Efficiency of AC-Machine and SNFA in Practical String Matching

Martin Bloch

Department of Computer Science and Engineering,
Faculty of Electrical Engineering,
Czech Technical University,
Karlovo nám. 13,
121 35 Prague 2,
Czech Republic

e-mail: bloch@cslab.felk.cvut.cz

Abstract. A note on practical experience with on Aho-Corasick-machine and SNFA (Searching NFA) estimating the construction aspects and run cost. It is shown that SNFA can be more practical than AC-machine.

Key words: string matching, AC algorithm, nondeterministic finite automaton, practical text searching

1 Introduction

Let P be finite set of pattern character strings of the finite length and let T be a text, i.e. string of characters of length t. The task is to find all occurrences of all patterns in the text using an abstract matching machine. We can use different machines optimizing $space$ or $time$ or $cost = space * time$ of their run phase. All these machines are defined by P completely. We can recognize two cases of P or pattern machines: static or dynamic. In static case set P is constant. In dynamic case patterns can be included into or deleted from the set P. Let us pay attention to two machines SNFA (Searching Nondeterministic Finite Automaton) and AC-machine (Aho-Corasick [AC75]) only.

Both these machines include G-trie as their basic structure. The G-trie is root $\rightarrow$ leaf oriented tree, in which every node is uniquely labeled by prefix $\in$ Prefix(P) and trajectory from root to the node is also labeled by this prefix. The G-depth of a node is length of trajectory from root to the node. Prefix(P) is set of all prefixes of all patterns in P. The G-trie represents goto function.

Undoubtedly, AC-machine represents great theoretical leap in stringology proving linear run time depending just on t and not depending on P. (AC-machine is specific case of implementation of DFA (Deterministic Finite Automaton) and makes less than $2t$ transitions after reading text T). This lovely miracle was obtained by two ingenious tricks:

- The G-trie is overlaid by leaf $\rightarrow$ root oriented fail F-tree containing nonlabeled fail arrows. (G and F have the same set of nodes and common root.) Each node (except common root) is equipped by fail arrow which points to the node which is labelled by longest suffix of label of the node. The F-tree represents failure function. The F-depth of a node is length of trajectory to the root.
- The root contains labeled loops for all characters in given alphabet which do not label any arrows going from the root to another node.

The space complexity of the AC-machine is linear dependent on the number of nodes in the trie. It can be implemented in different ways (saving the linearity) but practically requiring more space (say, linked list or direct access table implementation). Very good practical method designed by [AYS88] is based on the interspersed direct access tables. (Imagine a set of combs with missing teeth to be somehow assembled in line overlaying such a way that no tooth would mask another one and the length of this assembly is almost minimal).

This method preserves fast unit time state transition and space linearity with little space overhead caused by unused slots. The overhead under 5% can be easily reached in large practical cases [Hla96]. The only severe problem in the implementation of AC/Aoe is necessity of solving the "teeth conflict" by moving one comb to another place. Aoe et al. solve this task by moving the smaller comb having smaller number of teeth (that sounds somehow logically). Hladík moves the newer comb what spares the time necessary for counting teeth in both combs and allows to use the memory in a better way. It is not known whether Hladík's method has higher overhead than Aoe's method.

The construction of the goto function, ie. G-trie, is rather simple problem and moreover it can be created dynamically. The construction of fail function is not difficult for static case when the trie is scanned width-first. It is difficult to maintain the dynamic AC-machine because after addition or deletion of any pattern the fail function should be recalculated completely. In order to avoid this tedious recalculation, inverse arrow for every fail arrows should be implemented but it increases memory requirements further. The machine equipped with such inverse arrows is denoted as ACi-machine.

2 Comparison of AC-machine and SNFA

SNFA is a very specific case of NFA and its construction is simpler than AC-machine but its run is principally slower. SNFA is just the G-trie amended by set of labelled loops for all characters in given alphabet in its root. Therefore the root is the only source of nondeterminism.

Let us imagine that the general NFA is a sort of a playboard and its run is a sort of a game played by team of pawns or dwarfs. At most one pawn can stay at one node each time. After reading of an input character every pawn goes via all goto arrows labelled by the read character. If there is no such an arrow the pawn is removed. When more pawns are encountered in one node they join into one pawn. The game is over when there is no pawn on the playboard or when the input is exhausted.

