Theoretical Computer Science 562 (2015) 33–45

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

A PTAS for the Square Tiling Problem

Amihood Amir [a,b,1,2], Alberto Apostolico [c,d,3], Gad M. Landau [e,f,4], Ely Porat [a,5], Oren Sar Shalom [a,1]

[a] Department of Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel
[b] Department of Computer Science, Johns Hopkins University, Baltimore, MD 21218, USA
[c] College of Computing, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30318, USA
[d] University of Padova, Italy
[e] Department of Computer Science, University of Haifa, Mount Carmel, Haifa 31905, Israel
[f] Department of Computer Science and Engineering, NYU Polytechnic School of Engineering, New York University, NY, USA

ARTICLE INFO

Article history:
Received 9 March 2014
Received in revised form 21 August 2014
Accepted 10 September 2014
Available online 5 October 2014
Communicated by G. Ausiello

Keywords:
Two dimensional tiling
Finite alphabets
NP-Hardness

ABSTRACT

The Square Tiling Problem was recently introduced as equivalent to the problem of reconstructing an image from patches and a possible general-purpose indexing tool. Unfortunately, the Square Tiling Problem was shown to be $\mathcal{NP}$-hard. A 1/2-approximation is known.

We show that if the tile alphabet is fixed and finite, there is a Polynomial Time Approximation Scheme (PTAS) for the Square Tiling Problem with approximation ratio of $(1-\frac{\epsilon}{2\log n})$ for any given $\epsilon \leq 1$.

Another topic handled in this paper is the $\mathcal{NP}$-hardness of the Tiling problem with an infinite alphabet. We show that when the alphabet is not bounded, even the decision version for rectangles of size $3n$ is $\mathcal{NP}$-Complete.

1. Motivation

At the end of the movie "*Raiders of the Lost Ark*", the ark is deemed too dangerous and it is decided to hide it in a manner that it shall never be found. Consequently, in a memorable scene, the ark is deposited in a huge government warehouse full of crates. Now it is indeed lost forever...

The proliferation of digital data is staggering. Even with the speed of current computers, sequential search is impossible in many applications. If efficient indexing techniques are not available, the data is, for all intents and purposes, as lost as the ark. In other words, the current amount of data bring us to a state that a search in a data base is similar to a search for a needle in a haystack, and indexing techniques will be the main tool for handling this search.

E-mail addresses: amir@cs.biu.ac.il (A. Amir), axa@cc.gatech.edu (A. Apostolico), landau@univ.haifa.ac.il (G.M. Landau), porately@cs.biu.ac.il (E. Porat), oren.sarshalom@gmail.com (O. Sar Shalom).
[1] Tel.: +972 3 531 8770.
[2] Partly supported by NSF grant CCR-09-04581, ISF grant 347/09, and BSF grant 2008217.
[3] Partly supported by BSF grant 2008217.
[4] Partly supported by NSF grant CCR-09-04246, ISF grant 347/09, DFG and BSF grant 2008217.
[5] Tel.: +972 3 738 4572.

http://dx.doi.org/10.1016/j.tcs.2014.09.012
0304-3975/

Mechanical data handling is older than Computer Science. As far back as in 1890, Hollerith tabulating machines were used to analyze the US census. However, the Hollerith machine was a technologically faster sequential search, it cannot be considered an indexing method.

Dictionaries and concordances were in use by scholars for generations. By the late 1940s the field of *Information Retrieval* was created. For many years that information was mostly textual, and a large body of scientific work has been established in the field [19,10,16]. With the advent of Computational Biology, digital libraries, and the Web, indexing of non-textual data is becoming increasingly more crucial. Some examples are provided below.

