Joint NCC & IIS Bull., Comp. Science, 12 (1999), 10–14
© 1999 NCC Publisher

The use of finite automata to speed up searching in the text data bases

V.D. Gusev, L.A. Nemytikova

Pattern matching is a component of many information retrieval problems. We deal with searching for a set of patterns. The following generalizations of the problem are considered on the same basis: 1) searching for the set of patterns by a *partially–specified* query of *large cardinality*; 2) revealing the coincidences, insertions and intersections of patterns in the query; 3) searching for close "patterns" in the text (approximate string matching). The algorithms for solving the above problems using deterministic and nondeterministic finite automata oriented to personal medium–speed computers have been proposed. The algorithms have been tested on searching for a real set of patterns.

1. Introduction

Three information retrieval problems are considered on the same basis:

1) searching for the set of patterns by a *partially–specified* query of *large cardinality*;
2) revealing the coincidences, insertions and intersections of patterns in the query;
3) searching for close "patterns" in the text (approximate string matching).

Problem (1) is commonly considered for the degenerate case (a query with one pattern [1], or with several patterns with the total number of symbols not exceeding the length of a computer word in bits [2]). Problem (2) is urgent exactly for the case of queries of large cardinality where interrelations between the patterns are difficult to avoid. Problem (3) is similar to (2) but these are text fragments that serve as patterns, and the interrelations between them are formulated in terms of edit distance.

In all cases the solution of the problems is based on choosing relatively short reference fragments ("cores") in the patterns that play the role of initial "starting points" of searching, with the subsequent discard of those proved to be false.

2. The statement of the problems and notation

A partially–specified pattern is a string of symbols whose elements are specified as definite subsets of the initial alphabet. Such patterns naturally arise in the classification problems when generating a consensus (generalizing) sequence for a group of similar texts.

Here Σ is the initial (finite) alphabet, $s = |\Sigma|$ is the size of the alphabet Σ; T is the text composed of Σ–elements, $T[i]$ is the i-th symbol of the text, $T[i:j]$ are text symbols from the i-th to the j-th one inclusively; $N = |T|$ is the length of the text T; $D = \delta_1, \delta_2, \ldots, \delta_d$ is the set of the Σ subsets involved in the query $(|\delta_k| \geq 2, \quad 1 \leq k \leq d)$; $p = b_1 b_2 \ldots b_M \quad (b_i \in \Sigma \cup D)$ is a partially–specified pattern, $M = |p|$ is the length of the pattern, $P = p_1, p_2, \ldots, p_n$ is a partially–specified query for the set of patterns, n is the query cardinality.

We call the function

$$|b_i| = \begin{cases} 1, & \text{if } b_i \in \Sigma \\ |\delta_k|, & \text{if } b_i = \delta_k (\delta_k \in D). \end{cases}$$

the uncertainty degree of the i-th position of the pattern.

Each partially–specified pattern p may be replaced by an equivalent (in the sense of the search completeness) set of exactly specified (constant) patterns $E(p)$. The number of its elements $|E(p)| = Q(p) = \prod_{i=1}^{M} |b_i|$ can be treated as the uncertainty degree of the pattern p. The process of obtaining $E(p)$ from p is called the *expansion* of p.

The problem of searching for the set of patterns by a partially–specified query P consists in finding in the text T all the fragments that match at least one pattern from P. A fragment $a = a_1a_2 \dots a_M$ $(a_i \in \Sigma,\ 1 \le i \le M)$ will be considered as a matching pattern $p = b_1b_2 \dots b_M$ $(b_i \in \Sigma \cup D)$, if for all $1 \le i \le M$ the following takes place: $a_i = b_i$ if $b_i \in \Sigma$, or $a_i \in b_i$ if $b_i = \delta_k$ $(\delta_k \in D)$. In other words, a fragment "a" matches p if $a \in E(p)$.

