Amihood Amir
Gad M. Landau (Eds.)

LNCS 2089

Combinatorial Pattern Matching

12th Annual Symposium, CPM 2001
Jerusalem, Israel, July 2001
Proceedings

Springer

Amihood Amir Gad M. Landau (Eds.)

Combinatorial Pattern Matching

12th Annual Symposium, CPM 2001
Jerusalem, Israel, July 1-4, 2001
Proceedings

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Amihood Amir
Bar-Ilan University, Department of Computer Science
52900 Ramat-Gan, Israel; E-mail: amir@cs.biu.ac.il
and
Georgia Tech, Atlanta, Georgia 30332-0280, USA

Gad M. Landau
University of Haifa, Department of Computer Science
31905 Haifa, Israel; E-mail: landau@cs.haifa.ac.il
and
Polytechnic University, Brooklyn, NY 11201, USA

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Combinatorial pattern matching : 12th annual symposium ; proceedings / CPM 2001, Jerusalem, Israel, July 1 - 4, 2001. Amihood Amir ; Gad. M. Landau (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 2001
(Lecture notes in computer science ; Vol. 2089)
ISBN 3-540-42271-4

CR Subject Classification (1998): F.2.2, I.5.4, I.5.0, I.7.3, H.3.3, E.4, G.2.1

ISSN 0302-9743
ISBN 3-540-42271-4 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Christian Grosche, Hamburg
Printed on acid-free paper SPIN: 10839362 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 2089

Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Lecture Notes in Computer Science

For information about Vols. 1–1998
please contact your bookseller or Springer-Verlag

Vol. 1999: W. Emmerich, S. Tai (Eds.), Engineering Distributed Objects. Proceedings, 2000. VIII, 271 pages. 2001.

Vol. 2000: R. Wilhelm (Ed.), Informatics: 10 Years Back, 10 Years Ahead. IX, 369 pages. 2001.

Vol. 2001: G.A. Agha, F. De Cindio, G. Rozenberg (Eds.), Concurrent Object-Oriented Programming and Petri Nets. VIII, 539 pages. 2001.

Vol. 2002: H. Comon, C. Marché, R. Treinen (Eds.), Constraints in Computational Logics. Proceedings, 1999. XII, 309 pages. 2001.

Vol. 2003: F. Dignum, U. Cortés (Eds.), Agent Mediated Electronic Commerce III. XII, 193 pages. 2001. (Subseries LNAI).

Vol. 2004: A. Gelbukh (Ed.), Computational Linguistics and Intelligent Text Processing. Proceedings, 2001. XII, 528 pages. 2001.

Vol. 2006: R. Dunke, A. Abran (Eds.), New Approaches in Software Measurement. Proceedings, 2000. VIII, 245 pages. 2001.

Vol. 2007: J.F. Roddick, K. Hornsby (Eds.), Temporal, Spatial, and Spatio-Temporal Data Mining. Proceedings, 2000. VII, 165 pages. 2001. (Subseries LNAI).

Vol. 2009: H. Federrath (Ed.), Designing Privacy Enhancing Technologies. Proceedings, 2000. X, 231 pages. 2001.

Vol. 2010: A. Ferreira, H. Reichel (Eds.), STACS 2001. Proceedings, 2001. XV, 576 pages. 2001.

Vol. 2011: M. Mohnen, P. Koopman (Eds.), Implementation of Functional Languages. Proceedings, 2000. VIII, 267 pages. 2001.

Vol. 2012: D.R. Stinson, S. Tavares (Eds.), Selected Areas in Cryptography. Proceedings, 2000. IX, 339 pages. 2001.

Vol. 2013: S. Singh, N. Murshed, W. Kropatsch (Eds.), Advances in Pattern Recognition – ICAPR 2001. Proceedings, 2001. XIV, 476 pages. 2001.

Vol. 2014: M. Moortgat (Ed.), Logical Aspects of Computational Linguistics. Proceedings, 1998. X, 287 pages. 2001. (Subseries LNAI).

Vol. 2015: D. Won (Ed.), Information Security and Cryptology – ICISC 2000. Proceedings, 2000. X, 261 pages. 2001.

Vol. 2016: S. Murugesan, Y. Deshpande (Eds.), Web Engineering. IX, 357 pages. 2001.

Vol. 2018: M. Pollefeys, L. Van Gool, A. Zisserman, A. Fitzgibbon (Eds.), 3D Structure from Images – SMILE 2000. Proceedings, 2000. X, 243 pages. 2001.