Computational biology

The three dimensional structure of the protein plays an important role in its functionality and as F. Cohen [5] writes "...similar sequences yield similar structures, but quite distinct sequences can produce remarkably similar structures" [11]. Protein similarity had been measured in the past by comparing the amino acid sequences of the proteins. It has been determined, though, that even proteins that are quite different in their sequence similarities can perform surprisingly similar functions [6], and homologous structures do not necessarily imply homologous sequences. Searching the growing database of protein structures for structure similarity is, therefore, an important task. In essence, a good indexing method is necessary. Unfortunately, the rate of growth of the protein database exceeds the retrieval capacity of current indexing methods. The processes that most state-of-the-art methods use to-date extract various local features, such as curvature or torsion angles, and index by these features. This is both time-consuming and limited by the selected features. Current methods cannot efficiently index protein structure for more than a few thousands proteins. Consequently, the methods aggregate proteins with some rough similarity, find a group that is "roughly similar" to the given protein and then check within this group. The disadvantage of this method is that it blurs local features that may actually be important, and thus may point the search algorithm to the wrong group. Recently, a new approach (for example [13]) was taken, where the protein structure is represented as letter strings using structural alphabets.

Linguistics

Lexical categorization is an important task of Natural Language Processing. The idea is to correctly tag parts of speech (e.g. as verbs, nouns). Sets of constraints have been suggested as possible aids in the task of lexical categorization [12]. Parikh mappings have been used for identifying such sets [1]. (A *Parikh mapping* is a function counting for each letter of an alphabet the occurrences of this letter in a word w [17].) In seeking the Parikh mapping, one is interested in the content of a substring, but in a scrambled order. In [1] some interesting techniques were developed, and it was apparent that problems that have known efficient solutions in the traditional pattern matching context, are still open for exploration. In particular, can a text be efficiently indexed for Parikh mapping?

Computer vision

Indexing images is one of the important challenges of web retrieval. Currently, image searches in all search engines are actually textual searches. The image captions are indexed and not the images themselves. Not only it is impossible to scan a picture and ask to find all "similar" pictures, but even such a mundane task as, given a picture, finding the image from which the given picture has been cropped, is not efficiently doable. State-of-the-art methods of indexing images index features that group a set of images into a similar prototype, e.g. having the same color histogram. Nevertheless, as in the case of Biology, such methods are not capable of finding images that are similar to the input image by some other feature.

Audio indexing

Indexing large audio archives has emerged recently [14] as an important research topic as large audio archives now exist. There are several possible goals to audio indexing — speaker indexing and speech indexing. Both are important for a variety of commercial and security applications. As in the applications above, the methods of use are preprocessing the corpus by selecting a set of features that roughly describe similar inputs, and then hierarchically seeking the closest match to an input segment. The method suffers from the same weakness as the computer vision and biological structure indexing.

The above applications all point to the need of breaking up data to small particles and using them as identifiers of the data.

We will describe below in a general "hand-wavy" manner the intuition of an idea that has been applied in the various domains we had discussed. We will then give the combinatorial abstraction of the idea and outline our results.

The main idea we would like to explore is the following. Since we do not want to commit ourselves a-priori to any particular shape, model, or relation, we slice the object into a very large amount of small pieces. Intuitively, when we get the jumbled pieces of two objects, we consider only those pieces that occur in both piles. If there are many of these, we have a potential similarity. However, even a large number of pieces in the intersection may not mean that their sources are similar. As an extreme example consider two black and white matrices A and B. A is a checkerboard and B has a black top and a white bottom. If we slice the matrices into squares the size of a square in the checkerboard we will end up with exactly the same small pieces in both jumbled matrices, but they are clearly very different.

Thus, we go a step further. Every model has a set of rules whereby two pieces can be judged as adjacent. If we can piece together large sub-objects from the similar pieces of both objects, we expect the given objects to be, indeed, similar. For example, an image of a car in the desert and a car in the forest would have the car in common.

The above idea has been used successfully in computer vision. Ullman and his groups used such "image fragments" for object classification (e.g. [18,7]), since such fragmentation gives implicit spatial information. The computer vision world has indeed embraced the *patches* model. The idea has also been employed in the graphics community (see e.g. [20]).

As previously mentioned, the more primitive idea of comparing just the number of "small pieces" is quite old. Color histogram has been used for decades as a crude index for content-based image retrieval systems [22]. It has been used in Natural Language Processing as well (e.g. [12]).

