Fast Bit-Vector Algorithms for Approximate String Matching Under Indel Distance

Heikki Hyyrö[1], Yoan Pinzon[2,⋆], and Ayumi Shinohara[1,3]

[1] PRESTO, Japan Science and Technology Agency (JST), Japan
helmu@cs.uta.fi
[2] Department of Computer Science, King's College, London, UK
pinzon@dcs.kcl.ac.uk
[3] Department of Informatics, Kyushu University 33, Fukuoka 812-8581, Japan
ayumi@i.kyushu-u.ac.jp

Abstract. The approximate string matching problem is to find all locations at which a query p of length m matches a substring of a text t of length n with at most k differences (insertions, deletions, substitutions). The fastest solutions in practice for this problem are the bit-parallel NFA simulation algorithms of Wu & Manber [4] and Baeza-Yates & Navarro [1], and the bit-parallel dynamic programming algorithm of Myers [3]. In this paper we present modified versions of these algorithms to deal with the restricted case where only insertions and deletions (called indel for short) are permitted. We also show test results with the algorithms.

1 IndelMYE Algorithm

The bit-parallel approximate string matching algorithm of Myers [3], MYE, is based on the classical dynamic programming approach where a $(m+1) \times (n+1)$ matrix D is computed using the well-known recurrence $D_{i,j} = \min \{D_{i-1,j-1} + \delta(p_i, t_j),\ D_{i-1,j},\ D_{i,j-1}\}$, subject to the boundary condition $D_{0,j} = 0$ and $D_{i,0} = i$, where $\delta(p_i, t_j) = 0$ iff $p_i = t_j$, and 1 otherwise. The solution to the approximate string matching problem is all the locations j where $D_{m,j} \leq k$. MYE is based on the observation that the vertical and horizontal differences between adjacent cells in D (i.e. $D_{i,j} - D_{i-1,j}$ and $D_{i,j} - D_{i,j-1}$) have the value -1, 0, or 1, and the diagonal differences (i.e. $D_{i,j} - D_{i-1,j-1}$) have the value 0 or 1. This enables the algorithm, as presented in [2], to use the following length-m bit-vectors to represent the vertical, horizontal and diagonal differences:

— $Pv_i = 1$ iff $D_{i,j} - D_{i-1,j} = 1$, $Nv_i = 1$ iff $D_{i,j} - D_{i-1,j} = -1$
— $Ph_i = 1$ iff $D_{i,j} - D_{i,j-1} = 1$, $Nh_i = 1$ iff $D_{i,j} - D_{i,j-1} = -1$
— $Zd_i = 1$ iff $D_{i,j} = D_{i-1,j-1}$

The values of Pv and Nv are known for the initial case $j = 0$. The steps of the algorithm at text position j are as follows. First the new diagonal vector Zd'

⋆ Part of this work was done while visiting Kyushu University. Supported by PRESTO, Japan Science and Technology Agency (JST).

M. Bieliková et al. (Eds.): SOFSEM 2005, LNCS 3381, pp. 380–384, 2005.

is computed by using Pv, Nv and $M(t_j)$, where for each character λ, $M(\lambda)$ is a precomputed length-m match vector where $M(\lambda)_i = 1$ iff $p_i = \lambda$. Then the new horizontal vectors Ph' and Nh' are computed by using Zd', Pv and Nv. Finally the new vertical vectors Pv' and Nv' are computed by using Zd', Nh' and Ph'. The value of the dynamic programming cell $D_{m,j}$ is maintained during the process by using the horizontal delta vectors (the initial value is $D_{m,0} = m$). A match of the pattern with at most k errors is found whenever $D_{m,j} \leq k$.

The dynamic programming recurrence for indel distance is $D_{i,j} = (\text{if } p_i = t_j \text{ then } D_{i-1,j-1} \text{ else } \min\{D_{i-1,j},\ D_{i,j-1}\})$, which makes also the case $D_{i,j} - D_{i-1,j-1} = 2$ possible. To help deal with this complication, we will use the following additional vertical and horizontal zero vectors.

— $Zv_i = 1$ iff $D_{i,j} - D_{i-1,j} = 0$, $Zh_i = 1$ iff $D_{i,j} - D_{i,j-1} = 0$

Naturally $Zv = \sim (Pv \mid Nv)$ and $Zh = \sim (Ph \mid Nh)$, where $\sim$ is the bit-wise complement operation. In the following we describe the steps of our algorithm for updating the bit-vectors at text position j, where $1 \leq i \leq m$ and $1 \leq j \leq n$.

i.) The new diagonal vector Zd' is computed exactly as in MYE. That is, $Zd'=(((M(t_j)\ \&\ Pv) + Pv) \wedge Pv) \mid M(t_j) \mid Nh$.

