Combinatorial Algorithms on Words

Edited by
Alberto Apostolico

Zvi Galil

Combinatorial Algorithms on Words

Edited by

Alberto Apostolico

Department of Computer Sciences, Mathematical Sciences Building
West Lafayette, IN 47906/USA

Zvi Galil

Department of Computer Science, Columbia University
New York, NY 10027/USA

Springer-Verlag Berlin Heidelberg New York Tokyo
Published in cooperation with NATO Scientific Affairs Division

Proceedings of the NATO Advanced Research Workshop on Combinatorial Algorithms on Words held at Maratea, Italy, June 18–22, 1984

ISBN 3-540-15227-X Springer-Verlag Berlin Heidelberg New York Tokyo
ISBN 0-387-15227-X Springer-Verlag New York Heidelberg Berlin Tokyo

Library of Congress Cataloging in Publication Data.
NATO Advanced Research Workshop on Combinatorial Algorithms on Words (1984: Maratea, Italy) Combinatorial algorithms on words. (NATO ASI series. Series F, Computer and system sciences; vol. 12) "Proceedings of the NATO Advanced Research Workshop on Combinatorial Algorithms on Words held at Maratea, Italy, June 18–22, 1984"—T.p. verso. 1. Combinatorial analysis—Congresses. 2. Algorithms—Congresses. 2. Word problems (Mathematics)—Congresses. I. Apostolico, Alberto, 1948-. II. Galil, Zvi. III. Title. IV. Series: NATO ASI series. Series F, Computer and system sciences; no. 12. QA164.N35 1984 511'.6 85-8023
ISBN 0-387-15227-X (U.S.)

Printed in Germany

Printing: Beltz Offsetdruck, Hemsbach; Bookbinding: J. Schäffer OHG, Grünstadt
2145/3140-543210

PREFACE

Combinatorial Algorithms on Words refers to the collection of manipulations of strings of symbols (words) - not necessarily from a finite alphabet - that exploit the combinatorial properties of the logical/physical input arrangement to achieve efficient computational performances. The model of computation may be any of the established serial paradigms (e.g. RAM's, Turing Machines), or one of the emerging parallel models (e.g. PRAM , WRAM, Systolic Arrays, CCC).

This book focuses on some of the accomplishments of recent years in such disparate areas as pattern matching, data compression, free groups, coding theory, parallel and VLSI computation, and symbolic dynamics; these share a common flavor, yet have not been examined together in the past. In addition to being theoretically interesting, these studies have had significant applications. It happens that these works have all too frequently been carried out in isolation, with contributions addressing similar issues scattered throughout a rather diverse body of literature. We felt that it would be advantageous to both current and future researchers to collect this work in a single reference.

It should be clear that the book's emphasis is on aspects of combinatorics and complexity rather than logic, foundations, and decidability. In view of the large body of research and the degree of unity already achieved by studies in the theory of automata and formal languages, we have allocated very little space to them.

The material was divided, perhaps somewhat arbitrarily, into six sections. We encouraged several prominent scholars to provide overviews of their specific subfields. With its sizeable bibliography, the book seems well suited to serve as a reference text for a graduate course or seminar. Although there are no exercise sections, many open problems are proposed. Some of these may well serve as topics for term projects - a few may even blossom into theses.

Most of the papers contained in this volume originated as lectures delivered at the Workshop on Combinatorial Algorithms on Words, which was held in Maratea, Italy during the week of June 18-22, 1984. This workshop brought together researchers who had been active in the area so that a unified core of knowledge could be identified, and recommendations for future work could be outlined. The workshop was deliberately kept small and informal, and provided a congenial environment for lively discussions and valuable presentations.

The Maratea Workshop was sponsored by NATO under the Scientific Affair Division ARW Program, and it benefitted from the joint sponsorship of the University of Salerno and IBM Italia. Renato Capocelli, Mena De Santis, Dominique Perrin and Joel Seiferas joined us on the Program Committee for the Workshop; we thank them for their invaluable help. Mena De Santis did an excellent job of looking after the countless details of local arrangements. Finally, we would like to express our sincere gratitude to all the participants in the workshop.

