An Efficient Algorithm for Approximate Pattern Matching with Swaps

Matteo Campanelli[2] Domenico Cantone[1] Simone Faro[1]
Emanuele Giaquinta[1]

Department of Mathematics and Computer Science, University of Catania, Italy

Scuola Superiore di Catania, University of Catania, Italy

Pattern Matching with Swaps

A *swap permutation* for a string P of length m is a permutation $\pi : \{0, ..., m-1\} \rightarrow \{0, ..., m-1\}$ such that:

(a) if $\pi(i) = j$ then $\pi(j) = i$ (characters at positions i and j are swapped);

(b) for all i, $\pi(i) \in \{i-1, i, i+1\}$ (only adjacent characters are swapped);

(c) if $\pi(i) \neq i$ then $P[\pi(i)] \neq P[i]$ (identical characters can not be swapped).

Pattern Matching with Swaps

A *swap permutation* for a string P of length m is a permutation $\pi : \{0, ..., m-1\} \rightarrow \{0, ..., m-1\}$ such that:

(a) if $\pi(i) = j$ then $\pi(j) = i$ (characters at positions i and j are swapped);
(b) for all i, $\pi(i) \in \{i-1, i, i+1\}$ (only adjacent characters are swapped);
(c) if $\pi(i) \neq i$ then $P[\pi(i)] \neq P[i]$ (identical characters can not be swapped).

P has a swapped occurrence in T at location j with k swaps - $P \propto_k T_j$ - if a swap permutation π of P exists such that $\pi(P)$ matches T at location j and $k = |\{i : P[i] \neq P[\pi(i)]\}|/2$

Pattern Matching with Swaps

`fate`

`afte`	$\pi(1)=2, \pi(2)=1, \pi(3)=3, \pi(4)=4$
`afet`	$\pi(1)=2, \pi(2)=1, \pi(3)=4, \pi(4)=3$
`faet`	$\pi(1)=1, \pi(2)=2, \pi(3)=4, \pi(4)=3$
`ftae`	$\pi(1)=1, \pi(2)=3, \pi(3)=2, \pi(4)=4$

Pattern Matching with Swaps

Approximate Pattern Matching with Swaps problem:

- Alphabet Σ
- Pattern P
- Text T

Find all the pairs (j, k) such that P has a swapped occurrence in T at location j with k swaps

Previous work

- (Amir & Lewenstein & Porat, 2002): $\mathcal{O}(n \log m \log \min(m, |\Sigma|))$
- (Cantone & Faro, 2009):
  - $\mathcal{O}(mn)$ dynamic-programming algorithm
  - $\mathcal{O}(\lceil (mn \log m)/w \rceil)$ bit-parallel algorithm; linear $\mathcal{O}(n)$ if $m(\log(\lfloor m/2 \rfloor + 1) + 1) \leq w$

Approximate-BCS algorithm

- BDM-like algorithm:
  - right-to-left scans in windows of size m
  - window update by left-align with the longest prefix matched
- Find the longest prefix of the pattern which has a swapped occurrence in the current window and count the number of swap operations using dynamic-programming

Approximate-BCS algorithm

$$\mathcal{S}_j^h = \{h-1 \leq i \leq m-1 \mid \; P[i-h+1\,..\,i] \propto T_j\}$$

- The set S_j^h includes all the values i such that the h-substring of P ending at position i has a swapped occurrence ending at position j in T

Approximate-BCS algorithm

$$\mathcal{W}_j^h = \{h \leq i < m-1 \mid P[i-h+2\,..\,i] \propto T_j \ \text{ and } \ P[i-h+1] = T[j-h]\}$$

- The set W_j^h includes all the values i such that the $h-1$ substring of P ending at position i has a swapped occurrence at position j in T and the first part of the swap between characters $P[i-h]$ and $P[i-h+1]$ is recognized

$\textsc{Approximate-BCS}$ algorithm

The sets S_j^h and W_j^h can be computed using the following recurrences:

