match in overall O(1) time. The algorithm for h = 1 is depicted in Figure 2.

The text scanning plus text packing is performed in $O(n + ng_{\text{span}}(\mathcal{P})/h + occ)$ time. Concerning the preprocessing time, for h = 1 we can build the LUT in time $O(\sigma^2 g_{\text{span}}(\mathcal{P}) + |(\mathcal{P})|)$. For h > 1, observe that

 $\begin{array}{l} \operatorname{GQ1} (\mathcal{P}, T) \\ 1. \ \operatorname{for} (a_i \cdot j_i \cdot b_i) \in \operatorname{do} \\ 2. \quad \overline{G}(b_i) \leftarrow \overline{G}(b_i) \cup \{j_i\} \\ 3. \quad L_{ji}(a_i, b_i) \leftarrow i \\ 4. \ \operatorname{for} i \leftarrow 0 \ \operatorname{to} |T| - 1 \ \operatorname{do} \\ 5. \quad \operatorname{for} j \in \overline{G}(T[i]) \\ 6. \qquad \text{if } i - 1 - j \ge 0 \ \text{and} \ L_j(T[i - 1 - j], T[i]) \neq 0 \\ 7. \qquad \operatorname{report}(T[i - 1 - j] \cdot j \cdot T[i]) \end{array}$

FIG. 2. The GQ1 algorithm, with h = 1, for the string-matching problem with gapped 2-gram patterns.

there are at most *h* matching patterns per slot of L_j , and they can be found in O(h) time if we build the LUT for h = 1 and use it to resolve in constant time membership queries on \mathcal{P} . Hence, the total time required to build the LUT is $O(\sigma^{2h}g_{\text{span}}(\mathcal{P})h + |\mathcal{P}|)$. The space in words required for the LUT is $O(\sigma^{2h}g_{\text{span}}(\mathcal{P})h)$. An alternative approach, useful when *h* is large, is to compute the LUT lazily. We initialize each slot of the LUT to 0. This is fast in practice, and in theory it can also be done in constant time and with no overhead in the space complexity using the method described in Navarro (2012). Then, when the algorithm accesses a slot that is set to 0, it computes the corresponding list of patterns as described before and updates the slot value.

It is also possible to improve the performance in the average case by computing an additional array \overline{G} of size σ^h , where the entry $\overline{G}(S)$, for $S \in \Sigma^h$, contains the list of gap lengths j, together with a pointer to the corresponding LUT, such that there exists $S' \in \Sigma^h : L_j(S', S) \neq \emptyset$. In this way, we can limit the iterations, for each position i, to all the gap lengths $j \in \overline{G}(T_i)$, which can be significantly less than the total number of gap lengths on average.

Note that since $h \ge 1$, this algorithm can be used only for $\sigma = o(n^{1/2})$. We can set *h* to an appropriate value below $\log n/(2\lceil \log \sigma \rceil)$, to obtain $o(ng_{\text{span}}(\mathcal{P})(\log n/\log \sigma))$ words of space. With *h* as specified above, the LUT update time complexity can be absorbed in the search time. Overall, the algorithm works in $O(n+ng_{\text{span}}(\mathcal{P})\log \sigma/\log n+occ)$ time in the worst case.

We note that in theory, a slightly better algorithm using similar ideas is possible, which can handle a few gaps at a time if their lengths are close to each other. This algorithm is not practical though, and thus we do not present it. In general, *L* may be a large structure, so in practice one chooses a limit *M* (in words) on its size and computes the maximum *h* that satisfies $\sigma^{2h} g_{\text{span}}(\mathcal{P})h \leq M$. For example, for $M = 2^{22}$, $g_{\text{span}}(\mathcal{P}) = 10$ and $\sigma = 4$, we can safely use h = 4. If *M* is increased to 2^{26} , then h = 5 may be used.

4.2. The GQ2 algorithm

We now present the second algorithm, GQ2. Let \mathcal{P}_a be the set of all gapped 2-gram patterns in \mathcal{P} with first symbol equal to *a*. Let also $GA(\mathcal{P}_a) = (Q, \Sigma, \delta, q_0, F)$ be the nondeterministic finite automaton for the language $\bigcup_{(a:j:b)\in\mathcal{P}_a} \Sigma^* a\Sigma^j b$ and defined as follows:

$$Q = \{q_0, q_1, q_2, \dots, q_{gmax}(\mathcal{P}) + 2\}$$

$$F = \{q_{gmax}(\mathcal{P}) + 2\}$$

$$\delta(q_i, c) = \begin{cases} \{q_0, q_1\} & \text{if } i = 0 \text{ and } c = a \\ \{q_0\} & \text{if } i = 0 \text{ and } c \neq a \\ \{q_{i+1}, q_{gmax}(\mathcal{P}_a) + 2\} & \text{if } (a \cdot i - 1 \cdot c) \in \mathcal{P}_a \text{ and } i \leq g_{max}(\mathcal{P}_a) \\ \{q_{gmax}(\mathcal{P}_a) + 2\} & \text{if } (a \cdot i - 1 \cdot c) \in \mathcal{P}_a \text{ and } i = g_{max}(\mathcal{P}_a) + 1 \\ \{q_{i+1}\} & \text{if } 1 \leq i \leq g_{max}(\mathcal{P}_a) \\ \emptyset & \text{otherwise} \end{cases}$$

The main idea is to run the automata $GA(\mathcal{P}_{c_1}), \ldots, GA(\mathcal{P}_{c_{\sigma}})$ in parallel over the text *T*, using bit-parallelism (Baeza-Yates and Gonnet, 1992).

For each automaton, we represent only the states $q_1, \ldots, q_{g_{\max}(\mathcal{P})+1}$. First, the algorithm encodes the subset of the states of $GA(\mathcal{P}_a)$ with an outgoing transition to the final state labeled by c, for each $c \in \Sigma$, in a set $B_a(c)$ defined as follows:

$$B_a(c) = \{ j \mid \delta(q_{j+1}, c) \cap F \neq \emptyset \}.$$

It is easy to see that the state q_{j+1} of $GA(\mathcal{P}_a)$ is active at position *i* in *T* iff T[i-1-j] = a, for $j=0, \ldots, g_{\max}(\mathcal{P})$. Hence, the algorithm encodes the configuration of $GA(\mathcal{P}_a)$ at position *i* in *T* in a set D_a^i defined as follows:

$$D_a^i = \{ 0 \le j \le g_{\max}(\mathcal{P}_a) \mid T[i-1-j] = a \}.$$

Given the sets $B_a(c)$ and D_a^i , for all $a, c \in \Sigma$, as defined above, it holds that a gapped 2-gram $(p_1 \cdot j \cdot p_2)$ has an occurrence at position i in T iff $p_2 = T[i]$ and $j \in D_{p_1}^i \cap B_{p_1}(p_2)$. The list of gapped 2-gram patterns occurring at position i can thus be computed by iterating over all the elements j in $D_a^i \cap B_a(T[i])$, for all $a \in \Sigma$. The set D_a^i can be computed by using the following recurrence:

$$D_a^i = \{ 0 \le j \le g_{\max}(\mathcal{P}_a) \mid (j=0 \text{ and } T[i-1]=a) \text{ or } j-1 \in D_a^{i-1} \}$$
(1)

Let $\mathtt{SiZe}(a) = g_{\max}(\mathcal{P}_a) + 1$, if $\mathcal{P}_a \neq \emptyset$, and to 0 otherwise. Let $\ell = \sum_{i=1}^{\sigma} \mathtt{SiZe}(c_i)$ and let also $\mathtt{POS}(c_k) = \sum_{i=0}^{k-1} \mathtt{SiZe}(c_i)$. The algorithm maintains the sets D_a^i packed in a bit-vector D_i of size ℓ . Each position $0 \le j \le g_{\max}(\mathcal{P}_a)$ in D_a^i is mapped onto $\mathtt{bit} j + \mathtt{POS}(a)$ in the corresponding bit-vector. For each $c \in \Sigma$, the sets $B_a(c)$ are packed in a bit-vector $\mathsf{B}(c)$ in the same way. Let I be a bit-vector of size ℓ such that bit k is set iff $\mathtt{pos}(a) = k$ for some $a \in \Sigma$. Then, the vector D can be maintained, based on recurrence 1, using the following bitwise operations:

$$D_i \leftarrow ((D_{i-1} \ll 1) \& \sim I)|F$$

where $F = 1 \ll p \circ S(T[i-1])$, if Size(T[i-1]) > 0, and to 0 otherwise. Since the value of D_a^i depends only on D_a^{i-1} , we can maintain one vector only and update it using the above formula. To report all the occurrences of the patterns at a given position *i* in *T*, the algorithm iterates over all the bits set in D & B(T[i]) by repetitively computing the index of the highest bit set and masking it until there are no more bits set. Given a bit index *k*, it is not hard to see that the corresponding gap is equal to $k - \max\{p \circ S(a) \le k, a \in \Sigma\}$. This translates into computing the index of the highest bit set in I & $O^{m-k-1}1^{k+1}$. Hence, for each bit *k* set in D & B(T[i]), the algorithm reports an occurrence of $(T[i-1-j] \cdot j \cdot T[i])$, where $j = k - \lfloor \log_2(I \otimes O^{m-k-1}1^{k+1}) \rfloor$. The algorithm is depicted in Figure 3.

