Inplace Run-Length 2d Compressed Search

Amihood Amir* Gad M. Landau† Dina Sokol‡

1 Introduction

Recent developments in multimedia have led to a vast increase in the amount of stored data. This increase has made it critically important to store and transmit files in a compressed form. The need to quickly access this data has given rise to a new paradigm in searching, that of *compressed matching* [1, 5, 6]. In traditional pattern matching, the pattern (P) and text (T) are explicitly given, and all occurrences of P in T are sought. In compressed pattern matching the goal is the same, however, the pattern and text are given in compressed form. Let c be a compression algorithm, and let $c(D)$ be the result of c compressing data D. A compressed matching algorithm is *optimal* if its time complexity is $O(c(T) + c(P))$.

Although optimality in terms of time is always important, when dealing with compression, the criterion of **extra space** is equally important. In many applications there is a limited amount of extra available space. In such cases, an algorithm that is optimal in terms of time, but uses $O(c(T))$ space (in addition to the compressed file itself) may not be feasible. In the field of data compression, a compression technique that compresses the text to one third of its original size is considered very good. With this fact in mind, consider an algorithm that runs in $O(c(T) + c(P))$ time and uses $O(c(T) + c(P))$ space. If the constants hidden in the *"big Oh"* exceed 3, then the algorithm performs no better than the naive algorithm of decompressing the text. Consequently, the constant factors of the complexity of a compressed matching play an important role. However, constant factors become asymptotically insignificant again if the complexity is proportional to the compressed pattern alone. This may be impossible to achieve for the time complexity, but seems feasible for the measure of extra space. Thus, we say that a compressed matching is *inplace* if it uses $O(c(P))$ extra space.

In this paper we present an algorithm to solve the two dimensional compressed matching problem for the run-length compression (used by fax transmission). The algorithm is, to our knowledge, the first known *inplace* 2d compressed search algorithm. In the following section we give a formal definition of the problem. In Section 3 we present a high level description of the algorithm.

2 Problem description

Let $S = s_1 s_2 \dots s_n$ be a string over some alphabet Σ. The *run-length compression* of string S is the string $S' = \sigma_1^{r_1}\sigma_2^{r_2}\dots\sigma_k^{r_k}$ such that: (1) $\sigma_i \neq \sigma_{i+1}$ for $1 \leq i < k$; and (2) S can be described as the concatenation of k *segments*, the symbol σ_1 repeated r_1 times, the symbol σ_2 repeated r_2 times, ..., and the symbol σ_k repeated r_k times. The *two-dimensional run length compression* is the concatenation of the run-length compression of all the matrix rows (or columns).

We define the *Two-Dimensional Run-Length Compressed Matching Problem* as follows:

Input: Text array T and pattern array P of size $m \times m$ in Two-dimensional run-length compressed form.

Output: All locations in T of sharp occurrences of P. Formally, the output is the set of locations (i, j) such that $T[i+k, j+l] = P[k+1, l+1]$ $k, l = 0 \dots m-1$, and $T[i+k, j] \neq T[i+k, j-1]$, $T[i+k, j+m-1] \neq T[i+k, j+m]$ $k = 0 \dots m-1$.

Amir and Benson [1] presented an algorithm that solves this problem[1] in time $O(|c(T)| \log |c(T)|)$. The authors then improved this algorithm in [3], achieving an algorithm that has *both* time and space $O(|c(T)|)$. Using known techniques, it is possible to modify the Amir-Benson-Farach algorithm [3] so that its extra space is $O(|P|)$. However, the constants involved in the complexity are relatively large.

Our search for a different approach is motivated by space considerations, as explained in the previous section. We present the first inplace algorithm for compressed two-dimensional pattern matching. In addi-

*Department of Mathematics and Computer Science, Bar-Ilan University, 52900 Ramat-Gan, Israel, (972-3)531-8770, and College of Computing, Georgia Tech; amir@cs.biu.ac.il;
Partially supported by NSF grant CCR-96-10170 and BSF grant 96-00509.

†Department of Computer Science, Haifa University, Haifa 31905, Israel (972-4)824-0103, and Department of Computer and Information Science, Polytechnic University; landau@poly.edu;
Partially supported by NSF grant CCR-9610238 and by the Israel Science Foundation founded by the Israel Academy of Sciences and Humanities.

‡Department of Mathematics and Computer Science, Bar-Ilan University, 52900 Ramat-Gan, Israel, (972-3)531-8407; sokold@cs.biu.ac.il

[1] The problem solved in [1] has a slightly different definition of matching. There the boundaries of the pattern may blend with the surrounding text.

tion, our extra space is almost $|c(P)|$ (i.e. a constant of 1).