Vol. 2019: P. Stone, T. Balch, G. Kraetzschmar (Eds.), RoboCup 2000: Robot Soccer World Cup IV. XVII, 658 pages. 2001. (Subseries LNAI).

Vol. 2020: D. Naccache (Ed.), Topics in Cryptology – CT-RSA 2001. Proceedings, 2001. XII, 473 pages. 2001

Vol. 2021: J. N. Oliveira, P. Zave (Eds.), FME 2001: Formal Methods for Increasing Software Productivity. Proceedings, 2001. XIII, 629 pages. 2001.

Vol. 2022: A. Romanovsky, C. Dony, J. Lindskov Knudsen, A. Tripathi (Eds.), Advances in Exception Handling Techniques. XII, 289 pages. 2001

Vol. 2024: H. Kuchen, K. Ueda (Eds.), Functional and Logic Programming. Proceedings, 2001. X, 391 pages. 2001.

Vol. 2025: M. Kaufmann, D. Wagner (Eds.), Drawing Graphs. XIV, 312 pages. 2001.

Vol. 2026: F. Müller (Ed.), High-Level Parallel Programming Models and Supportive Environments. Proceedings, 2001. IX, 137 pages. 2001.

Vol. 2027: R. Wilhelm (Ed.), Compiler Construction. Proceedings, 2001. XI, 371 pages. 2001.

Vol. 2028: D. Sands (Ed.), Programming Languages and Systems. Proceedings, 2001. XIII, 433 pages. 2001.

Vol. 2029: H. Hussmann (Ed.), Fundamental Approaches to Software Engineering. Proceedings, 2001. XIII, 349 pages. 2001.

Vol. 2030: F. Honsell, M. Miculan (Eds.), Foundations of Software Science and Computation Structures. Proceedings, 2001. XII, 413 pages. 2001.

Vol. 2031: T. Margaria, W. Yi (Eds.), Tools and Algorithms for the Construction and Analysis of Systems. Proceedings, 2001. XIV, 588 pages. 2001.

Vol. 2032: R. Klette, T. Huang, G. Gimel'farb (Eds.), Multi-Image Analysis. Proceedings, 2000. VIII, 289 pages. 2001.

Vol. 2033: J. Liu, Y. Ye (Eds.), E-Commerce Agents. VI, 347 pages. 2001. (Subseries LNAI).

Vol. 2034: M.D. Di Benedetto, A. Sangiovanni-Vincentelli (Eds.), Hybrid Systems: Computation and Control. Proceedings, 2001. XIV, 516 pages. 2001.

Vol. 2035: D. Cheung, G.J. Williams, Q. Li (Eds.), Advances in Knowledge Discovery and Data Mining – PAKDD 2001. Proceedings, 2001. XVIII, 596 pages. 2001. (Subseries LNAI).

Vol. 2037: E.J.W. Boers et al. (Eds.), Applications of Evolutionary Computing. Proceedings, 2001. XIII, 516 pages. 2001.

Vol. 2038: J. Miller, M. Tomassini, P.L. Lanzi, C. Ryan, A.G.B. Tettamanzi, W.B. Langdon (Eds.), Genetic Programming. Proceedings, 2001. XI, 384 pages. 2001.

Vol. 2039: M. Schumacher, Objective Coordination in Multi-Agent System Engineering. XIV, 149 pages. 2001. (Subseries LNAI).

Vol. 2040: W. Kou, Y. Yesha, C.J. Tan (Eds.), Electronic Commerce Technologies. Proceedings, 2001. X, 187 pages. 2001.

Vol. 2041: I. Attali, T. Jensen (Eds.), Java on Smart Cards: Programming and Security. Proceedings, 2000. X, 163 pages. 2001.

Vol. 2042: K.-K. Lau (Ed.), Logic Based Program Synthesis and Transformation. Proceedings, 2000. VIII, 183 pages. 2001.

Vol. 2043: D. Craeynest, A. Strohmeier (Eds.), Reliable Software Technologies – Ada-Europe 2001. Proceedings, 2001. XV, 405 pages. 2001.

Vol. 2044: S. Abramsky (Ed.), Typed Lambda Calculi and Applications. Proceedings, 2001. XI, 431 pages. 2001.

Vol. 2045: B. Pfitzmann (Ed.), Advances in Cryptology – EUROCRYPT 2001. Proceedings, 2001. XII, 545 pages. 2001.

