Motif Matching Using Gapped Patterns

Emanuele Giaquinta\(^1\),⋆, Kimmo Fredriksson\(^2\),
Szymon Grabowski\(^3\), and Esko Ukkonen\(^1\)

\(^1\) Department of Computer Science, University of Helsinki, Finland
{emanuele.giaquinta,ukkonen}@cs.helsinki.fi
\(^2\) School of Computing, University of Eastern Finland, P.O. Box 1627, FI-70211 Kuopio, Finland
kimmo.fredriksson@uef.fi
\(^3\) Institute of Applied Computer Science, Lodz University of Technology, Al. Politechniki 11, 90–924 Łódź, Poland
sgrabow@kis.p.lodz.pl

1 Introduction and Basic Definitions

We consider the problem of matching a set \(\mathcal{P}\) of gapped patterns against a given text \(T\) of length \(n\), where a gapped pattern is a sequence of strings (keywords), over a finite alphabet \(\Sigma\) of size \(\sigma\), such that there is a gap of fixed length between each two consecutive strings. We assume the RAM model, with words of size \(w\) in bits. We are interested in computing the list of matching patterns for each position in the text. This problem is a specific instance of the \textit{Variable Length Gaps} problem \cite{Fredriksson2004} (VLG problem) for multiple patterns and has applications in the discovery of transcription factor (TF) binding sites in DNA sequences when using generalized versions of the Position Weight Matrix (PWM) model to represent TF binding specificities. The paper \cite{Grabowski2012} describes how a motif represented as a generalized PWM can be matched as a set of gapped patterns with unit-length keywords, and presents algorithms for the restricted case of patterns with two unit-length keywords.

In the VLG problem a pattern is a concatenation of strings and of variable-length gaps. The best time bounds for this problem are: i) \(O(n(k \log \sigma + \log \sigma))\) \cite{Fredriksson2004}, where \(k\) is the number of the strings and gaps in the pattern; ii) \(O(n \log \sigma + \alpha)\) \cite{Fredriksson2004}, where \(\alpha\) is the total number of occurrences of the strings in the patterns within the text\(^1\); iii) \(O(n(\log \sigma + K) + \alpha')\) \cite{Giaquinta2010}, where \(K\) is the maximum number of suffixes of a keyword that are also keywords and \(\alpha'\) is the number of text occurrences of pattern prefixes that end with a keyword. Recently, a variant of this algorithm based on word-level parallelism was presented in \cite{Giaquinta2013}. Let \(\text{len}(\mathcal{P})\) be the total number of alphabet symbols in the patterns. If all the keywords have unit length, the last two results are not ideal because in this case \(\alpha\) and \(\alpha'\) are \(\Omega(n \frac{\text{len}(\mathcal{P})}{\sigma})\) and \(\Omega(n \frac{\text{len}(\mathcal{P})}{\sigma})\) on average, respectively, if we assume that the symbols in the patterns are sampled from \(\Sigma\) according to a uniform distribution. When \(\alpha\) or \(\alpha'\) is large, the bound of \(\cite{Fredriksson2004}\) may be preferable. The drawback of this algorithm is that, to our knowledge, it is not practical.

\(^{\ast}\) Supported by the Academy of Finland, grant 118653 (ALGODAN).

\(^{1}\) Note that the number of occurrences of a keyword that occurs in \(r\) patterns and in \(l\) positions in the text is equal to \(r \times l\).

T. Lecroq and L. Mouchard (Eds.): IWOCA 2013, LNCS 8288, pp. 448–452, 2013.
© Springer-Verlag Berlin Heidelberg 2013
In this paper we present the following novel result, based on dynamic programming and word-level parallelism:

**Theorem 1.** Given a set $\mathcal{P}$ of gapped patterns and a text $T$ of length $n$, all the occurrences in $T$ of the patterns in $\mathcal{P}$ can be reported in time $O(n(\log \sigma + \log^2 g_{size}(\mathcal{P})[k-len(\mathcal{P})/w]) + occ)$

where $g_{size}(\mathcal{P})$ is the size of the variation range of the gap lengths, $k-len(\mathcal{P})$ is the total number of keywords in the patterns and $occ$ is the number of occurrences of the patterns in the text. Note that in the case of unit-length keywords we have $k-len(\mathcal{P}) = \text{len}(\mathcal{P})$. The proposed algorithm obtains a bound similar to the one of [3], in the restricted case of fixed-length gaps. In particular, it is a moderate improvement for $\log g_{size}(\mathcal{P}) = O(\sqrt{\log w})$. Moreover, it is also practical. For this reason, it provides an effective alternative when $\alpha$ or $\alpha'$ is large. For more details on the motivation, one additional result and an experimental evaluation of the algorithms against the state of the art see [4]. The proposed algorithms are fast in practice, and preferable if all the strings in the patterns have unit length.

Let $\Sigma^*$ denote the set of all possible sequences over $\Sigma$. \(|S|\) is the length of string $S$, $S[i], i \geq 0$, denotes its \((i+1)\)-th character, and $S[i \ldots j]$ denotes its substring ranging from $i$ to $j$. For any two strings $S$ and $S'$, we say that $S'$ is a suffix of $S$ (in symbols, $S' \supseteq S$) if $S' = S[i \ldots |S| - 1]$, for some $0 \leq i < |S|$. A gapped pattern $P$ is of the form $S_1 \cdot j_1 \cdot S_2 \cdot \ldots \cdot j_{\ell-1} \cdot S_\ell$, where $S_i \in \Sigma^*$, $|S_i| \geq 1$, is the $i$-th string (keyword) and $j_i \geq 0$ is the length of the gap between keywords $S_i$ and $S_{i+1}$, for $i = 1, \ldots, \ell$. We say that $P$ occurs in a string $T$ at ending position $i$ if $T[i-m+1 \ldots i] = S_1 \cdot A_1 \cdot S_2 \cdot \ldots \cdot A_{\ell-1} \cdot S_\ell$, where $A_i \in \Sigma^*$, $|A_i| = j_i$, for $1 \leq i \leq \ell - 1$, and $m = \sum_{i=1}^{\ell} |S_i| + \sum_{i=1}^{\ell-1} j_i$. In this case we write $P \sqsupseteq_g T_i$. We denote by $k-len(P)$ the length of the keywords in $P$. The gapped pattern $P_i = S_1 \cdot j_1 \cdot S_2 \cdot \ldots \cdot j_{i-1} \cdot S_i$ is the prefix of $P$ of length $i \leq \ell$.

We use some bitwise operations following the standard C language notation: $\&$, $\mid$, $\sim$, $\ll$ for and, or, not and left shift, respectively. The position of the most significant non-zero bit of a word $x$ is equal to $\lfloor \log_2(x) \rfloor$.

## 2 Online Algorithm for Matching a Set of Gapped Patterns

Let $P_k$ be the $k$-th pattern in $\mathcal{P}$. We adopt the superscript notation for $S_i$, $j_i$, and $P_i$ with the same meaning. We define the set

$$D_i = \{(k, l) \mid P_k^l \sqsupseteq_g T_i\},$$

of the prefixes of the patterns that occur at position $i$ in $T$, for $i = 0, \ldots, n - 1$, $1 \leq k \leq |\mathcal{P}|$ and $1 \leq l \leq k-len(P_k^l)$. From the definition of $D_i$ it follows that the pattern $P_k^l$ occurs in $T$ at position $i$ if and only if $(k, k-len(P_k)) \in D_i$. Let $\mathcal{K} = \{1, \ldots, k-len(\mathcal{P})\}$ be the set of indices of the keywords in $\mathcal{P}$ and let $T_i \subseteq \mathcal{K}$ be the set of indices of the matching keywords in $T$ ending at position $i$. The sequence $T_i$, for $0 \leq i < n$, is basically a new text with character classes over
respectively. We also denote with bit corresponding to the element \((i, \ell \in \ell)\) as bit-vectors of \(k\)-len(\(P\)) terms. The sets \(D_i\) can be computed using the following lemma:

**Lemma 1.** Let \(P\) and \(T\) be a set of gapped patterns and a text of length \(n\), respectively. Then \((k, l) \in D_i\), for \(1 \leq k \leq |P|\), \(1 \leq l \leq k\)-len(\(P^k\)) and \(i = 0, \ldots, n - 1\), if and only if

\[
(l = 1 \text{ or } (k, l - 1) \in D_{i-1-j^k_{i-1}}) \text{ and } S^k_i \in T_i.
\]

The idea is to match the transformed patterns against the text \(T\). Let \(g_{\min}(P)\) and \(g_{\max}(P)\) denote the minimum and maximum gap length in the patterns, respectively. We also denote with \(g_{\size}(P) = g_{\max}(P) - g_{\min}(P) + 1\) the size of the variation range of the gap lengths. For each position \(i\) in \(T\), the main steps of the algorithm are i) compute the set \(T_i\) in \(O(\log \sigma)\) time using the Aho-Corasick (AC) automaton \([1]\) for the set of distinct keywords in \(P\); ii) compute the set \(D_i\), using Lemma 1 and word-level parallelism, in time \(O(g_{\text{w-span}}[k\text{-len}(P)/w])\), where \(1 \leq g_{\text{w-span}} \leq w\) is the maximum number of distinct gap lengths that span a single word in our encoding. We also describe how to obtain an equivalent set of patterns with \(O(\log g_{\size}(P))\) distinct gap lengths at the price of \(O(\log g_{\size}(P))\) new keywords per gap, thus achieving \(O(\log^2 g_{\size}(P)[k\text{-len}(P)/w])\) time.

Let \(Q\) denote the set of states of the AC automaton, root the initial state and \(\text{label}(q)\) the string which labels the path from state root to \(q\), for any \(q \in Q\). The transition function \(\delta(q, c)\) is defined as the unique state \(q'\) such that \(\text{label}(q')\) is the longest suffix of \(\text{label}(q)\) · \(c\). We also store for each state \(q\) a pointer \(f_o(q)\) to the state \(q'\) such that \(\text{label}(q')\) is the longest suffix of \(\text{label}(q)\) that is also a keyword, if any. Let

\[
B(q) = \{(k, l) \mid S^k_i \supseteq \text{label}(q)\},
\]

be the set of all the occurrences of keywords in the patterns in \(P\) that are suffixes of \(\text{label}(q)\), for any \(q \in Q\). We preprocess \(B(q)\) for each state \(q\) such that \(\text{label}(q)\) is a keyword and compute it for any other state using \(B(f_o(q))\). Let \(G\) be the set of all the distinct gap lengths in the patterns. In addition to the sets \(B(q)\), we preprocess also a set \(C(g)\), for each \(g \in G\), defined as follows:

\[
C(g) = \{(k, l) \mid j^k_l = g\},
\]

for \(1 \leq k \leq |P|\) and \(1 \leq l < k\)-len(\(P^k\)). We encode the sets \(D_i, B(q)\) and \(C(g)\) as bit-vectors of \(k\)-len(\(P^k\)) bits. The generic element \((k, l)\) is mapped onto bit \(\sum_{i=1}^{k-1} k\)-len(\(P^i\)) + k\)-len(\(P_{i-1}\)), where k\)-len(\(P^k\)) = 0 for any \(k\). We denote with \(D_i, B(q)\) and \(C(g)\) the bit-vectors representing the sets \(D_i, B(q)\) and \(C(g)\), respectively. We also compute two additional bit-vectors \(I\) and \(M\), such that the bit corresponding to the element \((k, 1)\) in \(I\) and \((k, k\)-len(\(P^k\))) in \(M\) is set to 1, for \(1 \leq k \leq |P|\). We basically mark the first and the last bit of each pattern, respectively. Let \(H_i\) be the bit-vector equal to the bitwise \(\text{or}\) of the bit-vectors

\[
D_{i-1-g} \& C(g),
\]

(1)
for each $g \in G$. Then the corresponding set $H_i$ is equal to

$$\bigcup_{g \in G} \{(k, l) \mid (k, l) \in D_{i-1-g} \land \bar{b}^k_l = g\}.$$  