3 Algorithm Idea

Our idea is to apply the two-dimensional pattern matching algorithm of [2] to the compressed pattern and text. The concept of that algorithm has proven useful in solving several problems (e.g. [4]). The idea of the algorithm is briefly presented below:

A *candidate* is a location in the text where the pattern may occur. We say that two candidates are *consistent* if they expect the same text characters in their region of overlap (*i.e.* the value of the witness is a $*$). The goal of the candidate consistency step is to eliminate candidates by performing duels, until all remaining candidates are mutually consistent.

In the verification phase the text elements are compared to the pattern elements to discover actual occurrences of the pattern. Although a given text location may be contained in several candidates, since all of the candidates are mutually consistent, each text element must be compared to a single pattern element.

Unfortunately, the algorithm cannot be applied "off-the-shelf" to the compressed matching problem since both the dueling stage and the verification stage introduce new problems due to the fact that the text and pattern are given in a compressed form.

In the ensuing subsections we will present the problems and outline the solutions at every stage of the algorithm.

3.1 Pattern Preprocessing:

The pattern preprocessing stage consists of constructing a witness table, $Witness[1 \ldots m, 1 \ldots m]$, for the pattern. Note that it is only possible for the two overlapping occurrences of the pattern to agree on the overlap, if the second copy begins at a position in $c(P)$. Therefore, the witness table need be constructed only for those positions in $c(P)$, resulting in a table with size $O(|c(P)|)$.

3.2 Text Scanning:

The text scanning stage has two steps: (1) candidate consistency (dueling) (2) candidate verification.

3.2.1 Candidate Consistency

Problem: It is necessary to make sure that we do not have too many candidates (initially, every compressed text location is a potential candidate). It is important that we never keep a list of more than size $O(|c(P)|)$.

Lemma 1 *During the dueling stage, our algorithm never keeps more than $|c(P)| + m$ candidates.*

Proof: Note that in the final stage, the number of pairwise consistent candidates in a text block of size $2m^2$, cannot be more than $O(|c(P)|)$. Our algorithm is constructed in a manner that the potential candidates at every stage will not exceed $|c(P)| + m$. $\Box$

Problem: The dueling stage of [2] deals with text columns. In the compressed text, the columns we are interested in do not necessarily coincide with the physical columns.

Lemma 2 *Desired pointers to the logical columns (the columns in the uncompressed text) can be maintained using $2m$ extra space.*

Proof: We maintain $2m$ pointers to the text, one pointer per row. When performing a duel between two candidates, the pointer of the row which contains the witness is moved to the appropriate column. Because the duels are performed strictly from right to left, these pointers are never moved to the right. Therefore, overall, the pointers are not moved more than $O(|c(T)| + thenumberofduels)$. $\Box$

Problem: In the dueling stage, the witness location is computed by a preprocessing on the pattern. However, the text locations may be different from the pattern locations, due to different run-lengths. Thus, a witness in the text may eliminate the wrong candidate because the run-lengths of one candidate were wrong.

Lemma 3 *Every candidate eliminated in a duel cannot be a pattern start. This is achieved with no more than $2m$ extra space.*

Proof: Maintain pointers to the mth logical column. The mth column gives a reference to every witness of every possible candidate. The total run-lengths can be computed without the need to write the run-length at every compressed text location. $\Box$

3.2.2 Candidate Verification:

The verification phase consists of two *waves*, a vertical wave followed by a horizontal wave. The vertical wave verifies the first column of each candidate occurrence.

Problem: In compressed matching, the vertical wave poses a problem since we cannot simply move down a given column in the compressed text.

Solution: The $2m$ pointers that were used in the dueling phase can be used in a similar way in the verification phase. Once the vertical wave is complete, the horizontal wave simply follows the rows of $c(P)$.

References

[1] A. Amir and G. Benson. Efficient two dimensional compressed matching. *Proc. of Data Compression Conference, Snow Bird, Utah*, pages 279–288, Mar 1992.

[2] A. Amir, G. Benson, and M. Farach. An alphabet independent approach to two dimensional pattern matching. *SIAM J. Comp.*, 23(2):313–323, 1994.

[3] A. Amir, G. Benson, and M. Farach. Optimal two-dimensional compressed matching. *Journal of Algorithms*, 24(2):354–379, August 1997.

[4] A. Amir and G. Calinescu. Alphabet independent and dictionary scaled matching. *Proc. 7th Annual Symposium on Combinatorial Pattern Matching (CPM 96)*, pages 320–334, 1996.

[5] M. Farach and M. Thorup. String matching in lempel-ziv compressed strings. *Proc. 27th Annual ACM Symposium on the Theory of Computing*, pages 703–712, 1995.

[6] L. Gasieniec, M. Karpinski, W. Plandowski, and W. Rytter. Randomized efficient algorithms for compressed strings: The finger-print approach. In *Proc. 7th Annual Symposium on Combinatorial Pattern Matching (CPM 96)*, pages 39–49. LNCS 1075, Springer, 1996.