New York City, December 1984 *A. Apostolico and Z. Galil*

Contents

Open Problems in Stringology

Zvi Galil*

Department of Computer Science
Columbia University
and
Tel-Aviv University

Abstract: Several open problems concerning combinatorial algorithms on strings are described.

0. Introduction

Every problem in theoretical computer science can be stated as a problem on strings (e.g. P = NP?). In this paper we restrict attention to combinatorial algorithms on strings. We list several open problems. We divide them into four groups: string matching, generalizations of string matching, index construction and miscellaneous. This list is far from being exhaustive.

In this paper Σ is a finite alphabet and all strings are in Σ^*. Sometimes we add special symbols \$ and # not in Σ. For a string x, x_i is the i-th symbol of x, $|x|$ is the length of x and x^R is the string x reversed. We say that x occurs in y if $x = y_{i+1} \cdots y_{i+|x|}$ for some i.

The string matching problem is the following: given two strings, the pattern and the text, find all the occurrences of the pattern in the text. About half of the open problems in this paper are about the string matching problem or its generalizations.

Our model of computation is the random access machine (RAM) with uniform cost (see [2]). Each register will typically store a symbol of Σ or an address.

1. Questions about String Matching

String matching is one of the most extensively studied problems in theoretical computer science. We briefly sketch the history of the problem. The problem was solved in linear time by the Knuth Morris and Pratt algorithm (KMP in short) [19], and then several versions of the problem were solved in real time, even by a Turing machine [12]. Another linear-time algorithm [6], (see also [11]) was designed that is sublinear in the average, assuming the text is already stored in memory [28]. Then attention was given to saving space while maintaining the optimality of the time complexity. This resulted in a linear-time (real-time) algorithm on a RAM

*This work was supported in part by National Foundation Grant MCS-8303139.

NATO ASI Series, Vol. F12
Combinatorial Algorithms on Words
Edited by A. Apostolico and Z. Galil

which uses only five (six) registers which store pointers to the input [15]. A simple probabilistic linear-time, constant-space algorithm was also designed [18].

A study of more theoretical nature followed [15]. String matching can be done by a Turing machine in linear time (or even real time) and logarithmic space. Moreover, a six head (eight head) two-way deterministic finite automaton (dfa) can perform string matching in linear time (real time). This study tries to identify the weakest computation model that can do string matching in optimal time and space.

Question 1: Can a one-way multihead dfa perform string matching?

More specifically, can such an automaton recognize the language $\{x\$uxv|u, x, v \in \Sigma^*\}$. Obviously a one head dfa cannot do the job. It was shown in [21] that a two-head dfa cannot do it either. (For the latter, there is a short alternative proof using Kolmogorov complexity.) It has been further claimed in [20] that a three-head one-way dfa cannot do string matching. Note that without loss of generality a one-way multihead dfa always halts, and it must do so in linear time. We believe that in fact a one-way multihead dfa cannot perform string matching.

Question 2: The number of states in a "Boyer-Moore dfa".

The original Boyer-Moore algorithm [6], BM in short, requires quadratic time in the worst case. The reason is that the BM "forgets" the part of the text it has seen when it slides the pattern after a mismatch. Consequently, many comparisons made by the BM are redundant, since the outcomes of these comparisons have already been established. Knuth [19] suggested using a dfa that will "remember" those parts of the pattern that need not be compared. The question is to find the exact number, or alternatively close upper and lower bounds on the number of states such a dfa must have.

An obvious upper bound is $2^{|x|}$. In [19], it is explained why a pattern x consisting of $|x|$ distinct symbols requires $\Omega(|x|^2)$ states. An $\Omega(|x|^3)$ lower bound for a pattern over a three letter alphabet is known [16]. The challenge is to narrow the still large gap.

This question is of theoretical interest only. If we use only two states and do not remember everything, linear time suffices for the BM [11]; if $|y| - |x|$ states are used, then the resulting algorithm requires at most $2|y| - |x|$ comparisons [3]. Recall that the KMP requires a similar number of comparisons, but with a smaller overhead.

Question 3: Parallel algorithms.