The human genome projects [9,8] and other genomes discoveries are based on a similar idea. Namely, the input are short sequences taken from copies of the DNA of the same cell, and the goal is to assemble one copy of this DNA sequence.

We describe in greater detail the application where the phenomenon of unordered parts of the pattern occurs — that of *Patch-Based Computer Vision*.

Patch-based computer vision

In computer vision the smallest primitive employed for describing an image is the pixel. However, analyzing an image as an ensemble of patches (i.e. spatially adjacent pixels/descriptors which are treated collectively as a single primitive), rather than individual pixels/descriptors, has some inherent advantages (i.e. computation, generalization, context, etc.) for numerous computer vision applications (e.g. matching, correspondence, tracking, rendering, etc.). Common descriptors in literature, other than pixels, have been contours, shape, flow, etc.

Recently, many inroads have been made into novel areas of computer vision through the employment of patch-based representations with machine learning and pattern recognition techniques.

Patch rendering: algorithms have been developed for rendering continuous images of objects modeled with bi-variate parametric surface patches. Catmull's [4] algorithm works directly from the mathematical definition of the surface patches. The algorithm functions by recursively subdividing each patch into smaller patches until the image of each fragment covers only one picture element.

In [3] there are extensions of this algorithm in the areas of texture simulation and lighting models. The parametrization of a patch defines a coordinate system which is used as a key for mapping patterns onto the surface. The intensity of the pattern at each picture element is computed as a weighted average of regions of the pattern definition function. The shape and size of this weighting function are chosen using digital signal processing theory. The patch rendering algorithm allows accurate computation of the surface normal to the patch at each picture element, permitting the simulation of mirror reflections. The amount of light coming from a given direction is modeled in a similar manner to the texture mapping and then added to the intensity obtained from the texture mapping.

We are concerned with the question of low-level pattern matching in a patch-based system. Our patching will be the simplest possible. Recently [2] Amir and Parienty introduced the *Patches Model* as a possible abstraction of ad-hoc techniques that have been used to index various domains. The text is cut into equal small overlapping squares. Each square overlaps with half the area of each of the squares to its left, right, below and above it, and a quarter of the area of each of the squares to its left bottom, right bottom, right top and left top. Patterns whose patches match a large number of patches in an indexed object, are likely to appear in the image.

The first task tackled in [2] was reconstructing an image from its patches — the *Square Tiling Problem*. Unfortunately, it was proven that this problem is $\mathcal{NP}$-hard. Amir and Parienty showed a polynomial 1/2 approximation algorithm for reconstructing a $2n \times 2n$ image from its n^2 2×2 patches. An image constructed from $n \times n$ tiles has $2n^2 - 2n$ "*seams*" between tiles. If it is correctly constructed then two adjacent tiles *match* at the seams, i.e., have equal alphabet symbols on their adjacent edges. We then say that the seam is *correct*, otherwise there is an *error* at the seam. We count a single error at the seam whether one or two symbols are not equal at the adjacent tiles. An *approximation* in that context is an $n \times n$ square where "many" seams match. In [2] a polynomial-time algorithm that constructs a square with at least $\frac{1}{2}$ of the seams being correct, was shown. If the idea of patch indexing is to have any chance of applicability, much better approximations are necessary.

In this paper we present a Polynomial Time Approximation Scheme (PTAS) for the Square Tiling Problem with a fixed finite alphabet. This assumption is reasonable since most applications have a fixed bounded alphabet (in DNA and RNA the alphabet is of size 4, in proteins, the alphabet size is 23, in image processing one may talk about 256 grey levels, etc.). We show that for such finite alphabets, for any given ϵ and n^2 tiles, there is an algorithm polynomial in n and ϵ that constructs a square with at least $(1 - \frac{\epsilon}{2 \log n})n^2$ correct seams. In fact, we show a stronger results. In [2], only sets of patches that can be square tiled with *no errors* were considered. We generalize the problem to assume that we are given n^2 tiles, which may not necessarily match without seam errors. However, we would like to find the square tiling that produces the smallest possible number of errors. Let s_{max} be the maximum number of correct seams in a square tiling. Our algorithm constructs a square tiling with at least $(1 - \frac{\epsilon}{2 \log n})s_{max}$ correct seams.