Let $q_{i-1} = \text{root}$ and $q_i = \delta(q_{i-1}, T[i])$ be the state of the AC automaton after reading symbol $T[i]$. It is not hard to see that $B(f_o(q_i))$ encodes the set $T_i$. The bit-vector $D_i$ can then be computed using the following bitwise operations:

$$D_i \leftarrow (|H_i| \ll 1) \land B(f_o(q_i))$$  

which correspond to the relation

$$\{(k, l) \mid (l = 1 \lor (k, l-1) \in H_i) \land (k, l) \in B(f_o(q_i))\}.$$  

To report all the patterns that match at position $i$ it is enough to iterate over all the bits set in $D_i \& M$. The algorithm, named \textsc{gq-matcher}, is given in Figure 1.

The bit-vector $H_i$ can be constructed in time $O(g_{w\text{-span}}[\text{k-len}(P)/w])$, $1 \leq g_{w\text{-span}} \leq w$, as follows: we compute Equation 1 for each word of the bit-vector separately, starting from the least significant one. For a given word with index $j$, we have to compute equation 1 only for each $g \in G$ such that the $j$-th word of $C(g)$ has at least one bit set. Each position in the bit-vector is spanned by exactly one gap, so the number of such $g$ is at most $w$. Hence, if we maintain, for each word index $j$, the list $G_j$ of all the distinct gap lengths that span the corresponding positions, we can compute $H_i$ in time $\sum_{j=1}^{[\text{k-len}(P)/w]} |G_j|$, which yields the advertised bound by replacing $|G_j|$ with $g_{w\text{-span}} = \max_j |G_j|$.

The time complexity of the searching phase of the algorithm is then $O(n(\log \sigma + g_{w\text{-span}}[\text{k-len}(P)/w]) + \text{occ})$, while the space complexity is $O(\text{len}(P) + (g_{\text{max}}(P) + \text{k-len}(P)))[\text{k-len}(P)/w])$. 

\textbf{Fig. 1.} The gq-matcher algorithm for the string matching problem with gapped patterns.

\begin{verbatim}
GQ-MATCHER-preprocess (P, T)
1. (δ, root, B, fo) ← AC(P)
2. G ← ∅
3. m ← k-len(P)
4. l ← 0, M ← 0
5. for g = 0, ..., gmax(P) do C(g) ← 0
6. l ← 0
7. for S1 · j1 · S2 · · · · jℓ · St ∈ P do
8.  l ← l | 1 ≪ l
9.  for k = 1, ..., ℓ do
10.   if k = ℓ then
11.      M ← M | 1 ≪ ℓ
12.      else g ← jk + |Sk+1| − 1
13.      C(g) ← C(g) | 1 ≪ ℓ
14.      G ← G ∪ {g}
15.  l ← l + 1

GQ-MATCHER-search (P, T)
1. q ← root
2. for i = 0, ..., |T| − 1 do
3.   q ← δ(q, T[i]), H ← 0
4.   for g ∈ G do
5.     H ← H | (D_{i−1−g} & C(g))
6.     D_i ← (H ≪ 1) | 1) & B(f_o(q))
7.   H ← D_i & M
8.  REPORT(H)

REPORT(H)
1. while H ≠ 0 do
2.   k ← [log\text{\textunderscore}2(H)]
3.   report(k)
4.   H ← H ≪ (1 ≤ k)
\end{verbatim}
The gq-matcher algorithm is preferable only when \( g_{\text{w-span}} \ll w \). We now show how to improve the time complexity in the worst-case by constructing an equivalent set of patterns with \( O(\log g_{\text{size}}(P)) \) distinct gap lengths. W.l.o.g. we assume that \( g_{\min}(P) \) and \( g_{\max}(P) \) are a power of two (if they are not we round them down and up, respectively, to the nearest power of two). Let \( \text{lsb}(n) \) be the bit position of the least significant bit set in the binary encoding of \( n \), for \( n \geq 1 \).