Vol. 2047: R. Dumke, C. Rautenstrauch, A. Schmietendorf, A. Scholz (Eds.), Performance Engineering. XIV, 349 pages. 2001.

Vol. 2048: J. Pauli, Learning Based Robot Vision. IX, 288 pages. 2001.

Vol. 2051: A. Middeldorp (Ed.), Rewriting Techniques and Applications. Proceedings, 2001. XII, 363 pages. 2001.

Vol. 2052: V.I. Gorodetski, V.A. Skormin, L.J. Popyack (Eds.), Information Assurance in Computer Networks. Proceedings, 2001. XIII, 313 pages. 2001.

Vol. 2053: O. Danvy, A. Filinski (Eds.), Programs as Data Objects. Proceedings, 2001. VIII, 279 pages. 2001.

Vol. 2054: A. Condon, G. Rozenberg (Eds.), DNA Computing. Proceedings, 2000. X, 271 pages. 2001.

Vol. 2055: M. Margenstern, Y. Rogozhin (Eds.), Machines, Computations, and Universality. Proceedings, 2001. VIII, 321 pages. 2001.

Vol. 2056: E. Stroulia, S. Matwin (Eds.), Advances in Artificial Intelligence. Proceedings, 2001. XII, 366 pages. 2001. (Subseries LNAI).

Vol. 2057: M. Dwyer (Ed.), Model Checking Software. Proceedings, 2001. X, 313 pages. 2001.

Vol. 2059: C. Arcelli, L.P. Cordella, G. Sanniti di Baja (Eds.), Visual Form 2001. Proceedings, 2001. XIV, 799 pages. 2001.

Vol. 2060: T. Böhme, H. Unger (Eds.), Innovative Internet Computing Systems. Proceedings, 2001. VIII, 183 pages. 2001.

Vol. 2062: A. Nareyek, Constraint-Based Agents. XIV, 178 pages. 2001. (Subseries LNAI).

Vol. 2064: J. Blanck, V. Brattka, P. Hertling (Eds.), Computability and Complexity in Analysis. Proceedings, 2000. VIII, 395 pages. 2001.

Vol. 2065: H. Balster, B. de Brock, S. Conrad (Eds.), Database Schema Evolution and Meta-Modeling. Proceedings, 2000. X, 245 pages. 2001.

Vol. 2066: O. Gascuel, M.-F. Sagot (Eds.), Computational Biology. Proceedings, 2000. X, 165 pages. 2001.

Vol. 2068: K.R. Dittrich, A. Geppert, M.C. Norrie (Eds.), Advanced Information Systems Engineering. Proceedings, 2001. XII, 484 pages. 2001.

Vol. 2070: L. Monostori, J. Váncza, M. Ali (Eds.), Engineering of Intelligent Systems. Proceedings, 2001. XVIII, 951 pages. 2001. (Subseries LNAI).

Vol. 2071: R. Harper (Ed.), Types in Compilation. Proceedings, 2000. IX, 207 pages. 2001.

Vol. 2072: J. Lindskov Knudsen (Ed.), ECOOP 2001 – Object-Oriented Programming. Proceedings, 2001. XIII, 429 pages. 2001.

Vol. 2073: V.N. Alexandrov, J.J. Dongarra, B.A. Juliano, R.S. Renner, C.J.K. Tan (Eds.), Computational Science – ICCS 2001. Part I. Proceedings, 2001. XXVIII, 1306 pages. 2001.

Vol. 2074: V.N. Alexandrov, J.J. Dongarra, B.A. Juliano, R.S. Renner, C.J.K. Tan (Eds.), Computational Science – ICCS 2001. Part II. Proceedings, 2001. XXVIII, 1076 pages. 2001.

Vol. 2075: J.-M. Colom, M. Koutny (Eds.), Applications and Theory of Petri Nets 2001. Proceedings, 2001. XII, 403 pages. 2001.

Vol. 2077: V. Ambriola (Ed.), Software Process Technology. Proceedings, 2001. VIII, 247 pages. 2001.

Vol. 2078: R. Reed, J. Reed (Eds.), SDL 2001: Meeting UML. Proceedings, 2001. XI, 439 pages. 2001.

Vol. 2081: K. Aardal, B. Gerards (Eds.), Integer Programming and Combinatorial Optimization. Proceedings, 2001. XI, 423 pages. 2001.