ii.) The new horizontal zero vector Zh' is computed by using Pv, Zv and Zd'. By inspecting the possible difference combinations, we get the formula $Zh'_i = (Zh'_{i-1}\ \&\ Pv_i\ \&\ (\sim Zd'_i)) \mid (Zv_i\ \&\ Zd'_i)$. We use a superscript to denote bit repetition (e.g. $001011 = 0^2101^2 = 0(01)^21$). The self-reference of Zh' implies that each set bit $Zh'_i{=}1$ in Zh' can be assigned into a distinct region $Zh'_{a..b}{=}1^{b-a+1}$, where $1 \leq a \leq i \leq b \leq m$, $Zv_a\ \&\ Zd'_a{=}1$ or $a{=}1$, $Zh'_{r-1}\ \&\ Pv_r\ \&\ (\sim Zd'_r){=}1$ for $r \in a+1 \ldots b$, and $Zh'_{b+1}\ \&\ Pv_{b+1}\ \&\ (\sim Zd'_{b+1}){=}0$. We also notice that the conditions $Zv_a\ \&\ Zd'_a{=}1$ and $Pv_r\ \&\ (\sim Zd'_r){=}1$ for $r \in a+1 \ldots b$ imply that $Zh'_r{=}1$ for $r \in a \ldots b$. If we shift the region $a+1 \ldots b$ of set bits one step right to overlap the region $a \ldots b-1$ and perform an arithmetic addition with a set bit into position a, then the bits in the range $a \ldots b-1$ will change from 1 to 0 and the bit b from 0 to 1. These changed bits can be set to 1 by performing XOR. From this we derive the final formula: $Zh'=(((Zv\ \&\ Zd') \mid ((Pv\ \&\ (\sim Zd'))\ \&\ 0^{m-1}1)) + ((Pv\ \&\ (\sim Zd'))\ {>>}\ 1)) \wedge ((Pv\ \&\ (\sim Zd'))\ {>>}\ 1)$.

iii.) The new horizontal vector Nh' is computed as in MYE by setting $Nh' = Pv\ \&\ Zd'$, after which we can also compute $Ph' = \sim (Zh' \mid Nh')$.

iv.) The new vertical vector Zv' is computed by using Zh', Ph', Zd' and Zv. We notice that $D_{i,j}{=}D_{i-1,j}$ iff either $D_{i-1,j}{=}D_{i-1,j-1}{=}D_{i,j}$, or $D_{i,j-1}{=}D_{i-1,j-1}$ and $D_{i-1,j}{=}D_{i-1,j-1}+1$. In terms of the delta vectors this means that $Zv'_i{=}1$ iff $Zh'_{i-1}\ \&\ Zd'_i{=}1$ or $Ph'_{i-1}\ \&\ Zv_i\ \&\ (\sim Zd'_i){=}1$. From this we get the formula $Zv'=(((Zh'\ {<<}\ 1) \mid 0^{m-1}1)\ \&\ Zd') \mid ((Ph'\ {<<}\ 1)\ \&\ Zv\ \&\ (\sim Zd')$.

v.) The new vertical vector Nv' is computed as in MYE by setting $Nv'{=}(Ph'\ {<<}\ 1)\ \&\ Zd'$, after which we can also compute $Pv' = \sim (Zv' \mid Nv')$.

Fig. 1 (*upper left*) shows a high-level template for the bit-parallel algorithms, and Fig. 1 (*lower left*) shows the complete formula for computing the new difference

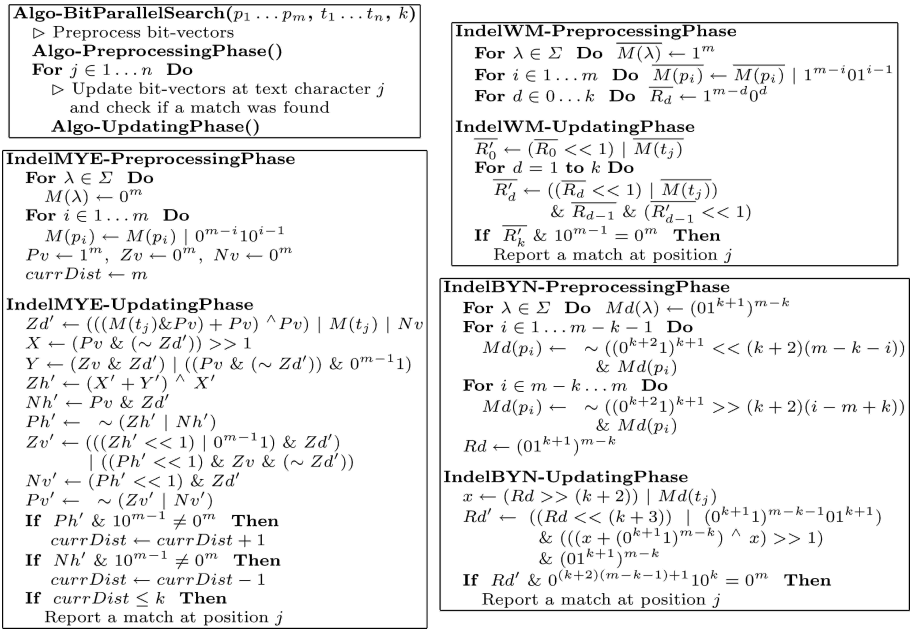

Fig. 1. Bit-parallel approximate string matching under indel distance: a template (*upper left*), indelMYE (*lower left*), indelWM (*upper right*), and indelBYN (*lower right*)

vectors at text position j. The running time of indelMYE is $O(\lceil m/w \rceil n)$ as a vector of length m may be simulated in $O(\lceil m/w \rceil)$ time using $O(\lceil m/w \rceil)$ bit-vectors of length w. The cost of preprocessing is $O(\sigma \lceil m/w \rceil + m)$.