Observe that, for any positive \( g \in G \), the minimum and maximum value for \( \text{lsb}(g) \) are \( \log g_{\min}(P) \) and \( \log g_{\max}(P) \), respectively, and the number of bits set in the binary encoding of \( g \) is \( O(\log g_{\text{size}}(P)) \). Let also \( G' = \{0\} \cup \{2^i \mid \log g_{\min}(P) \leq i \leq \log g_{\max}(P)\} \). We augment the alphabet \( \Sigma \) with a wildcard symbol \( * \) that matches any symbol of the original alphabet and define by recursion the function

\[
\phi(g) = \begin{cases} 
  g & \text{if } g \in G' \\
  (2^{\text{lsb}(g)} - 1) \cdot * \cdot \phi(g - 2^{\text{lsb}(g)}) & \text{otherwise}
\end{cases}
\]

that maps a gap length \( g \) onto a concatenation of \( l \) gap lengths and \( l-1 \) wildcard symbols, where \( l \) is the number of bits set in the binary encoding of \( g \) if \( g \) is positive or 1 otherwise. By definition, all the gaps in the resulting sequence belong to the set \( G'' = G' \cup \{2^i - 1 \mid \log g_{\min}(P) \leq i < \log g_{\max}(P)\} \). We generate a new set of patterns \( P' \) from \( P \), by transforming each pattern \( S_{\bar{1}} \cdot j_1 \cdot S_{\bar{2}} \cdot \ldots \cdot j_{l-1} \cdot S_{\bar{l}} \) in \( P \) into the equivalent pattern \( S_{\bar{1}} \cdot \phi(j_1) \cdot S_{\bar{2}} \cdot \ldots \cdot \phi(j_{l-1}) \cdot S_{\bar{l}} \). Observe that extending the algorithm presented above to handle wildcard symbols is straightforward. By definition of \( \phi \) we have that \( \text{k-len}(P') < \log g_{\text{size}}(P) \cdot \text{k-len}(P) \), since the number of gaps that are split is at most \( \text{k-len}(P) - |P| \) and the number of wildcard symbols that are added per gap is at most \( \log g_{\text{size}}(P) \). The number of words needed for a bit-vector is then \( \lceil \log g_{\text{size}}(P) \text{k-len}(P)/w \rceil \leq \log g_{\text{size}}(P) \cdot \text{k-len}(P)/w \). In this way we obtain an equivalent set of patterns such that the set \( G \) of distinct gap lengths is contained in \( G'' \) and so its cardinality is \( O(\log g_{\text{size}}(P)) \).

References

1. Aho, A.V., Corasick, M.J.: Efficient string matching: An aid to bibliographic search. Commun. ACM 18(6), 333–340 (1975)
2. Bille, P., Gertz, I.L., Vildhøj, H.W., Wind, D.K.: String matching with variable length gaps. Theor. Comput. Sci. 443, 25–34 (2012)
3. Bille, P., Thorup, M.: Regular expression matching with multi-strings and intervals. In: Charikar, M. (ed.) SODA, pp. 1297–1308. SIAM (2010)
4. Giaquinta, E., Fredriksson, K., Grabowski, S., Tomescu, A.I., Ukkonen, E.: Motif matching using gapped patterns. CoRR abs/1306.2483 (2013)
5. Giaquinta, E., Grabowski, S., Ukkonen, E.: Fast matching of transcription factor motifs using generalized position weight matrix models. Journal of Computational Biology 20(9), 1–10 (2013)
6. Haapasalo, T., Silvasti, P., Sippu, S., Soisalon-Soininen, E.: Online dictionary matching with variable-length gaps. In: Pardalos, P.M., Rebennack, S. (eds.) SEA 2011. LNCS, vol. 6630, pp. 76–87. Springer, Heidelberg (2011)
7. Sippu, S., Soisalon-Soininen, E.: Online matching of multiple regular patterns with gaps and character classes. In: Dediu, A.-H., Martín-Vide, C., Truthe, B. (eds.) LATA 2013. LNCS, vol. 7810, pp. 523–534. Springer, Heidelberg (2013)