Vol. 2082: M.F. Insana, R.M. Leahy (Eds.), Information Processing in Medical Imaging. Proceedings, 2001. XVI, 537 pages. 2001.

Vol. 2083: R. Goré, A. Leitsch, T. Nipkow (Eds.), Automated Reasoning. Proceedings, 2001. XV, 708 pages. 2001. (Subseries LNAI).

Vol. 2084: J. Mira, A. Prieto (Eds.), Connectionist Models of Neurons, Learning Processes, and Artificial Intelligence. Proceedings, 2001. Part I. XXVII, 836 pages. 2001.

Vol. 2085: J. Mira, A. Prieto (Eds.), Bio-Inspired Applications of Connectionism. Proceedings, 2001. Part II. XXVII, 848 pages. 2001.

Vol. 2089: A. Amir, G.M. Landau (Eds.), Combinatorial Pattern Matching. Proceedings, 2001. VIII, 273 pages. 2001.

Vol. 2091: J. Bigun, F. Smeraldi (Eds.), Audio- and Video-Based Biometric Person Authentication. Proceedings, 2001. XIII, 374 pages. 2001.

Vol. 2092: L. Wolf, D. Hutchison, R. Steinmetz (Eds.), Quality of Service – IWQoS 2001. Proceedings, 2001. XII, 435 pages. 2001.

Vol. 2096: J. Kittler, F. Roli (Eds.), Multiple Classifier Systems. Proceedings, 2001. XII, 456 pages. 2001.

Vol. 2097: B. Read (Ed.), Advances in Databases. Proceedings, 2001. X, 219 pages. 2001.

Vol. 2099: P. de Groote, G. Morrill, C. Retoré (Eds.), Logical Aspects of Computational Linguistics. Proceedings, 2001. VIII, 311 pages. 2001. (Subseries LNAI).

Vol. 2110: B. Hertzberger, A. Hoekstra, R. Williams (Eds.), High-Performance Computing and Networking. Proceedings, 2001. XVII, 733 pages. 2001.

Foreword

The papers contained in this volume were presented at the 12th Annual Symposium on Combinatorial Pattern Matching, held July 1–4, 2001 at the *Dan Panorama Hotel* in Jerusalem, Israel. They were selected from 35 abstracts submitted in response to the call for papers. In addition, there were invited lectures by Aviezri Fraenkel (*Weizmann Institute of Science*), Zvi Galil (*Columbia*), Rao Kosaraju (*Johns Hopkins University*), and Uzi Vishkin (*Technion and U. Maryland*). This year the call for papers invited short (poster) presentations. They also appear in the proceedings.

Combinatorial Pattern Matching (CPM) addresses issues of searching and matching strings and more complicated patterns such as trees, regular expressions, graphs, point sets, and arrays, in various formats. The goal is to derive nontrivial combinatorial properties of such structures and to exploit these properties in order to achieve superior performance for the corresponding computational problems. On the other hand, an important aim is to analyze and pinpoint the properties and conditions under which searches can not be performed efficiently.

Over the past decade a steady flow of high quality research on this subject has changed a sparse set of isolated results into a full-fledged area of algorithmics. This area is continuing to grow even further due to the increasing demand for speed and efficiency that stems from important applications such as the World Wide Web, computational biology, computer vision, and multimedia systems. These involve requirements for information retrieval in heterogeneous databases, data compression, and pattern recognition. The objective of the annual CPM gathering is to provide an international forum for the presentation of research results in combinatorial pattern matching and related applications.

The first 11 meetings were held in Paris, London, Tucson, Padova, Asilomar, Helsinki, Laguna Beach, Aarhus, Piscataway, Warwick, and Montreal, over the years 1990–2000. After the first meeting, a selection of papers appeared as a special issue of *Theoretical Computer Science* in volume 92. The proceedings of the 3rd to 11th meetings appeared as volumes 644, 684, 807, 937, 1075, 1264, 1448, 1645, and 1848 of the Springer LNCS series. Selected papers of the 12th meeting will appear in a special issue of *Discrete Applied Mathematics.*

The general organization and orientation of the CPM conferences is coordinated by a steering committee composed of Alberto Apostolico (*Padova and Purdue*), Maxime Crochemore (*Marne-la-Vallée*), Zvi Galil (*Columbia*) and Udi Manber (*Yahoo!*).