The paper is constructed as follows. We begin with background and definitions in Section 2. In Section 3 we show how to reconstruct a rectangle of size $n \times \log n$ tiles in polynomial time (or, for a given c an $n \times c$ rectangle). We then show in Section 4 how multidimensional knapsack techniques can be used to approximate a square tiling.

2. Background and definitions

The definition below was used in [2] to combinatorially describe the patches concept.

Definition 1. Given matrix M,

$$\begin{pmatrix} m_{0,0} \cdots\cdots\cdots m_{0,n} \\ \cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots \\ m_{n,0} \cdots\cdots\cdots m_{n,n} \end{pmatrix}$$

A is a *division of M to patches* if $A = \{a_{0,0}, \cdots a_{0,n-1}, \cdots\cdots a_{n-1,0}, \cdots a_{n-1,n-1}\}$ and $\forall i, j$, $a_{i,j} = \left[\begin{smallmatrix} m_{i,j}, m_{i,j+1} \\ m_{i+1,j}, m_{i+1,j+1} \end{smallmatrix}\right]$.

The problem we are concerned with is the converse.

Definition 2. The problem of *constructing an image from patches* is defined as follows:
INPUT: $A = \{a_0, \cdots, a_{n^2-1}\}$ be a set of 2×2 matrices over alphabet Σ.
OUTPUT: Construct an $(n+1) \times (n+1)$ matrix

$$M = \begin{pmatrix} m_{0,0} \cdots\cdots\cdots m_{0,n} \\ \cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots \\ m_{n,0} \cdots\cdots\cdots m_{n,n} \end{pmatrix}$$

such that A is the division of M to patches, if such a matrix exists. Otherwise report that no matrix can be constructed from the input.

In [2] it was proven that the problem of constructing an image from overlapping patches is equivalent to the square tiling problem, where we are given 2×2 patches over alphabet Σ and we are only allowed to place patches next to each other if their symbols match. Formally:

Definition 3. A patch A may be *correctly placed* to the right (left, top, bottom) of patch B if the pair of letters on the right (left, top, bottom) side of B are the same as the pair on the left (right, bottom, top) of patch A. The common edge of two patches is called a *seam*. The seam is *correct* if the adjacent tiles are correctly placed. Otherwise, it is an *error*.

Definition 4. The *Square Tiling Problem* is defined as follows:
INPUT: A multiset S of n^2 tiles. Each tile is a 2×2 matrix over alphabet Σ
DECIDE: Whether there exists a square of tiles correctly placed next to each other, whose tiles are exactly those in the multiset.

In [2] it was proven that the Square Tiling Problem is $\mathcal{NP}$-hard.

We seek a polynomial time algorithm that approximates the square. The approximation means reconstructing an image from all of the tiles, with the minimum amount of seam errors. The square tiling of n^2 tiles has $2n^2 - 2n$ seams. In [2], a square that is guaranteed to have n^2 correct seams was constructed in polynomial time. Thus, that algorithm approximates the matches within a factor of $\frac{1}{2}$.

Our goal is to improve this result. We show an algorithm with approximation ratio of $1 - \frac{\epsilon}{2 \log n}$, for any constant $\epsilon \leq 1$, if the given alphabet Σ is fixed and finite. In fact, we even generalize that. Assume that the maximum number of correct seams that can be achieved by *any* tiling of a given set of patches S is s_{max}. Then for any given ϵ we can tile, in time polynomial in the size of S and ϵ, the patches of S achieving at least $(1 - \frac{\epsilon}{2 \log n}) s_{max}$ correct seams.

3. Rectangle tiling over fixed and finite alphabet

We begin by showing that the $\mathcal{NP}$-hardness is dependent on the dimensions of the square. For some rectangle dimensions, the Tiling Problem is polynomial-time computable.