Note that $g_{\max}(\mathcal{P}) \leq \ell \leq \sigma g_{\max}(\mathcal{P})$. However, it is easy to replace the $g_{\max}(\mathcal{P})$ term with $g_{\text{size}}(\mathcal{P})$: we reduce the size of the bit-vectors of each automaton $GA(\mathcal{P}_a)$ by $g_{\min}(\mathcal{P})$ bits by representing only the states $q_{g\min}(\mathcal{P})_{+1}, \ldots, q_{g\max}(\mathcal{P}_a)_{+1}$ (we decided not to include this improvement into the description for simplicity). For each position in the text, the bitwise operations needed to update the D vector and report the occurrences have to access $\lfloor \ell/w \rfloor$ words. Hence, the time complexity of the searching phase of the algorithm is

$\operatorname{GQ2}(\mathcal{P},T)$
PREPROCESSING
8. for $c \in \Sigma$ do $g_{\max}[c] \leftarrow 0$
9. for $(a \cdot j \cdot b) \in \mathcal{P}$ do
10. if $g_{\max}[a] < j + 1$ then
11. $g_{\max}[a] \leftarrow j+1$
12. $m \leftarrow 0$
13. for $c \in \Sigma$ do
14. $m \leftarrow m + g_{\max}[c]$
15. $k \leftarrow 0, I \leftarrow 0^m$
16. for $c \in \Sigma$ do
17. $B[c] \leftarrow 0^m, F[c] \leftarrow 0^m$
18. if $g_{\max}[c] > 0$ then
19. $F[c] \leftarrow 1 \ll k$
20. $k \leftarrow k + g_{\max}[c]$
21. $I \leftarrow I \mid F[c]$
22. for $(a \cdot j \cdot b) \in \mathcal{P}$ do
23. $B[b] \leftarrow B[b] \mid (F[a] \ll j)$
SEARCHING
24. $D \leftarrow 0^m$
25. for $i \leftarrow 0$ to $ T - 1$ do
26. $M \leftarrow D \& B[T[i]]$
27. while $M \neq 0^m$ do
$28. \qquad k \leftarrow \lfloor \log_2(M) \rfloor$
29. $j \leftarrow k - \lfloor \log_2(I \& 0^{m-k-1} 1^{k+1}) \rfloor$
30. report ($T[i-1-j] \cdot j \cdot T[i]$)
31. $M \leftarrow M \& \sim (1 \ll k)$
32. $D \leftarrow ((D \ll 1) \& \sim I) \mid F[T[i]]$

FIG. 3. The GQ2 algorithm for the string-matching problem with gapped 2-gram patterns.

 $O(n[\ell/w] + occ)$, where *occ* is the number of occurrences of the patterns in the text. The space complexity is $O(\sigma[\ell/w])$.

5. RESULTS

In this section, we evaluate the performance of our method. All the algorithms, except PMA, whose source code was kindly provided by the authors, have been implemented in the C++ programming language and compiled with the GNU C++ Compiler 4.4, using the options -03. The test machine was a 3 GHz Intel Core 2 Quad running Ubuntu 10.04, and running times were measured with the getrusage function. We performed two experiments to assess the performance of our motif-matching tool and of our algorithms for gapped q-gram patterns, respectively.

The first experiment consisted in running our motif finder, named FMF (FMM-motif-finder), with real FMM motifs on a human genome chromosome of 249, 250, 621 base pairs. Note that the particular type of sequence is not relevant for the benchmark. The motifs used were downloaded from the Segal Lab website. These motifs include features that span one and two positions. The features that span one position are basically a PWM matrix and are naively matched since optimizations like the lookahead technique cannot be applied in this more general model (see Pizzi and Ukkonen, 2008, for a survey on PWM matching algorithms). For long motifs (>20) we use the superalphabet technique described in Pizzi et al. (2011). The features that span two positions are matched using the GQ2 algorithm if $g_{size}(\mathcal{P})$ is small (which is always the case for short motifs, since $g_{size}(\mathcal{P})$ can be at most equal to the motif length), and with GQ1 otherwise. For a motif of length *m* the program computes the total score of each substring of length *m* of the sequence. Table 1 lists the running times for different motifs and also the corresponding motif length and the number of features that span two positions. The table also lists the running time of a naive scan in which all the features that span two positions are matched naively, that is, each site is checked for all the features. The results show that our method is fast and is thus suitable in practice, and about three times faster than a naive approach. To show the overhead due to searching for two-position features and also for naively matching the PWM matrix, we list in Table 2 the running times of the MOODS software (Korhonen et al., 2009), a suite of algorithms for PWM matching, to search for different PWM motifs downloaded from the JASPAR database. The results show that PWM matching is faster by a factor between 1.4 and 2, which, although not negligible, is an acceptable trade-off for dealing with the more complex FMM model.