April 2001

Amihood Amir
Gad M. Landau

Program Committee

Amihood Amir, co-chair,
Bar-Ilan & Georgia Tech
Setsuo Arikawa, *Kyushu*
Gary Benson, *Mt. Sinai*
Andrei Broder, *Altavista*
Maxime Crochemore, *Marne-la-Vallée*
Leszek Gasieniec, *Liverpool*
Raffaele Giancarlo, *Palermo*
Costas S. Iliopoulos,
King's College, London
Tomi Klein, *Bar-Ilan*
Gad Landau, co-chair,
Haifa & Polytechnic, New York
Thierry Lecroq, *Rouen*
Moshe Lewenstein, *IBM Yorktown*
Yoelle Maarek, *IBM Haifa*
Kunsoo Park, *Seoul National*
Hershel Safer, *Compugen*
David Sankoff, *Montréal*
Jeanette Schmidt, *Incyte*
Dafna Sheinwald, *IBM Haifa*
Divesh Srivastava, *AT&T*
Naftali Tishbi, *Hebrew University*
Ian Witten, *Waikato*

Local Organization

Local arrangements were made by the program committee co-chairs. The conference Web site was created and maintained by Revital Erez. Organizational help was provided by Gal Goldschmidt, Noa Gur, Libi Raz, and Daphna Stern.

Sponsoring Institutions

- The Caesarea Edmond Benjamin de Rothschild Foundation, Institute for Interdisciplinary Applications of Computer Science, Haifa University.
- Bar Ilan University.
- Haifa University.

List of Additional Reviewers

Jacques Desarmenien
Paolo Ferragina
Franya Franek
Dora Giammarresi
Jan Holub
Jesper Jansson
Dong Kyue Kim
Ralf Klasing
Andrzej Lingas
Gonzalo Navarro
Igor Potapov
Tomasz Radzik
Mathieu Raffinot
Rajeev Raman
Marie-France Sagot
Marinella Sciortino
Patrice Seebold
Ayumi Shinohara
Noam Slonim
W.F. Smyth
Dina Sokol
Shigeru Takano
Masayuki Takeda
Mutsunori Yagiura
Michal Ziv-Ukelson

Table of Contents

Regular Expression Searching over Ziv–Lempel Compressed Text

Gonzalo Navarro*

Dept. of Computer Science, University of Chile
Blanco Encalada 2120, Santiago, Chile
gnavarro@dcc.uchile.cl

Abstract. We present a solution to the problem of regular expression searching on compressed text. The format we choose is the Ziv–Lempel family, specifically the LZ78 and LZW variants. Given a text of length u compressed into length n, and a pattern of length m, we report all the R occurrences of the pattern in the text in $O(2^m + mn + Rm \log m)$ worst case time. On average this drops to $O(m^2 + (n + R) \log m)$ or $O(m^2 + n + Ru/n)$ for most regular expressions. This is the first nontrivial result for this problem. The experimental results show that our compressed search algorithm needs half the time necessary for decompression plus searching, which is currently the only alternative.

1 Introduction

The need to search for regular expressions arises in many text-based applications, such as text retrieval, text editing and computational biology, to name a few. A *regular expression* is a generalized pattern composed of (i) basic strings, (ii) union, concatenation and Kleene closure of other regular expressions [1]. The problem of regular expression searching is quite old and has received continuous attention since the sixties until our days (see Section 2.1).

A particularly interesting case of text searching arises when the text is compressed. Text compression [6] exploits the redundancies of the text to represent it using less space. There are many different compression schemes, among which the Ziv–Lempel family [32, 33] is one of the best in practice because of its good compression ratios combined with efficient compression and decompression times. The compressed matching problem consists of searching a pattern on a compressed text without uncompressing it. Its main goal is to search the compressed text faster than the trivial approach of decompressing it and then searching. This problem is important in practice. Today's textual databases are an excellent example of applications where both problems are crucial: the texts should be kept compressed to save space and I/O time, and they should be efficiently searched. Surprisingly, these two combined requirements are not easy to achieve together, as the only solution before the 90's was to process queries by uncompressing the texts and then searching into them.

* Partially supported by Fondecyt grant 1-990627.

A. Amir and G.M. Landau (Eds.): CPM 2001, LNCS 2089, pp. 1–17, 2001.

Since then, a lot of research has been conducted on the problem. A wealth of solutions have been proposed (see Section 2.2) to deal with simple, multiple and, very recently, approximate compressed pattern matching. Regular expression searching on compressed text seems to be the last goal which still defies the existence of any nontrivial solution.