Definition 5. The *entropy* of multiset m of size $a \cdot b$ is the minimum number of errors obtained by tiling m as an $a \times b$ rectangle.

Theorem 1. *The entropy of all multisets of size* $n \times \frac{\log n}{\epsilon}$ *can be computed in time polynomial in* ϵ *and* n.

Proof. We want to compute the entropy for **all** multisets, so we must first make sure that there are not too many multisets.

Claim. *There is a polynomial number of multisets of size* $n \times \frac{\log n}{\epsilon}$.

Proof. The alphabet is fixed and finite. Let us assume that $\Sigma = \{1, ..., c\}$. This means there are $O(c^4)$ types of tiles.

A multiset is a histogram of the types of tiles that compose it.

There are at most $n \cdot \frac{\log n}{\epsilon}$ tiles of each type, while the sum of all types is also $n \cdot \frac{\log n}{\epsilon}$. Combinatorially, the number of different multisets it is equal to the number of combinations to insert n' balls into k' bins, when $n' = n\frac{\log n}{\epsilon}$ and $k' = c^4$. There are $\binom{n'+k'-1}{n'} = O(n'^{(k'-1)}) = O((n \cdot \frac{\log n}{\epsilon})^{c^4-1})$ multisets. □

We now show an exact polynomial-time algorithm for a fixed finite alphabet that calculates the entropy for each multiset m of size $n \cdot \frac{\log n}{\epsilon}$. We will need an auxiliary data structure to keep track of some values for the multisets constructed during the execution of the algorithm. In particular, consider a possible tiling of multiset of size $i \cdot \frac{\log n}{\epsilon}$ into a rectangle of i columns and $\frac{\log n}{\epsilon}$ rows. The rightmost column, rc, of such a tiling is composed of $\frac{\log n}{\epsilon}$ 2×2 patches. For the sake of improving the time complexity, we consider the column composed only of the $\frac{2 \log n}{\epsilon}$ symbols on the *right side* of the patches in the rightmost column. It is possible that different columns of $\frac{\log n}{\epsilon}$ patches have the same right side. For any multiset m and right side r, choose a tiling with the lowest entropy. For that tiling, record in addition to the right side r, also the $\frac{2 \log n}{\epsilon}$ symbols on the *left side* of the patches in the rightmost column.

Example. For

$$m = \left\{ \begin{bmatrix} a,a \\ b,b \end{bmatrix}, \begin{bmatrix} c,a \\ d,b \end{bmatrix}, \begin{bmatrix} b,b \\ c,c \end{bmatrix}, \begin{bmatrix} d,b \\ d,c \end{bmatrix} \right\}$$

There are eight possible tilings with right side $\begin{bmatrix} a \\ b \\ b \\ c \end{bmatrix}$.

For example:

$\begin{array}{|c|c|}\hline a,a & c,a \\ b,b & d,b \\ \hline b,b & d,b \\ c,c & d,c \\ \hline \end{array}$, which has entropy 2,

$\begin{array}{|c|c|}\hline c,a & a,a \\ d,b & b,b \\ \hline b,b & d,b \\ c,c & d,c \\ \hline \end{array}$, which has entropy 3,

$\begin{array}{|c|c|}\hline a,a & c,a \\ b,b & d,b \\ \hline d,b & b,b \\ d,c & c,c \\ \hline \end{array}$, which has entropy 3, and $\begin{array}{|c|c|}\hline c,a & a,a \\ d,b & b,b \\ \hline d,b & b,b \\ d,c & c,c \\ \hline \end{array}$, which has entropy 0.

Thus, the right side is $\begin{bmatrix} a \\ b \\ b \\ c \end{bmatrix}$, and the left side is $\begin{bmatrix} a \\ b \\ b \\ c \end{bmatrix}$.