Motif	Motif length	Features	Time	Naive alg. time
NRSF	23	20	10.04	32.75
CTCF	19	20	9.26	30.47
STAT1_U	17	20	9.15	27.77
E2F4_Boyer	14	15	7.96	22.39
OCT4_Boyer	15	15	9.45	25.60
SOX2_Boyer	12	16	8.30	24.56

 TABLE 1.
 Running Times in Seconds of FMM Motif Finder on a Human Genome Chromosome of 225, 280, 621 Base Pairs for Different Motifs

Table 2. Running Times in Seconds of the MOODS Software on a Human Genome Chromosome of 225, 280, 621 Base Pairs for Different PWM Motifs with a p-value of 10^{-2}

Motif	Motif length	Time
Ar	22	7.83
REST	19	8.96
NR1H2	17	7.35
RXRA::VDR	15	7.03
br_Z1	14	5.80
dl_1	12	4.54

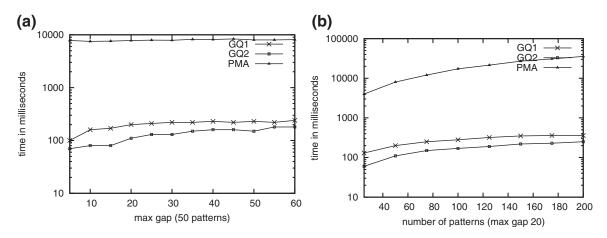


FIG. 4. Experimental results on a genome sequence of *Escherichia coli* with randomly generated gapped 2-gram patterns obtained for (a) varying gap interval with a set of 50 patterns and (b) varying number of patterns with maximum gap 20.

In the second experiment, we compared the GQ1 and GQ2 algorithms with the PMA algorithm (Haapasalo et al., 2011), which is one of the best practical methods for the VLG problem. The experiment consisted in searching for a set of randomly generated gapped q-gram patterns on the genome sequence of 4, 638, 690 base pairs of Escherichia coli. Figure 4a shows the running times for searching a set of randomly generated 2-gram patterns with a fixed number of patterns equal to 50 and such that the maximum gap varies between 5 and 60. Figure 4b shows the running times for searching a set of randomly generated 2-gram patterns with a fixed maximum gap of 20 and such that the number of patterns varies between 25 and 200. We used a logarithmic scale on the y axis. The experimental results show that the new algorithms are significantly faster (up to 50 times) than the PMA algorithm in this particular scenario. The approach based on locating the occurrences of the key words does not scale well when all the key words, and in particular, the first key word, are of length 1. Note that in the general case of arbitrary length key words, the PMA algorithm is very fast (Haapasalo et al., 2011). In all the results, the GQ2 algorithm is slightly faster than the GQ1 algorithm. This is expected because in all the experiments $g_{\text{span}}(\mathcal{P})$ is almost equal to $g_{\text{size}}(\mathcal{P})$, and thus the word-level parallelism pays off. To show the difference between the two algorithms for a sparse set of gaps, we performed another experiment in which the number of patterns is fixed at 25 and the maximum gap varies between 50 and 150, so that, for increasing values of the maximum gap, $g_{\text{span}}(\mathcal{P})$ is always 25 at most while

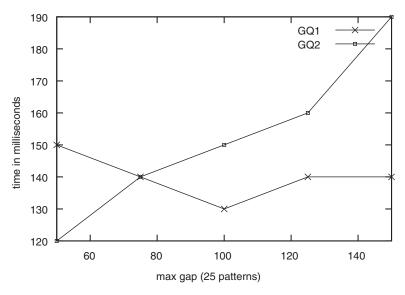


FIG. 5. Comparison of the GQ1 and GQ2 algorithms for a sparse set of gaps.

FAST MATCHING OF TF MOTIFS USING GENERALIZED PWM MODELS

 $g_{\text{size}}(\mathcal{P})$ increases. The results, depicted in Figure 5, show that in this scenario the GQ1 algorithm is faster than the GQ2 algorithm when the maximum gap is at least four times the number of patterns.

