Tighter Packed Bit-Parallel NFA for Approximate String Matching

Heikki Hyyrö

Department of Computer Sciences, University of Tampere, Finland
heikki.hyyro@cs.uta.fi

Given a length-m pattern P and an error threshold k, the bit-parallel NFA of Baeza-Yates and Navarro uses $(m-k)(k+2)$ bits of space. In this paper we decrease this to $(m-k)(k+1)$ by modifying the NFA simulation algorithm. As a side-effect, also the original NFA simulation is slightly improved.

For a string A, let A_i denote the ith character, and $A_{i..j}$ denote the substring whose endpoints are A_i and A_j (for $i \leq j$). We consider the task of approximate matching where we wish to find from a text T all locations j where $ed_L(P, T_{j-h..j}) \leq k$ for some $h \geq 0$. Here k is an error threshold and $ed_L(A, B)$ denotes Levenshtein edit distance between strings A and B.

The Bit-Parallel by Diagonals (BPD) algorithm of Baeza-Yates and Navarro [1] is the fastest verification capable approximate string matching algorithm for a wide range of moderate values of m and k [2]. BPD encodes the type of NFA shown in Fig. 1 into a length-$(k+2)(m-k)$ bit-vector $D = 0\ D_1\ 0\ D_2\ 0...0\ D_{m-k}$. Each D_i is a sequence of $k+1$ bits that describes the status of the $k+1$ states $i+d$ on rows $d = 0 \ldots k$. BPD also preprocesses vectors M_λ that describe matching transitions for character λ (see [1]). The core of BPD is an efficient algorithm for updating the automaton status bit-vector D at text character T_j. See Fig. 2 (*Left*). Here '&', '|', and '^' denote bitwise "and", "or", and "xor", respectively, and '<<' and '>>' denote shifting the bit-vector left and right. Superscript denotes repetition in bit-vectors (e.g. $1^2(01)^2 = 110101$). The segments D_i in D are separated by a 0 bit to avoid overflow in the arithmetic addition of the update algorithm. The following Lemmata enable removing the separator bits.

Lemma 1. *The operation* $(((x + (0^{k+1}1)^{m-k})\ \wedge\ x) >> 1)$ *in algorithm BPD is equivalent to* $(((x + (0^{k+1}1)^{m-k})\ \wedge\ x)\ \&\ x)$.

Lemma 2. *If operation* $(((x + (0^{k+1}1)^{m-k})\ \wedge\ x) >> 1)$ *is replaced by* $(((x + (0^{k+1}1)^{m-k})\ \wedge\ x)\ \&\ x)$ *in BPD, the separator bits do not need explicit resetting.*

Lemma 3. *Let y be an arbitrary bit-sequence of length q, and set $z = y\ \&\ 01^{q-1}$. Then $u = ((y+1)\ \wedge\ y)\ \&\ y)$ is equal to $v = ((z+1)\ \wedge\ z)\ \&\ y)$.*

The modification of Lemma 1 does not alter the number of operations in BPD. Lemma 2 enables removing the operation that resets the separator bits (last line in Fig. 2 (*Left*)). Lemma 3 gives a modification that makes the separator bits obsolete: The algorithm remains correct if we perform the arithmetic addition on a version of D where the $(k+1)$th bit in each D_i is set to 0 (avoiding overflow).

O.H. Ibarra and H.-C. Yen (Eds.): CIAA 2006, LNCS 4094, pp. 287–289, 2006.

We can now form the complete update algorithm for D of form $D_1 D_2 ... D_{m-k}$. It is shown in Fig. 2 (*Right*). Now M_λ must be built without the separator bits. We implemented both BPD variants in C and performed tests on a 32-bit SUN Sparc Ultra 2 with 128 MB RAM and GCC 4.0.2 compiler (using '-O3' switch). Fig. 3 shows the results. The methods used horizontal partitioning (see [1]) when the NFA required more than one computer word. Our BPD used separator bits (removing 2nd last line in our code) if it did not increase the number of words.

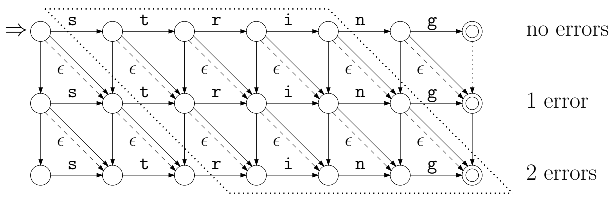

Fig. 1. NFA for approximate string matching with $P =$ "string" and $k = 2$

$$\begin{aligned}
&x \leftarrow (D >> (k+2)) \mid M_{T_j}\\
&D' \leftarrow ((D << 1) \;\mid\; (0^{k+1}1)^{m-k})\\
&\quad \& \; ((D << (k+3)) \;\mid\; (0^{k+1}1)^{m-k-1}01^{k+1})\\
&\quad \& \; (((x + (0^{k+1}1)^{m-k}) \wedge x) >> 1)\\
&\quad \& \; (0\,1^{k+1})^{m-k}
\end{aligned}$$

$$\begin{aligned}
&x \leftarrow (D >> (k+1)) \mid M_{T_j}\\
&D' \leftarrow ((D << 1) \;\mid\; (0^{k}1)^{m-k})\\
&\quad \& \; ((D << (k+2)) \;\mid\; (0^{k}1)^{m-k-1}1^{k+1})\\
&z \leftarrow x \;\&\; (0\,1^{k})^{m-k}\\
&D' \leftarrow D' \;\&\; (((z + (0^{k}1)^{m-k}) \wedge z) \;\&\; x
\end{aligned}$$

Fig. 2. Algorithms for updating D. (*Left*) Original BPD. (*Right*) Our tight BPD.

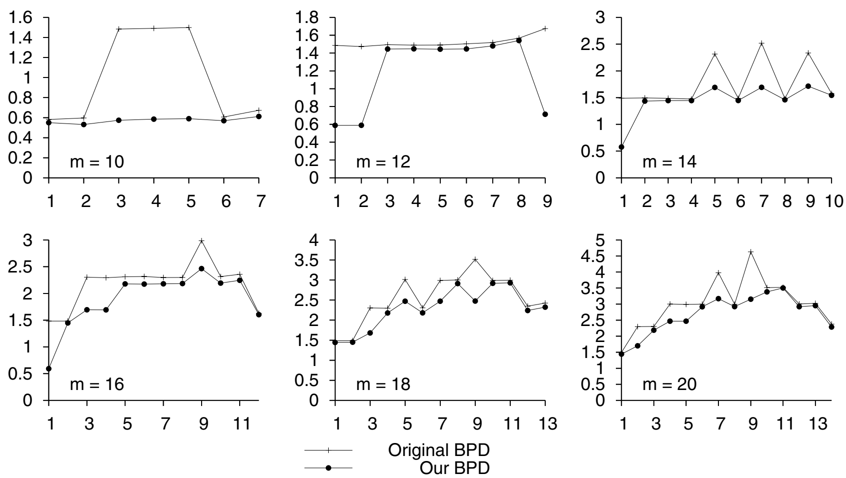

Fig. 3. Average time in seconds for approximate search in 8 MB English text

References

1. Baeza-Yates, R., and Navarro, G. Faster Approximate String Matching. *Algorithmica*, 23:127–158, 1999.
2. Navarro, G., and Raffinot, M. *Flexible Pattern Matching in Strings – Practical on-line search algorithms for texts and biological sequences.* Cambridge University Press, Cambridge, UK, 2002.