This is the problem we solve in this paper: we present the first solution for compressed regular expression searching. The format we choose is the Ziv–Lempel family, focusing in the LZ78 and LZW variants [33, 29]. Given a text of length u compressed into length n, we are able to find the R occurrences of a regular expression of length m in $O(2^m + mn + Rm \log m)$ worst case time, needing $O(2^m + mn)$ space. We also propose two modifications which achieve $O(m^2 + (n+R)\log m)$ or $O(m^2 + n + Ru/n)$ average case time and, respectively, $O(m+n\log m)$ or $O(m+n)$ space, for "admissible" regular expressions, i.e. those whose automaton runs out of active states after reading $O(1)$ text characters. These results are achieved using bit-parallelism and are valid for short enough patterns, otherwise the search times have to be multiplied by $\lceil m/w \rceil$, where w is the number of bits in the computer word.

We have implemented our algorithm on LZW and compared it against the best existing algorithms on uncompressed text, showing that we can search the compressed text twice as fast as the naive approach of uncompressing and then searching.

2 Related Work

2.1 Regular Expression Searching

The traditional technique [26] to search a regular expression of length m (which means m letters, not counting the special operators such as "*", "|", etc.) in a text of length u is to convert the expression into a nondeterministic finite automaton (NFA) with $O(m)$ nodes. Then, it is possible to search the text using the automaton at $O(mu)$ worst case time. The cost comes from the fact that more than one state of the NFA may be active at each step, and therefore all may need to be updated.

On top of the basic algorithm for converting a regular expression into an NFA, we have to add a self-loop at the initial state which guarantees that it keeps always active, so it is able to detect a match starting anywhere in the text. At each text position where a final state gets active we signal the end point of an occurrence.

A more efficient choice [1] is to convert the NFA into a deterministic finite automaton (DFA), which has only one active state at a time and therefore allows searching the text at $O(u)$ cost, which is worst-case optimal. The cost of this approach is that the DFA may have $O(2^m)$ states, which implies a preprocessing cost and extra space exponential in m.

An easy way to obtain a DFA from an NFA is via *bit-parallelism*, which is a technique to code many elements in the bits of a single computer word and

manage to update all them in a single operation. In this case, the vector of active and inactive states is stored as the bits of a computer word. Instead of (ala Thompson [26]) examining the active states one by one, the whole computer word is used to index a table which, given the current text character, provides the new set of active states (another computer word). This can be considered either as a bit-parallel simulation of an NFA, or as an implementation of a DFA (where the identifier of each deterministic state is the bit mask as a whole). This idea was first proposed by Wu and Manber [31, 30].

Later, Navarro and Raffinot [23] used a similar procedure, this time using Glushkov's [7] construction of the NFA. This construction has the advantage of producing an automaton of exactly $m + 1$ states, while Thompson's may reach $2m$ states. A drawback is that the structure is not so regular and therefore a table $D : 2^{m+1} \times (\sigma+1) \rightarrow 2^{m+1}$ is required, where σ is the size of the pattern alphabet Σ. Thompson's construction, on the other hand, is more regular and only needs a table $D : 2^{2m} \rightarrow 2^{2m}$ for the ε-transitions. It has been shown [23] that Glushkov's construction normally yields faster search time. In any case, if the table is too big it can be split horizontally in two or more tables [31]. For example, a table of size 2^m can be split into 2 subtables of size $2^{m/2}$. We need to access two tables for a transition but need only the square root of the space.

Some techniques have been proposed to obtain a tradeoff between NFAs and DFAs. In 1992, Myers [19] presented a four-russians approach which obtains $O(mu/\log u)$ worst-case time and extra space. The idea is to divide the syntax tree of the regular expression into "modules", which are subtrees of a reasonable size. These subtrees are implemented as DFAs and are thereafter considered as leaf nodes in the syntax tree. The process continues with this reduced tree until a single final module is obtained.

The ideas presented up to now aim at a good implementation of the automaton, but they must inspect all the text characters. Other proposals try to skip some text characters, as it is usual for simple pattern matching. For example, Watson [28, chapter 5] presented an algorithm that determines the minimum length of a string matching the regular expression and forms a tree with all the prefixes of that length of strings matching the regular expression. A multipattern search algorithm like Commentz-Walter [8] is run over those prefixes as a filter to detect text areas where a complete occurrence may start. Another technique of this kind is used in *Gnu Grep 2.0*, which extracts a set of strings which must appear in any match. This string is searched for and the neighborhoods of its occurrences are checked for complete matches using a lazy deterministic automaton.