- $$\begin{aligned}\mathcal{S}_j^{h+1} \;=\; \{h-1 \leq i \leq m-1 \mid (i \in \mathcal{S}_j^h \text{ and } P[i-h] = T[j-h]) \text{ or} \\ (i \in \mathcal{W}_j^h \text{ and } P[i-h] = T[j-h+1])\}\end{aligned}$$
- $\mathcal{W}_j^{h+1} \;=\; \{h \leq i \leq m-1 \mid i \in \mathcal{S}_j^h \text{ and } P[i-h] = T[j-h-1]\}$

Base cases:

- $\mathcal{S}_j^0 = \{i \mid 0 \leq i < m\}$
- $\mathcal{W}_j^0 = \{0 \leq i < m-1 \mid P[i+1] = T[j]\}$

Approximate-BCS algorithm

- If $h - 1 \in S_j^h$ there is a swapped occurrence of the prefix of P of length h
- The window is shifted by $m - l$, where $l = \max\{h : h - 1 \in S_j^h\}$

Approximate-BCS algorithm

- If $m-1 \in S_j^m$ P has a swapped occurrence at position j in T
- $m-1 \in S_j^m \iff m-1 \in (S_j^h \cup W_j^h), 1 \leq h \leq m$
- Swap between characters $P[m-1-h]$ and $P[m-1-h+1] \iff m-1 \in S_j^{h+1} \wedge m-1 \in W_j^h \wedge m-1 \notin S_j^h$

Approximate-BCS algorithm

$$P = \text{ooze}, T = \text{ooez}$$

$$m-1 \in W_j^1, m-1 \notin S_j^1$$

$$m-1 \notin W_j^2, m-1 \in S_j^2$$

$$m-1 \in W_j^3, m-1 \in S_j^3$$

$$m-1 \in W_j^4, m-1 \in S_j^4$$

APPROXIMATE-BCS algorithm

- The number of swaps for a match at position j is given by $|\{1 \leq h < m \ : \ (m-1) \ \in \ (S_j^{h+1} \setminus S_j^h)\}|$
- The algorithm maintains a single counter per window
- At iteration h the counter is incremented if $m-1 \in S_j^{h+1} \setminus S_j^h$

APPROXIMATE-BPBCS algorithm

- Simulation of APPROXIMATE-BCS using bit-parallelism
- S_j^h and W_j^h represented as vector of m bits
  - $S_j^h \rightarrow D_j^h$: the $i-h+1$-th bit of D_j^h is set to 1 if $i \in S_j^h$
  - $W_j^h \rightarrow C_j^h$: the $i-h+1$-th bit of C_j^h is set to 1 if $i \in W_j^h$
- Bit mask $M[c]$, i-th bit is set to 1 if $P[i] = c$, as in Shift-And

Approximate-BPBCS algorithm

$$
\begin{array}{ll}
(a) & \mathcal{S}_j^{h+1} \leftarrow \{i : i \in \mathcal{S}_j^h \text{ and } P[i-h] = T[j-h]\} \\
(a') & D_j^{h+1} \leftarrow (D_j^h \ll 1) \mathbin{\&} M[T[j-h]] \\
 & \\
(b) & \mathcal{S}_j^{h+1} \leftarrow \mathcal{S}_j^{h+1} \cup \{i : i \in \mathcal{W}_j^h \text{ and } P[i-h] = T[j-h+1]\} \\
(b') & D_j^{h+1} \leftarrow D_j^{h+1} \mid ((C_j^h \ll 1) \mathbin{\&} M[T[j-h+1]])
\end{array}
$$

APPROXIMATE-BPBCS algorithm

(c) $\mathcal{W}_j^{h+1} \leftarrow \{i : i \in \mathcal{S}_j^h \text{ and } P[i-h] = T[j-h-1]\}$
(c') $C_j^{h+1} \leftarrow (D_j^h \ll 1) \ \& \ M[T[j-h-1]]$

(d) $m - 1 \in S_j^{h+1} \setminus S_j^h$
(d') $((D_j^{h+1} \ \& \ \sim (D_j^h \ll 1)) \ \& \ (1 \ll h)) \neq 0$

Complexity

- Approximate-BCS: $\mathcal{O}(nm^2)$ worst case time complexity, $\mathcal{O}(m)$ space complexity
- Approximate-BPBCS: $\mathcal{O}(\lceil nm^2/w \rceil)$ worst case time complexity, $\mathcal{O}(\sigma \lceil m/w \rceil + \log(\lfloor m/2 \rfloor + 1))$ space complexity