Recently, optimal parallel algorithms (those with $pt = O(n)$, where p = number of processors, t = time) were developed for string matching for a wide range of values of p: on the WRAM for $p \leq n/\log n$ and on the PRAM (for $p \leq n/\log^2 n$), first for fixed size alphabet [13], then for any alphabet [26]. (Recall that a WRAM [PRAM] is a collection of RAM's that share a common memory and are allowed simultaneous reads and writes [only simultaneous reads]. In the case of WRAM's, there is in addition a mechanism for resolving write conflicts.) Is it possible to design optimal parallel algorithms with a larger number of processors (e.g. $p \leq n$ on the WRAM, $p \leq n/\log n$ on the PRAM)? The algorithm in [26] needs concurrent writes only in preprocessing the pattern x but not during the search. Hence, the question for

PRAM is answered affirmatively if preprocessing is not included. Another question is whether it is possible to design optimal parallel algorithms on the more realistic model of a network of processors of constant degree.

2. Generalizations of String Matching

Question 4: A membership test for regular expressions.

Given a regular expression α over the alphabet Σ with operators $\bigcup$, $\cdot$, * (union, concatenation and Kleene star) and a string x, test whether $x \in L(\alpha)$ (the language described by α).

The obvious algorithm converts α to a nondeterministic finite automaton (nfa) A in linear time; then inductively finds the set of states that A can be in after reading x_i for $i = 1, ..., |x|$. A similar algorithm works directly on α by inductively finding all places in α we can be in after reading x_i. (This construction is known as the dot construction.) The time bound of this algorithm is $O(|x||\alpha|)$. The open problem is whether this time bound can be improved. If $|\alpha|$ is fixed or very small the answer is positive. A time bound of $O(|x| + 2^{|\alpha|})$ can easily be obtained by converting A to a dfa first.

For two important special cases the problem can be solved in linear time ($O(|x| + |\alpha|)$).

(1) $\alpha = \Sigma^* u \Sigma^*$, where $u \in \Sigma^*$ and

(2) $\alpha = \Sigma^*(u^1 \bigcup u^2 \bigcup ... \bigcup u^k)\Sigma^*$, where $u^i \in \Sigma^*$.

These are the single pattern and multi-pattern string matching problems, respectively; in the first we ask whether u occurs in x and in the second whether one of the u^i's occurs in x. These linear-time algorithms [19,1] might seem encouraging. However, these special cases are solved efficiently because in both cases α can be converted in linear time to a dfa (of linear size) which is impossible in the general case.

Question 5: String matching with don't-cares.

We add to the alphabet Σ a new "don't-care" symbol ϕ which matches any single symbol. Given two strings x, y over $(\Sigma \bigcup \{\phi\})^*$ we want to find all occurrences of x in y. There is an occurrence of x at position $i + 1$ of y if, for all $j = 1, ..., |x|$ whenever x_j and y_{i+j} are not ϕ, they are equal. Observe that all known string matching algorithms simply do not work if we allow "mistakes" in the form of don't-cares or as in the next problem.

In [9] the problem was reduced to that of integer multiplication. If an algorithm multiplies two binary numbers of n and m bits in time $T(n, m)$, then the time bound for our problem is $O(T(|x|, |y|) \log |x| \log |\Sigma|)$. Using the multiplication algorithm that is currently asymptotically best, we obtain a bound of $O(|y| \log^2 |x| \log \log |x| \log |\Sigma|)$.

The open problem is whether we can do better. Perhaps we can do better in the case that the don't-care symbols do not appear in the text (y). This case is considered in [24]. Of course, in case of a small alphabet and a small (constant)

number of don't-cares we can solve all the corresponding exact string matching problems. Note that the related problem of string matching with don't-care symbols in the pattern that can match an arbitrary string is easy: just treat each maximal pattern fragment separately.

Question 6: String matching with mistakes.

Given a pattern x and a text y and an integer k, find all occurrences of strings of length $|x|$ with distance at most k from x, where the distance between two equal size strings is defined as the number of positions in which they have different symbols.

Even for a small constant k we do not know how to solve the problem in time smaller that $O(|x||y|)$, which is an upper bound on the naive algorithm that computes the distance of x and each substring of y of length $|x|$.