The most recent development, also in this line, is from Navarro and Raffinot [23]. They invert the arrows of the DFA and make all states initial and the initial state final. The result is an automaton that recognizes all the reverse prefixes of strings matching the regular expression. The idea is in this sense similar to that of Watson, but takes less space. The search method is also different: instead of a Boyer-Moore like algorithm, it is based on BNDM [23].

2.2 Compressed Pattern Matching

The *compressed matching problem* was first defined in the work of Amir and Benson [2] as the task of performing string matching in a compressed text without decompressing it. Given a text T, a corresponding compressed string $Z = z_1 \ldots z_n$, and a pattern P, the compressed matching problem consists in finding all occurrences of P in T, using only P and Z. A naive algorithm, which first decompresses the string Z and then performs standard string matching, takes time $O(m+u)$. An optimal algorithm takes worst-case time $O(m+n+R)$, where R is the number of matches (note that it could be that $R = u > n$).

Two different approaches exist to search compressed text. The first one is rather practical. Efficient solutions based on Huffman coding [10] on words have been presented by Moura et al. [18], but they need that the text contains natural language and is large (say, 10 Mb or more). Moreover, they allow only searching for whole words and phrases. There are also other practical ad-hoc methods [15], but the compression they obtain is poor. Moreover, in these compression formats $n = \Theta(u)$, so the speedups can only be measured in practical terms.

The second line of research considers Ziv–Lempel compression, which is based on finding repetitions in the text and replacing them with references to similar strings previously appeared. LZ77 [32] is able to reference any substring of the text already processed, while LZ78 [33] and LZW [29] reference only a single previous reference plus a new letter that is added.

String matching in Ziv–Lempel compressed texts is much more complex, since the pattern can appear in different forms across the compressed text. The first algorithm for exact searching is from 1994, by Amir, Benson and Farach [3], who search in LZ78 needing time and space $O(m^2 + n)$.

The only search technique for LZ77 is by Farach and Thorup [9], a randomized algorithm to determine in time $O(m + n \log^2(u/n))$ whether a pattern is present or not in the text.

An extension of the first work [3] to multipattern searching was presented by Kida et al. [13], together with the first experimental results in this area. They achieve $O(m^2 + n)$ time and space, although this time m is the total length of all the patterns.

New practical results were presented by Navarro and Raffinot [24], who proposed a general scheme to search on Ziv–Lempel compressed texts (simple and extended patterns) and specialized it for the particular cases of LZ77, LZ78 and a new variant proposed which was competitive and convenient for search purposes. A similar result, restricted to the LZW format, was independently found and presented by Kida et al. [14]. The same group generalized the existing algorithms and nicely unified the concepts in a general framework [12]. Recently, Navarro and Tarhio [25] presented a new, faster, algorithm based on Boyer-Moore.

Approximate string matching on compressed text aims at finding the pattern where a limited number of differences between the pattern and its occurrences are permitted. The problem, advocated in 1992 [2], had been solved for Huffman coding of words [18], but the solution is limited to search a whole word and retrieve whole words that are similar. The first true solutions appeared very

recently, by Kärkkäinen et al. [11], Matsumoto et al. [16] and Navarro et al. [22].

3 The Ziv–Lempel Compression Formats LZ78 and LZW

The general idea of Ziv–Lempel compression is to replace substrings in the text by a pointer to a previous occurrence of them. If the pointer takes less space than the string it is replacing, compression is obtained. Different variants over this type of compression exist, see for example [6]. We are particularly interested in the LZ78/LZW format, which we describe in depth.

The Ziv–Lempel compression algorithm of 1978 (usually named LZ78 [33]) is based on a dictionary of blocks, in which we add every new block computed. At the beginning of the compression, the dictionary contains a single block b_0 of length 0. The current step of the compression is as follows: if we assume that a prefix $T_{1...j}$ of T has been already compressed in a sequence of blocks $Z = b_1 \dots b_r$, all them in the dictionary, then we look for the longest prefix of the rest of the text $T_{j+1...u}$ which is a block of the dictionary. Once we found this block, say b_s of length ℓ_s, we construct a new block $b_{r+1} = (s, T_{j+\ell_s+1})$, we write the pair at the end of the compressed file Z, i.e $Z = b_1 \dots b_r b_{r+1}$, and we add the block to the dictionary. It is easy to see that this dictionary is prefix-closed (i.e. any prefix of an element is also an element of the dictionary) and a natural way to represent it is a tree.