Auxiliary data

Define a 2-dimensional array M with the following values:

At iteration i, for each multiset m of size $i \cdot \frac{\log n}{\epsilon}$, that represents a rectangle of i columns and $\frac{\log n}{\epsilon}$ rows, and for each r — the right side of the rightmost column of multiset m:

- $M[m, r].entropy$ holds the minimum entropy of m subject to the constraint that r is the right side column of the $\frac{\log n}{\epsilon} \times i$ rectangle tiling of m.
- $M[m, r].left$ is the left side of the rightmost column in a tiling that obtained the minimum entropy. This field's goal is to enable us to reconstruct the optimal tiling of m.

The size of array M is bounded by the product of the number of multisets and the number of possibilities for the right side of column r. The number of multisets for the last iteration was calculated above as $O((n\frac{\log n}{\epsilon})^{c^4-1})$.

Calculation of the number of different possibilities for the right side of a column:

There are c^2 possibilities for each tile, and $\frac{\log n}{\epsilon}$ tiles altogether. Therefore, the right side of a column can be arranged in $(c^2)^{\frac{\log n}{\epsilon}} = (2^{\log c^2})^{\frac{\log n}{\epsilon}} = (2^{\log n})^{\frac{\log c^2}{\epsilon}} = n^{\frac{\log c^2}{\epsilon}} = n^{\frac{2\log c}{\epsilon}} = z$ different ways.

Thus the size is clearly polynomial.

Algorithm outline

Initialization phase: initialize all of the *entropy* values of M to ∞.

A column can be considered a Cartesian product of two sides $\langle l, r\rangle$, so there are z^2 different columns.

Count the number of errors of every possible column c, i.e., an ordered $\frac{\log n}{\epsilon}$ tiles placed one above the other. In this case, m is the multiset comprised by c and r is the right side of column c. Obviously, if there are some columns $c_1, ..., c_k$ with number of errors $e_1, ..., e_k$ respectively, such that their collections of tiles are the same, denote by m, and their right side is the same, denote by r, then the combination m, r will get the value $\min e_i$

Assume we have computed array M for the first i iterations.

Iteration step: For each combination of (m, r) computed in the previous iteration, the algorithm attaches all possibilities as column c to r. Let C be the multiset of the elements of c. Then each such column c creates a new multiset $m \leftarrow m \cup C$, and new right side $r \leftarrow$ right side of column c. The new entropy e' is the entropy of the old entry + the number of errors introduced by attaching c to the right of the old entry.

Thus, $M[m, r] \leftarrow \min(M[m, r], e')$

Observation: let $[m', r']$ and $[m', r'']$ be two entries of M, with respective entropies e', e''. Assume that attaching c to the right of $[m', r']$ introduces err' errors, and attaching c to the right of r'' introduces err'' errors. Then the value of $M[m' \cup C, c]$ is taken as $\min(e' + err', e'' + err'')$. Clearly, if there are more than two possibilities of achieving the same entry, the minimum number of errors is chosen as the entropy.

The algorithm

Define a function $f : \{1, \ldots, z\} \times \{1, \ldots, z\} \rightarrow \mathbb{N}^{c^4}$ such that:
$\forall$ column $\langle l, r\rangle$, $f(\langle l, r\rangle)$ = # of occurrences of each type of tile in $\langle l, r\rangle$.
We also define a function g: Let r be the right side of a column and $\langle l', r'\rangle$ be a column.
$g(r, \langle l', r'\rangle)$ is the number of errors introduced by attaching $\langle l', r'\rangle$ to the right of r.
This value is the sum of the errors at the seams between r and $\langle l', r'\rangle$ and the errors within $\langle l', r'\rangle$.