The second problem under study consists in revealing the coincidences, insertions and intersections of patterns from P. For a pair of patterns p_1 and p_2 such that $|p_1| = |p_2|$, the dependences of such kinds are interpreted in the language of equivalent sets as $E(p_1) = E(p_2)$, $E(p_1) \subset E(p_2)$ (or vice versa), and $E(p_1) \cap E(p_2) \neq \emptyset$, respectively. If $|p_1| = M_1$, $|p_2| = M_2$ and $M_1 < M_2$, this can mean the insertion of a short pattern into a longer one, i.e., searching for a position $1 \le i \le M_2 - M_1 + 1$ in p_2 such that p_1 is inserted into $p_2[i : i + M_1 - 1]$ (or vice versa). The problem of the intersection between short and long fragments may be considered similarly.

The third problem consists in finding in the text T all possible pairs of fragments $\{p, q\}$ of length not less than the prescribed limit, close in the sense of the edit distance $d(p, q)$. It is defined as a minimum number of "substitution", "insertion" and "deletion" operations that transform p into q. Fragments p and q are considered as close if $d(p, q) \le k$, where $k \ll \min\{|p|, |q|\}$.

3. Schemes of solutions

Let us use the most attractive of currently existing schemes of searching for the set of *constant* patterns — Aho–Corasick deterministic finite automaton [3]. Formally problem (1) may be reduced to this case by expanding each pattern from P. However, the number of constant patterns increases exponentially with increasing lengths of the patterns.

The solution consists in choosing a relatively *short* reference core z in each of the patterns, replacing it by an equivalent set of constant cores $E(z)$, and synthesizing a search automaton by a combination of sets $E(z)$. Since the size of the core L is essentially less than the length of the pattern, and the choice of cores is optimized, a double effect is achieved: 1) the number of the automaton states is considerably reduced; 2) the effect of an exponential increase in the number of elements in $E(z)$ fails to manifest itself for the lack of time.

The searching procedure becomes two-step. First the automaton reveals the initial starting points, *cores*, common for the text and patterns (problem (1)), or common for different patterns (problems (2) and (3)). Then the false starting points are eliminated by expanding the cores already found. As a rule, the expansion procedure is not time-consuming, since most commonly it is completed at the first step (a mismatch is detected).

Let us consider the specific features of each of the algorithms in detail.

3.1. The search for the set of patterns by a partially–specified query

The size of the core L is chosen taking into account the alphabet cardinality s, the uncertainty degree of patterns in the query, and the operation memory. The value $L = 3 \div 4$ is acceptable for alphabets of medium cardinality $(s = 20 \div 50)$, the value $L = 7 \div 8$ — for small alphabets $(s = 2 \div 4)$. For a given L, the core z_i in the pattern $p_i = b_1b_2 \dots b_M$ $(1 \le i \le n)$ is chosen in the form of a fragment $p_i[b_k : b_{k+L-1}]$ with a minimum value of Q $(1 \le k \le M - L + 1)$.

The expansion of the cores z_i is combined with the construction of the automaton by the sets $E(z_i)$. The principle of construction is "sticking together" the common prefix parts in the patterns. The automaton is described by the next-move function $g(st, a)$, the failure function $f(st)$ and the output function $o(st)$, where st is the automaton state, "a" is the input text symbol playing the role of a control action [3].

Cores (starting points) are detected by scanning the text. The false starting points are selected by matching the common core of the pattern and the text fragment with subsequent checking whether

their neigbourhoods match. Total preprocessing and search time is on the average

$$O(N + \sum_i |p_i| + n \cdot \overline{Q}_L(N/s^L + L)),$$

where $\overline{Q}_L$ is the uncertainty degree of the selected cores averaged over all patterns [4].

Note 1. The complexity analysis shows that, in the case of relatively small lengths of the text, time is primarily spent on the construction of the automaton by a large number of patterns, rather than on the search itself. One can proceed in another way: to construct the automaton (without failure functions) by all possible text fragments of the length L, and to process the patterns by the automaton. For relatively small values of N ($N < s^L$) this strategy is preferable [4].

Note 2. Patterns containing the elements of X type ("don't care" characters) have the greatest degree of uncertainty. For such patterns it is not always possible to choose the core not involving X, and this increases the time and memory consumption in preprocessing. Two techniques of "including" the element X into Σ are proposed [5]. With these techniques, alternative paths marked by symbol X appear, and the automaton becomes nondeterministic. As a consequence, the failure function is defined in a way different from [3], and the return function is introduced. In such an automaton, the search time slightly increases but memory consumption is essentially reduced [5].