The only known improvement seems to be in the case $k = 1$ [26]. A linear-time algorithm follows from a linear-time algorithm that finds for each position in the text the occurrence of the largest prefix of the pattern [22]. The latter is a modification of the KMP.

3. Index Construction

In efficient string matching we preprocess the pattern (or patterns) in linear time so we can search for it (or for them) in time linear in the length of the text. Another approach is to preprocess the text, to construct some kind of an index, and then use it to search for some pattern x (or answer some queries on x like finding the number of occurrences of x) in time linear in $|x|$. A number of linear-time algorithms are known, but all of them are closely related [7]. The first of these is the one discovered by Weiner [27].

Question 7: Can we construct an index in (almost) real time?

More specifically, can we read the text followed by one (or possibly more) pattern(s) one symbol at a time, spend a constant amount of time on each symbol, and identify occurrences immediately. This question can be rephrased as follows: Can we accept the language $L = \{uxv\$x\$|u, x, v \in \Sigma^*\}$ in real time. The word "almost" in Question 7 refers to the fact that we might still be constructing the index when we are already reading x.

One of the earliest results in computational complexity [17] implies that L cannot be accepted in real time by a multitape Turing machine, but Question 7 refers to RAM. It was shown in [17] that $L_1 = \{w_1\#w_2\#...\#w_k\$w^R|w_j \in \{0,1\}^*, k \geq 0, w = w_i \text{ for some } i\}$ cannot be accepted by a multitape Turing machine in real time, and the same proof implies that neither can $L_2 = \{w_1\#w_2\#...w_k\$w|w_j \in \{0,1\}^*, k \geq 0, w = w_i \text{ for some } i\}$. L_2 is a subset of L, and this is why the same is true for L. But L_1 and L_2 can be accepted easily in real time by a RAM [12]. Hence the results on Turing machines only show that the model is too weak.

A positive answer to Question 7 is claimed by Slisenko in [25] and in several of its earlier versions. In fact, the seventy-page solution claims to have characterized all periodicities of the text in real time. Unfortunately, so far I have not been able

to understand the solution. So, the question may be interpreted as follows: Can we find a reasonably simple real-time construction?

Question 8: Parallel Algorithms.

There are three interpretations to this question:
(1) a parallel construction of indices for sequential searches;
(2) a parallel construction of indices for parallel searches; or even
(3) a sequential construction of indices for parallel searches.
In all cases we would like to design efficient parallel algorithms. In the second and third cases we first have to find a good way to define a specific index.

Question 9: Dependence on the alphabet size.

Most of the algorithms for string matching (with a single pattern) do not depend on the size of the alphabet Σ. In fact Σ can be infinite. The only assumption needed is that two symbols can be compared in one unit of time.

All the algorithms for index construction do depend on the size of Σ. Every index is either a tree or a dfa with up to $|\Sigma|$ successors for every node. If we use an array of size $|\Sigma|$ for every node, then the index construction takes time (and space) $O(|y||\Sigma|)$, while the search takes time $O(|x|)$. If we use lists of successors, then the former takes time $O(|y||\Sigma|)$ (but space $O(|y|)$) and the latter takes time $O(|x||\Sigma|)$. If we use a search tree for the set of successors of each node, the former takes time $O(|y| \log |\Sigma|)$ and the latter $O(|x| \log |\Sigma|)$. Alternatively, we can use hashing. The problem is to determine the exact dependence on the alphabet size.

A similar problem is to determine the exact dependence on the alphabet size in the problem of multi-pattern string matching.

Question 10: Generalized Indices.

Can we efficiently construct indices that will (efficiently) support harder queries. An example of such a query is finding the maximal number of nonoverlapping occurrences of a given string. Counting all the occurrences is a simple application of the known indices. If we insist on nonoverlapping occurrences, the best algorithm [5] is neither simple nor linear time.

4. Miscellaneous Problems.

Question 11: Testing unique decipherability.

Given a set Γ of n strings $c_1, ..., c_n$ over Σ of total length L, is there a string in Γ^* that can be parsed in two different ways, or is $\bigcup_{i \neq j} (c_i \Gamma^* \bigcap c_j \Gamma^*) = \Phi$?