2 Bit-Parallel NFA Simulation Algorithms: IndelWM and IndelBYN

The bit-parallel approximate string matching algorithms of Wu & Manber [4] (WM) and Baeza-Yates & Navarro [1] (BYN) simulate a non-deterministic finite automaton (NFA), R, by using bit-vectors. R has $(k+1)$ rows, numbered from 0 to k, and each row contains m states. Let $R_{d,i}$ denote the ith state on row d of R. $R_{d,i}$ is active after processing the jth text character iff $D_{i,j} \leq d$ in the corresponding dynamic programming matrix D, and so an approximate occurrence of the pattern is found when the state $R_{k,m}$ is active.

WM uses $(k+1)\lceil m/w \rceil$ and BYN $\lceil (k+2)(m-k)/w \rceil$ bit-vectors of length w to encode R. Both perform a constant number of operations per bit-vector at text position j. For reasons of space, we do not discuss here further details of these algorithms. We just note that the bit-vector update formulas for

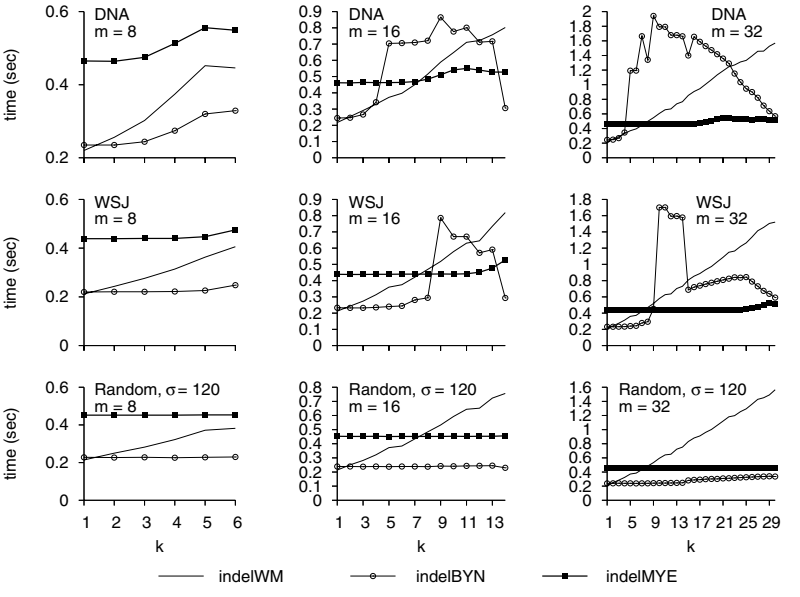

Fig. 2. The average time for searching in a ≈ 20 MB text. The first row is for DNA (a duplicated yeast genome), the second row for a sample of Wall Street Journal articles from TREC-collection, and the third row for random text with alphabet size $\sigma = 120$

both correspond to the dynamic programming recurrence, where each edit operation has a distinct part in the formula. Hence the modification for indel distance is straightforward: we only need to remove the part for substitution. Fig. 1 (*upper right*) shows indelWM and Fig. 1 (*lower right*) shows indelBYN: our versions of WM and BYN, respectively, that are modified for indel distance. The running time of indelWM is $O(k\lceil m/w \rceil n)$, and its time for preprocessing is $O(\sigma\lceil m/w \rceil + m)$. The running time of indelBYN is $O(\lceil (k+2)(m-k)/w \rceil n)$, and its time for preprocessing is $O(\sigma\lceil (k+2)(m-k)/w \rceil + m)$.

3 Experiments

We implemented and tested the three bit-parallel variants for approximate string matching under indel distance. IndelWM and IndelMYE were implemented fully by us, and IndelBYN was modified from a code by Baeza-Yates & Navarro. The computer was a 3.2Ghz AMD Athlon64 with word size w=32, 1.5 GB RAM, Windows XP, and MS Visual C++ 6.0 compiler using high optimization. The patterns were selected randomly from the text, and for each (m, k) we recorded the average time over searching 100 patterns. Fig. 2 shows the results.

References

1. R. Baeza-Yates and G. Navarro. Faster approximate string matching. *Algorithmica*, 23(2):127–158, 1999.
2. H. Hyyrö. Explaining and extending the bit-parallel approximate string matching algorithm of Myers. Technical Report A-2001-10, University of Tampere, 2001.
3. G. Myers. A fast bit-vector algorithm for approximate string matching based on dynamic progamming. *Journal of the ACM*, 46(3):395–415, 1999.
4. S. Wu and U. Manber. Fast text searching allowing errors. *Comm. of the ACM*, 35(10):83–91, October 1992.