3.2. Revealing coincidences, insertions, and intersections of patterns

We call the reader's attention to two complicating factors.

1) To detect insertions of short patterns into longer ones, all patterns are formally reduced to one and the same length $\bar{M}$ without loss of information they carry. The number of patterns in the query increases from n to

$$n' = n_{\bar{M}} + \sum_{k=1}^{M_{\max}-\bar{M}} (k+1) \cdot n_{\bar{M}+k} + \sum_{k=1}^{\bar{M}-M_{\min}} (k+1) \cdot n_{\bar{M}-k},$$

where $M_{\min} \div M_{\max}$ is the variation range of pattern lengths in the query, n_s is the number of patterns of length s, and $\bar{M} = argu \max_s n_s$.

2) The cores chosen in the patterns of length $\bar{M}$ must occupy the *same positions* in all patterns so as to ensure detection of all interrelated patterns. On the average, this increases their uncertainty degree Q as compared to problem 1.

By virtue of the foregoing, one has to turn to the well-known combinatory strategy "divide and conquer". A new set of patterns $P' = \{p_1, p_2, \ldots, p_{n'}\}$ where $p_i = b_1^i b_2^i \ldots b_{\bar{M}}^i$ is partitioned by one of the positions k^* into $|\Sigma|$ subsets $P'_j = \{p_i : b_{k^*}^i$ contains the j-th element of the alphabet $\Sigma\}$ intersecting in the general case. The solution of the initial problem is formed from the solutions obtained by separate sets P'_j.

For each P'_j the automaton of the core type whose finite states contain relatively short lists of patterns to be matched (the core expansion step) is constructed. The running time of the algorithm is on the average $O(n' \cdot r \cdot \overline{Q}_L \cdot L)$ where L, $\overline{Q}_L$ and n' are defined above, and r is the average uncertainty degree falling at one position in P' ($r < |\Sigma|$).

3.3. Approximate string matching in the text

Text fragments separated by a sliding window of size $\bar{M}$ play the role of patterns (already constant). Expectation of the maximum repeat length in a random text ($LR \sim 2\ln(N)/|\ln \sum_{r=1}^{s} p_r^2|$, where p_r ($1 \le r \le s$) are the probabilities of the alphabet element appearance) may serve as the lower bound for the value of $\bar{M}$. If $\bar{M} > LR$ and $k \ll \bar{M}$, where "k" is the restriction on the length of the edit

distance between close patterns, then the expectation of the number of (M,k)-repeats is not large, and the repeats are functionally significant.

For fixed M and k, the guaranteed size L of the core without errors (for example, if errors take the form of substitutions, then for $M = 9$ and $k = 2$ we have $L = 3$) common for any pairs of patterns is calculated. The automaton of the core type is constructed by all possible fragments of a text of length L. Finite states fix all occurrences of identical cores in the text. The expansion of each pair of identical cores up to the size of M-word (or until the edit distance limit is exceeded) is performed immediately in the text in both directions (the smallness of the parameter "k" is essentially used).

4. Testing of the algorithms

4.1. Problem (1)

To test the algorithms related to problem (1), real searching for the set of patterns by a partially–specified query of large cardinality ($n \approx 2{,}3 \cdot 10^4$ $6 \le |p_i| \le 10$, $|\Sigma| = 20$, $|D| \approx 2300$) was used. Each pattern was a consensus description of some informative area in one of the protein families. A specific example of a pattern is given here: p = *N–DE–FGHY–CN–DKLN–CRT–B–C*. Here lines separate the pattern positions, all symbols are the elements of the aminoacid alphabet Σ, the subsets δ_k of the alphabet Σ are specified by enumeration of their elements (between pairs of the neighbouring lines), $Q(p) = |E(p)| = 2 \cdot 4 \cdot 2 \cdot 4 \cdot 3 = 192$.