There are several algorithms for solving the problem (see for example [4]). They are essentially the same, and they solve the problem in time $O(nL)$ by constructing a certain graph in a search for a "counter example". The time bound follows immediately from the fact that the graph may have up to L vertices of degree up to n. This time bound is quadratic in the worst case and is linear only for $n =$ constant. Can we do better?

Question 12: Solving string problems with two-way deterministic pushdown automata.

Two-way deterministic pushdown automata (2dpda's) have been closely related to string matching. The linearity of string matching (and palindrome recognition) can be easily established by first showing that a 2dpda accepts the corresponding language, since Cook showed that membership for a 2dpda language can be determined in linear time by a RAM [8] (see also [10]). There are some linear-time recognizable string languages that we do not know how to recognize with a 2dpda. Two such examples are the following.

$$\text{PREFIXSQUARE} = \{wwu | w, u \in \Sigma^*\}, \text{and PALSTAR} = (\text{PAL})^*,$$

where $\text{PAL} = \{w | w \in \Sigma^*, w = w^R, |w| > 1\}$ is the language of nontrivial palindromes. (The related language $\{ww^R u | w, u \in \Sigma^*\}$ can be recognized by a 2dpda.) For a linear-time recognition of PALSTAR and other similar open problems, see [14].

Question 13: String problems on DNA.

There are many interesting problems that arise in the study of DNA sequences. Here we mention one such problem. We assume that for some of the pairs of symbols (in $\Sigma \times \Sigma$) there is an associated positive real number. The meaning of this number is that if we fold a string so that these two symbols touch the associated number represents the amount of energy that is released. Given a string, we want to find an optimal way to fold it (a way that maximizes the energy released). The problem can be stated as finding a planar matching (graph matching, not string matching) of maximal weight. The problem can be solved easily in time $O(n^3)$ using dynamic programming [23]. Can we do better?

Acknowledgement: Alberto Apostolico, Stuart Haber and Joel Seiferas read an earlier version of the paper and gave me many helpful suggestions.

5. References

1. A.V. Aho and M.J. Corasick, Efficient string matching: An aid to bibliographic search, *Communications of the ACM 18* (1975), 333-340.

2. A. Aho, J. Hopcroft and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA 1974.

3. A. Apostolico and R. Giancarlo, The Boyer-Moore-Galil string searching strategies revisited, *SIAM J. on Computing*, to appear.

4. A. Apostolico and R. Giancarlo, Pattern matching machine implementation of a fast test for unique decipherability, *Information Processing Letters 18* (1984), pp. 155-158.

5. A. Apostolico and F. Preparata, A structure for the statistics of all substrings in a text string with or without overlap, *Proc. 2nd World Conf. on Math. at the Service of Man*, 1982, pp. 104-109.

6. R.S. Boyer and J.S. Moore, A fast string searching algorithm, *Communications of the ACM 20* (1977), 762-772.

7. M.T. Chen and J.I. Seiferas, Efficient and elegant subword tree construction, *Combinatorial Algorithms on Words*, A. Apostolico and Z. Galil eds., Springer Verlag Lecture Notes, 1985.

8. S. Cook, Linear time simulation of deterministic two-way pushdown automata, *Proc. IFIP Congress* (1971), pp. 172-179.

9. M.J. Fischer and M.S. Paterson, String-matching and other products, in: *Complexity of Computation (SIAM-AMS Proceedings 7)*, R.M. Karp ed., American Mathematical Society, Providence, RI, 1974, pp. 113-125.

10. Z. Galil, Two fast simulations which imply some fast string matching and palindrome recognition algorithms, *Information Processing Letters 4* (1976), pp. 85-87.

11. Z. Galil, On improving the worst case running time of the Boyer-Moore string matching algorithm, *Communications of the ACM 22* (1979), 505-508.

12. Z. Galil, String matching in real time, *J. ACM 28* (1981), pp. 134-149.

13. Z. Galil, Optimal parallel algorithms for string matching, *Proc. 16th ACM Symposium on Theory of Computing*, 1984, pp. 240-248.