1 For each combination of right side of column r and column $\langle l', r'\rangle$
1.1 Set the value of $g(r, \langle l', r'\rangle)$
2 For each cell, initialize $M[m, r].entropy$ to ∞.
3 For each column $\langle l, r\rangle$, $1 \le r, l \le m$ do
3.1 Let $w = \langle w_1, ..., w_{c^4}\rangle$ be the number of occurrences of each type of tile in $\langle l, r\rangle$
3.2 Set $f(\langle l, r\rangle) = w$
3.2 $err \leftarrow$ number of errors in $\langle l, r\rangle$
3.3 If $M[w, r].entropy > err$ then
3.3.1 $M[w, r].entropy \leftarrow err$
3.3.2 $M[w, r].left \leftarrow l$

Column Construction Phase:

1 For $i = 2, ..., n$ do
1.1 For each combination in M of multiset of size $(i-1)\frac{\log n}{\epsilon}$ and right side of column (w, r)
1.1.1 $err \leftarrow M[w, r].entropy$
1.1.2 For each column $\langle l', r'\rangle$, $1 \le r',\ l' \le m$ do
1.1.2.1 $err' \leftarrow err + g(r, \langle l', r'\rangle)$
1.1.2.2 $w' \leftarrow f(\langle l', r'\rangle) \cup w$
1.1.2.3 If $M[w', r'].entropy > err'$ then
1.1.2.3.1 $M[w', r'].entropy \leftarrow err'$
1.1.2.3.2 $M[w', r'].left \leftarrow l'$
1.1.2.3.3 $M[w', r'].prev \leftarrow M[w, r]$
2 For each multiset m of size $n\frac{\log n}{\epsilon}$, the entropy of m is $\min_r M[m, r]$.

Correctness: We prove by induction that for each combination of multiset m of size $i\frac{\log n}{\epsilon}$ and right side of column r, the algorithm finds the entropy of (m, r).

Base case: For $i = 1$, the assumption holds, because the algorithm goes through all columns and picks the minimum for each combination.

Inductive step: Assume correctness for i and prove for $i+1$: Let m_{i+1} be a multiset of size $(i+1)\frac{\log n}{\epsilon}$ and r_{i+1} a right side of a column. Any optimal tiling of m_{i+1} as a rectangle such that r_{i+1} is its right side of the rightmost column, is a selection among an optimal tiling of multiset m_i as a rectangle of size $i \times \frac{\log n}{\epsilon}$ such that the right side of the rightmost column is r_i adjacent to column $\langle l_{i+1}, r_{i+1}\rangle$, such that the number of errors of m_i and $\langle l_{i+1}, r_{i+1}\rangle$ is smallest. By the induction assumption, the algorithm finds m_i as iteration i. Iteration $i+1$ attaches all possible columns, particularly $\langle l_{i+1}, r_{i+1}\rangle$.

Therefore the algorithm finds (m_{i+1}, r_{i+1}). □

Claim 1. *The algorithm's complexity is polynomial.*

Proof. *Initialization phase:* Considering all combinations of right side of column r and column $\langle l', r'\rangle$ is $O(z^3) = O(n^{\frac{6\log c}{\epsilon}})$.

Column construction: We seek after the number of multisets of size $i\frac{\log n}{\epsilon}$ $\forall i \le n$. There are $O((n \cdot \frac{\log n}{\epsilon})^{c^4-1})$ multisets of size exactly $n\frac{\log n}{\epsilon}$. Therefore, there are $O(n \cdot (n \cdot \frac{\log n}{\epsilon})^{c^4-1})$ multisets altogether. For each multiset we go through all m^2 columns and perform a constant time work.

Therefore, the algorithm's running time is $O(n \cdot (n \cdot \frac{\log n}{\epsilon})^{c^4-1} n^{\frac{4\log c}{\epsilon}})$.

Finding the optimal tiling: Once a multiset $m = \langle t_1, ..., t_{c^4}\rangle$ of size $n\frac{\log n}{\epsilon}$, is constructed, we can trace the backward pointers and tile it as a rectangle with a minimum number of errors:

1 Choose r such that $M[m, r].entropy$ is minimal
2 Do $n-1$ times:
2.1 Column $c \leftarrow \langle M[m, r].left, r\rangle$
2.2 Output c
2.3 $M[m, r] \leftarrow M[m, r].prev$. □

4. The approximation

In the previous section we enumerated all multisets and their entropies. That action could be referred as a pre-processing action, because it does not use the input of the problem, but only exploits the problem size n.