On the whole, for the given searching for the set of patterns, the average uncertainty degree for one pattern $\overline{Q}(p)$ is close to 27. For the core size $L = 3$ and 4 we have optimal time of total preprocessing and search. If the choice of the position of the core z is optimal, then $\overline{Q}(z) = \overline{Q}_L \approx 3{,}3$, i.e., a three-fold (on the average) decrease in the core size as compared to the average length of patterns brings more than an eight-fold decrease in the uncertainty degree. This is an essential argument in favour of the core algorithm.

When searching in the text, the expansion procedure is very fast: the average number of checks was 1,2 before the result was obtained (positive or negative). With the automaton constructed by patterns, searching on personal medium-speed computers proceeds almost in real time. When the automaton was constructed by the text, a noticeable gain in time was observed, as compared to the previous case, for the text length $N < 3 \cdot 10^4$.

4.2. Problem (2)

Problem (2) was also solved for the above searching for the set of patterns formed automatically. 443 pairs of patterns with interrelations of "coincidence", "insertion" and "intersection" types were revealed. The interrelations between patterns imply the relations (possibly, occasional) between different protein families. Multiple interrelations between two families may result in revision of the idea of their independence. The relation between patterns p_1 = *MNQ–P–GQ–HL–IV–DH–I–Y* and p_2 = *P–NQ–H–I–DH–I–Y* from NEUCON and FURIN families, respectively, is an example of the "insertion with intersection" interrelation. It is easily seen that p_2 intersects the suffix part of p_1 ($E(p_2) \cap E(p_1[2:8]) \neq \emptyset$).

4.3. Problem (3)

Approximate string matching was performed on the set of musical texts (Russian, French and American folk songs). The length of the texts was about $2 \cdot 10^4$ symbols, the interval-metric representation of melodies ($|\Sigma| \approx 40$) was employed. We have considered the repeats differing: a) only in substitutions ($M = 9$, $k = 2$; $M = 12$, $k = 3$, and etc.); b) only in insertions (with approximately the same values of parameters); c) in string substitutions (a string of symbols is replaced by another one of about the same length).

Based on the repeats obtained, the statistics of substitutions, insertions or string substitutions was formed. It can serve as a basis for introducing the weights of the corresponding operations in musically-oriented closeness measures of the edit distance type. The difference in statistic between different groups of melodies may be used for classification purposes (classification by style, genre, etc.).

4.4. Problems (1) ÷ (3)

Elementary structural units of Old Russian sacred music ("popevky") collected by V.M.Metallov served as another testing set for the developed algorithms ($n \approx 500$, $4 \leq |p_i| \leq 21$, $|\Sigma| = 30$). The following points were considered: covering the texts of hymns ("pesnopeniya") by popevky (problem (1)), insertions of popevky into each other (problem (2)), and their distinguishability (problem (3)). A wide scatter in the lengths of "popevky" turned out to be essential, and this gave rise to another modification of problem (2). It is associated with the detection of composite popevky that proved to be rather numerous. They are represented by concatenation of shorter "popevky".

The authors are grateful to the Russian Foundation for Basic Research for financial support (Project N 96-06-80576).

References

[1] U. Vishkin, *Deterministic sampling — a new technique for fast pattern matching*, SIAM J. Comput., **20**, No 1, 1991, 22–40.

[2] Sun Wu, Udi Manber, *Fast text Searching allowing errors*, Communications of the ACM, **35**, No 10, 1992, 83–91.

[3] A.V. Aho, M.J. Corasick, *Efficient string matching: an aid to bibliographic search*, Communications of the ACM, **18**, No 6, 1975, 333–340.

[4] V.D. Gusev, L.A. Nemytikova, *Algorithms of searching for the set of patterns by partially-specified query*. Artificial intelligence and expert systems, Novosibirsk, 1996, Issue 157: Vychislitelnye sistemy, 12–39 (in Russian).

[5] L.A. Nemytikova, *The use of nondeterministic finite automata to speed up searching in text data bases*. Artificial intelligence and expert systems. Novosibirsk, 1997, Issue 160: Vychislitelnye sistemy, 188–209 (in Russian).