We give as an example the compression of the word *ananas* in Figure 1. The first block is $(0, a)$, and next $(0, n)$. When we read the next a, a is already the block 1 in the dictionary, but an is not in the dictionary. So we create a third block $(1, n)$. We then read the next a, a is already the block 1 in the dictionary, but as do not appear. So we create a new block $(1, s)$.

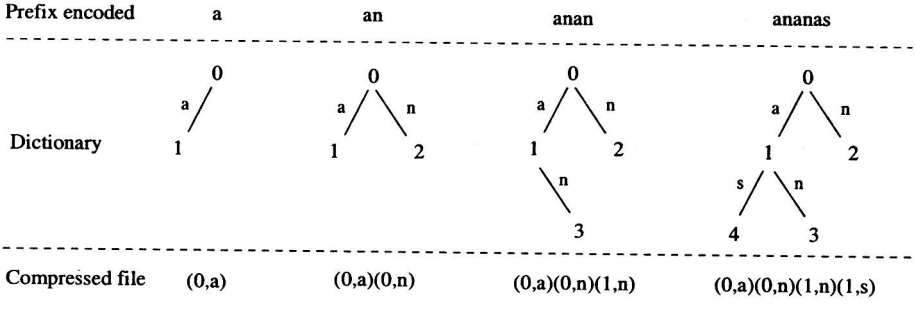

Fig. 1. Compression of the word *ananas* with the algorithm LZ78.

The compression algorithm is $O(u)$ time in the worst case and efficient in practice if the dictionary is stored as a tree, which allows rapid searching of the new text prefix (for each character of T we move once in the tree). The

decompression needs to build the same dictionary (the pair that defines the block r is read at the r-th step of the algorithm), although this time it is not convenient to have a tree, and an array implementation is preferable. Compared to LZ77, the compression is rather fast but decompression is slow.

Many variations on LZ78 exist, which deal basically with the best way to code the pairs in the compressed file, or with the best way to cope with limited memory for compression. A particularly interesting variant is from Welch, called LZW [29]. In this case, the extra letter (second element of the pair) is not coded, but it is taken as the first letter of the next block (the dictionary is started with one block per letter). LZW is used by Unix's *Compress* program.

In this paper we do not consider LZW separately but just as a coding variant of LZ78. This is because the final letter of LZ78 can be readily obtained by keeping count of the first letter of each block (this is copied directly from the referenced block) and then looking at the first letter of the next block.

4 A Search Algorithm

We present now our approach for regular expression searching over a text $Z = b_1 \ldots b_n$, that is expressed as a sequence of n *blocks*. Each block b_r represents a substring B_r of T, such that $B_1 \ldots B_n = T$. Moreover, each block B_r is formed by a concatenation of a previously seen block and an explicit letter. This comprises the LZ78 and LZW formats. Our goal is to find the positions in T where the pattern occurrences end, using Z.

Our approach is to modify the DFA algorithm based on bit-parallelism, which is designed to process T character by character, so that it processes T block by block using the fact that blocks are built from previous blocks and explicit letters. We assume that Glushkov's construction [7] is used, so the NFA has $m+1$ states. So we start by building the DFA in $O(2^m)$ time and space.

Our bit masks will denote sets of NFA states, so they will be of width $m+1$. For clarity we will write the sets of states, keeping in mind that we can compute $A \cup B$, $A \cap B$, A^c, $A = B$, $A \leftarrow B$, $a \in A$ in constant time (or, for long patterns, in $O(\lceil m/w \rceil)$ time, where w is the number of bits in the computer word). Another operation we will need to perform in constant time is to select any element of a set. This can be achieved with "bit magic", which means precomputing the table storing the position of, say, the highest bit for each possible bit mask of length $m+1$, which is not much given that we already store σ such tables.

About our automaton, we assume that the states are numbered $0 \ldots m$, being 0 the initial state. We call F the bit mask of final states and the transition function is $D : \mathit{bitmasks} \times \Sigma \rightarrow \mathit{bitmasks}$.

The general mechanism of the search is as follows: we read the blocks b_r one by one. For each new block b read, representing a string B, and where we have already processed $T_{1 \ldots j}$, we update the state of the search so that after working on the block we have processed $T_{1 \ldots j+|B|} = T_{1 \ldots j} B$. To process each block, three steps are carried out: (1) its *description* is computed and stored, (2)