14. Z. Galil and J.I. Seiferas, A linear-time on-line recognition algorithm for "Palstar", *J. ACM 25* (1978), pp. 102-111.

15. Z. Galil and J.I. Seiferas, Time-space-optimal string matching, *J Computer and System Sciences 26* (1983), pp. 280-294.

16. L. Guibas and A. Odlyzko, private communication.

17. J. Hartmanis and R.E. Stearns, On the computational complexity of algorithms, *Transactions of the American Mathematical Society 117* (1965), 285-306.

18. R.M. Karp and M.O. Rabin, Efficient randomized pattern-matching algorithms, manuscript.

19. D.E. Knuth, J.H. Morris, Jr., and V.R. Pratt, Fast pattern matching in strings, *SIAM Journal on Computing 6* (1977), 323-350.

20. M. Li, Lower bound on string-matching, Technical Report TR-84-636, Department of Computer Science, Cornell University, 1984.

21. M. Li and Y. Yesha, String matching cannot be done by a two-head one-way deterministic finite automaton, Technical Report TR-83-579, Cornell University, 1983.

22. M.G. Main and R.J. Lorentz, An $O(n \log n)$ algorithm for finding all repetitions in a string, *J. of Algorithms* (1984), pp. 422-432.

23. R. Nussinov, G. Pieczenik, J.R. Griggs, and D.J. Kleitman, Algorithms for loop matchings, *SIAM J. of Applied Math 35* (1978), pp. 68-82.

24. R. Pinter, Efficient string matching with don't-care patterns, *Combinatorial Algorithms on Words*, A. Apostolico and Z. Galil eds., Springer Verlag Lecture Notes, 1985.

25. A.O. Slisenko, Detection of periodicities and string-matching in real time, *J. of Soviet Mathematics 22*, Plenum Publishing Co. (1983), pp. 1316-1386.

26. U. Vishkin, Private communication.

27. P. Weiner, Linear pattern matching algorithms, *Proc. 14th IEEE Annual Symposium on Switching and Automata Theory*, 1973, pp. 1-11.

28. A.C.C. Yao, The complexity of pattern matching for a random string, *SIAM J. on Comput. 8* (1979), pp. 368-387.

EFFICIENT STRING MATCHING WITH DON'T-CARE PATTERNS*

Ron Y. Pinter
IBM Israel Scientific Center
Technion City
Haifa 32000, ISRAEL

ABSTRACT

The occurrences of a constant pattern in a given text string can be found in linear time using the famous algorithm of Knuth, Morris, and Pratt [KMP]. Aho and Corasick [AC] independently solved the problem for patterns consisting of a set of strings, where the occurrence of one member is considered a match. Both algorithms preprocess the pattern so that the text can be searched efficiently. This paper considers the extension of their methods to deal with patterns involving more expressive descriptions, such as don't-care (wild-card) symbols, complements, etc. Such extensions are useful in the context of clever text-editors and the analysis of chemical compounds.

The main result of this paper is an algorithm to deal efficiently with patterns containing a definite number of don't-care symbols. Our method is to collect "evidence" about the occurrences of the constant parts of the pattern in the text, using the algorithm of Aho and Corasick [AC]. We arrange the consequences of the intermediate results of the search in an array of small counters whose length is equal to that of the pattern. As soon as a match for the whole pattern is found, it is reported. If we assume that the counters can be incremented in parallel, the overall (time and space) complexity of the algorithm remains linear. Otherwise, the worst-case time complexity becomes quadratic, without changing the space requirements.

We include here a discussion of why alternative ways to solve the problem, especially those trying to preserve the purely automaton-driven constructions of [KMP] and [AC], do not work. Finally, we describe a minor extension to an algorithm of Fischer and Paterson [FP]. Originally, it could deal with don't-cares in the text; now it can also handle complements of single characters within the same computational complexity.

---

* This work was supported in part by the National Science Foundation, U.S.A., under grant No. MCS78-05849, and by a graduate fellowship from the Hebrew Technical Institute, New York, N.Y.

NATO ASI Series, Vol. F12
Combinatorial Algorithms on Words
Edited by A. Apostolico and Z. Galil