SNFA starts with just one pawn in the root node. This root pawn permanently stays in the root because of the set of root loops and plays the role of a bee queen

yielding other pawns. Pawns cannot encounter each other at one node (the playboard is a trie) and therefore no problem arise with the joining operation. Two pawns never stay at the nodes with the same G-depth. The maximum number of pawns is the length of the longest pattern in P plus one.

AC-machine is a DFA and therefore just one pawn plays the game starting in the root node, too. This pawn cannot be removed because he uses the fail arrow to save his life when the input character does not match any goto arrow. In the worst case the pawn falls down into the root and the root node provides his immortality.

There is a certain remarkable similarity between SNFA and AC-machine. AC-machine pawn determines where pawns would stay in equivalent SNFA (having identical G-trie). This is given by the fail arrow trajectory from any node to the root which shows where all pawns would stay in SNFA. The number of pawns is given by the F-depth of such node (plus one for permanent pawn in the root). The maximum number of pawns is therefore limited by the depth of the F-tree.

3 Experiments

Several tests have been performed on two large practical pattern sets:

CA Queries for SDI retrieval in the Chemical Abstracts data base. Pattern are chemical terms mainly.

WN Word Net thesaurus of the English language prepared at Princeton University containing nouns, verbs, adjectives and adverbs.

The following table shows main characteristics of pattern sets, G-tries and F-trees.

Pattern set	**CA** Chem. Abstracts SDI queries	**WN** WordNet thesaurus
no. of patterns	13872	174678
no. of nodes	70315	882831
avg. G-depth	10.8	10.2
max. G-depth	47	63
avg. F-depth		4.3
max. F-depth	8	9

The following table shows typical branching of the G-trie that can be useful for its implementation.

Node type	**output degree**	**percentage**
fork	>1	8
single	1	77
leaf	0	15

In order to estimate the run efficiency of the AC-machine and the SNFA, following characteristics have been introduced:

RSR is a relative space requirement given by the ratio of number of arrows to the number of nodes. For the G-trie is RSR=1, for the AC-machine is RSR=2, for the ACi-machine RSR=3.

ATC is the arrow transition coefficient defined as the average number of transitions via G-arrows or F-arrows caused by one input character.

ANP is an average number of pawns taking part in the game.

RRT is a relative run time consumed. RRT = ATC * ANP.

RRC is a relative run cost where RRC = RSR * ATC * ANP.

The following table shows these run characteristics:

Pattern set	**CA**		**WN**	
Machine	SNFA	AC	SNFA	AC
RSR	1	2	1	2
ATC	1	1.42	1	1.56
ANP	3.3	1	2.8	1
RRT=ATC*ANP	3.3	1.42	2.8	1.56
RRC=RSR*ATC*ANP	3.3	2.84	2.8	3.12

4 Conclusions

It follows from previous table that time ratio RRT_{SNFA}/RRT_{AC} is 2.32 or 1.79 and therefore AC is about twice faster. Nevertheless, the cost ratio RRC_{SNFA}/RRC_{AC} varies from 1.16 to 0.90 and therefore SNFA run can be sometimes cheaper than AC run. This intimates that for some practical cases the SNFA is not so bad. Taking into account its simpler construction, the lower storage requirements and the dynamic ability it can be preferred in practical cases where time requirements have not the absolute priority.

References

[AC75] Aho, A. V. - Corasick, J. M.: Efficient string matching: An aid to bibliographic search, CACM, 18, June 1975, no. 6, pp. 333-340.

[AYS88] Aoe, J. - Yasutome, S. - Sato, T.: An efficient digital search algorithm by using a double-array structure, IEEE 1988, pp. 472-479.

[Blo89] Bloch, M.: Optimalizace vyhledávání vzorku v textových řetězech (Optimization of pattern retrieval in text strings), Doctoral thesis, Dept. of Comp. Sci. and Eng., Fac. of Electrical Eng., Czech Tech. Univ., Prague 1989, p. 79, (in Czech).

[Hla96] Hladík, J.: Vyhledávací stroj AC/Aoe (Retrieval machine AC/Aoe.), Diploma thesis, Dept. of Comp. Sci. and Eng., Fac. of Electrical Eng., Czech Tech. Univ., Prague 1996, p.136, (in Czech).