Experimental results

- Implementation in C, compiled with gcc, run on AMD Turion X2 2GHz
- Randσ problems, protein and genome sequences, natural language text
- Set of 100 patterns of fixed length $m \in \{4, 8, 12, 16, 20, 24, 28, 32\}$, randomly extracted from the text
- Comparison between the following algorithms:
  - Approximate-Cross-Sampling (ACS)
  - BP-Approximate-Cross-Sampling (BPACS)
  - Approximate-BCS (ABCS)
  - Approximate-BPBCS (BPABCS)
  - Iliopoulos-Rahman algorithm with a naive check of the swaps (IR&C)
  - BP-Backward-Cross-Sampling algorithm with a naive check of the swaps (BPBCS&C)

Experimental results

Running times for a Rand8 problem

m	4	8	12	16	20	24	28	32
ACS	4.769	4.756	4.762	4.786	4.761	4.808	4.765	4.796
ABCS	11.675	7.273	5.632	4.736	4.167	3.782	3.511	3.305
BPACS	0.832	0.830	0.828	0.831	0.830	0.829	0.827	0.827
BPABCS	0.413	**0.229**	**0.175**	**0.145**	**0.127**	**0.114**	**0.104**	**0.096**
IR&C	**0.282**	0.279	0.279	0.277	0.280	0.279	0.283	0.285
BPBCS&C	0.388	0.249	0.193	0.157	0.141	0.121	0.111	0.101

Running times for a natural language text ($\sigma = 93$)

m	4	8	12	16	20	24	28	32
ACS	3.170	2.757	2.748	2.756	2.761	2.745	2.746	2.754
ABCS	6.175	4.054	3.164	2.705	2.306	2.288	2.042	1.866
BPACS	0.492	0.497	0.492	0.491	0.492	0.491	0.494	0.493
BPABCS	0.194	**0.114**	**0.086**	**0.071**	**0.062**	**0.056**	**0.051**	**0.049**
IR&C	0.171	0.165	0.164	0.168	0.165	0.165	0.165	0.167
BPBCS&C	**0.164**	0.126	0.094	0.076	0.070	0.059	0.056	0.055

Experimental results

Running times for a genome segence ($\sigma = 4$)

m	4	8	12	16	20	24	28	32
ACS	5.629	5.643	5.654	5.636	5.644	5.640	5.647	6.043
ABCS	18.018	11.261	8.805	7.523	6.700	6.117	5.710	5.359
BPACS	0.950	0.914	0.917	0.766	0.874	0.934	0.935	0.843
BPABCS	0.647	**0.318**	**0.266**	**0.232**	**0.195**	**0.174**	**0.160**	0.147
IR&C	**0.262**	0.287	0.314	0.311	0.311	0.311	0.310	0.311
BPBCS&C	0.678	0.367	0.290	0.233	0.204	0.176	**0.160**	**0.146**

Running times for a protein sequence ($\sigma = 22$)

m	4	8	12	16	20	24	28	32
ACS	3.777	3.784	3.671	3.729	3.766	3.703	3.716	3.741
ABCS	7.045	4.557	3.734	3.162	2.806	2.661	2.600	2.351
BPACS	0.565	0.581	0.561	0.563	0.584	0.580	0.534	0.519
BPABCS	0.249	**0.142**	**0.103**	**0.084**	**0.074**	**0.066**	**0.061**	**0.058**
IR&C	0.388	0.390	0.391	0.389	0.391	0.391	0.396	0.389
BPBCS&C	**0.241**	0.145	0.107	0.087	0.075	0.068	0.062	**0.058**

Conclusions

- The Approximate-BPBCS algorithm is the fastest for $m \geq 8$
- The Approximate-BPBCS algorithm scales better than BP-Approximate-Cross-Sampling
    - BP-Approximate-Cross-Sampling: m counters, linear if $m(\log(\lfloor m/2 \rfloor + 1) + 1) \leq w$
    - Approximate-BPBCS: one counter, linear if $m \leq w$