6. CONCLUSIONS

We presented a method for the problem of finding all the locations in DNA sequences that match a motif describing transcription factor binding sites under the feature motif model. Our approach is based on reducing the problem to the one of matching a set of gapped q-gram patterns, where a gapped q-gram pattern is a sequence of q symbols such that there is a gap of fixed length between each two consecutive symbols. To this end, we presented novel algorithms for this problem in the q = 2 case, which we believe could be of independent interest. We note that a more general but slower algorithm that works for any q is possible using a similar approach. The experimental results show that our method achieves very good performance and is thus effective. It is worth observing that any PWM motif can be expressed in the FMM model, so there is no drawback in the use of this model.

7. ACKNOWLEDGMENTS

This work was supported by the Academy of Finland, grant 118653 (ALGODAN), and by the Polish Ministry of Science and Higher Education, project N N516 441938.

8. DISCLOSURE STATEMENT

The authors declare that no competing financial interests exist.

REFERENCES

Arlazarov, V., Dinic, E., Kronrod, M., and Faradzev, I. 1970. On economical construction of the transitive closure of a directed graph. Soviet Math. Doklady 11, 1209–1210.

Baeza-Yates, R.A., and Gonnet, G.H. 1992. A new approach to text searching. Commun. ACM 35, 74-82.

- Bi, Y., Kim, H., Gupta, R., and Davuluri, R.V. 2011. Tree-based position weight matrix approach to model transcription factor binding site profiles. *PLoS ONE* 6.
- Bille, P., Gørtz, I.L., Vildhøj, H.W., and Wind, D.K. 2010. String matching with variable length gaps, 385–394. *In* Chávez, E., and Lonardi, S. ed. *SPIRE*, Vol. 6393, *Lecture Notes in Computer Science*, Springer, New York.
- Burge, C., and Karlin, S. 1997. Prediction of complete gene structures in human genomic DNA. J of Mol Biol 268, 78-94.
- Gribskov, M., McLachlan, A.D., and Eisenberg, D. 1987. Profile analysis: detection of distantly related proteins. *Proceedings of the National Academy of Sciences* 84, 4355–4358.
- Haapasalo, T., Silvasti, P., Sippu, S., and Soisalon-Soininen, E. 2011. Online dictionary matching with variable-length gaps, 76–87. *In* Pardalos, P.M., and Rebennack, S. ed. *SEA*, Vol. 6630, *Lecture Notes in Computer Science*, Springer, New York.
- Hallikas, O., Palin, K., Sinjushina, N., et al. 2006. Genome-wide prediction of mammalian enhancers based on analysis of transcription-factor binding affinity. *Cell* 124, 47–59.
- Korhonen, J., Martinmäki, P., Pizzi, C., et al. 2009. MOODS: fast search for position weight matrix matches in DNA sequences. *Bioinformatics* 25, 3181–3182.

Navarro, G. 2012. Constant-time array initialization in little space. In *Proc. XXXI International Conference of the Chilean Computer Science Society (SCCC).* IEEE CS Press. In press.

- Pizzi C., and Ukkonen, E. 2008. Fast profile matching algorithms a survey. Theor. Comput. Sci. 395, 137-157.
- Pizzi, C., Rastas, P., and Ukkonen, E. 2011. Finding significant matches of position weight matrices in linear time. *IEEE/ACM Trans. Comput. Biology Bioinform.* 8, 69–79.
- Schneider, T.D., Stormo, G.D., Gold, L., and Ehrenfeucht, A. 1986. Information content of binding sites on nucleotide sequences. J Mol. Biol. 188, 415–431.
- Sharon, E., Lubliner, S., and Segal, E. 2008. A feature-based approach to modeling protein-DNA interactions. *PLoS Comp. Biol.* 4.

- Stormo, G.D., Shneider, T.D., Gold, L., and Ehrenfeucht, A. 1982. Use of 'perceptron' algorithm to distinguish translational initiation sites in *E. coli. Nucleic Acids Research* 10, 2997–3011.
- Zhao, X., Huang, H., and Speed, T.P. 2005. Finding short DNA motifs using permuted Markov models. *J Comp. Bio.* 12, 894–906.
- Zhou, Q., and Liu, J.S. 2004. Modeling within-motif dependence for transcription factor binding site predictions. *Bioinformatics* 20, 909–916.

Address correspondence to: Dr. Emanuele Giaquinta Department of Computer Science University of Helsinki P.O. Box 68 (Gustaf Hällströmin katu 2b) 00014 Helsinki Finland

E-mail: emanuele.giaquinta@cs.helsinki.fi