We now partition the n^2 input patches into $\frac{n\cdot\epsilon}{\log n}$ sets of $n \cdot \frac{\log n}{\epsilon}$ tiles, such that the sum of their entropies is minimal. This will also be done by partitioning *all* multisets of size n^2 into such sets.

Algorithm outline

Step 1 — Find the entropy for all multisets of size $n \cdot \frac{\log n}{\epsilon}$.
Step 2 — Select a set of multisets, consisting exactly of the input tiles, with minimal entropy.

Implementation

Step 1: The implementation of step 1 was shown in Section 3.
Step 2: This problem is similar to another $\mathcal{NP}$-hard problem — The *Multidimensional Knapsack problem*. It needs to select a multiset of given objects (or items) in such a way that the total profit of the selected objects is maximized while a set of knapsack constraints are satisfied.

The problem is formally stated as follows:

Definition 6. The *Multidimensional Knapsack Problem (MDK)* is defined as a solution to the following Integer Program:

$$\textbf{Maximize} \quad \sum_{j=1}^{m} c_j \cdot x_j$$

$$\textbf{Subject to:} \quad \sum_{j=1}^{m} a_{i,j} \cdot x_j \le b_i, \quad \forall i = 1..k$$

$$x_j \ge 0,\ 1 \le j \le m$$

where m is the number of objects, k is the number of dimensions that constrains the knapsack, c_j represents the benefit of the object j in the knapsack, x_j is a variable that indicates how many times the object j has been stored in the knapsack, b_i represents the i-th dimension's capacity of the knapsack, and $a_{i,j}$ represents the entries of the knapsack's constraints matrix.

In our case:

- m is the number of multisets of size $n\frac{\log n}{\epsilon}$
- k is the number of types of tiles, which is c^4
- c_j is the number of matches in multiset j, i.e., $2(n-1)\log n$ - the entropy of the multiset
- $a_{i,j}$ is the number of tiles of type i that are in multiset j
- b_i is the number of tiles of type i that are in the input

Unfortunately, MDK is $\mathcal{NP}$-hard. It was also shown that finding an FPTAS even for a special case where all profits are the same and equal to 1 and $m=2$ is $\mathcal{NP}$-hard [15].

On the other hand, there is a pseudo-polynomial time algorithm for the Knapsack problem. The complexity of the algorithm is in the **value** of the size of the knapsack, which is not polynomial in the **length** of the input to the problem. However, in our case this is not an obstacle, because the input is a multiset of size n^2, and the output is their tiling, i.e., the positioning of each tile. So we can assume the input is a collection of size n^2, and not just the histogram of the types of tiles. Therefore the input is polynomial in the value n^2. But the MDK problem is $\mathcal{NP}$-hard in the strong sense and thus any dynamic programming approach would result in strictly exponential time bounds [21].

We want an exact algorithm to be polynomial on the one hand, and on the other hand to support multiple knapsacks, utilizing the unique characteristics of our problem:

- All objects have the same sum of weights, $n\frac{\log n}{\epsilon}$, over the dimensions.
- The input size is $n^2 \Rightarrow$ Any multiset may appear up to $\frac{n\cdot\epsilon}{\log n}$ times in the solution.
- A solution is valid only if all knapsacks are completely filled.

Let $MS = \{c_1, \ldots, c_x\}$, be the set of all multisets of size $n\frac{\log n}{\epsilon}$, where x is of size $O((n\cdot\frac{\log n}{\epsilon})^{c^4-1})$. Multiset c_i can be represented by $\langle w_{i1}, \ldots, w_{ic^4}\rangle$, where w_{ij} is the number of patches of type j there are in multiset c_i. Let e_i be the entropy of multiset c_i. We assume that MS is sorted in non-decreasing order of its entropy, i.e., $\forall i \le x-1$, $e_i \le e_{i+1}$.

The input of the tiling problem is a set of patches S of size n^2. S can be represented by the tuple $\langle S_1, \ldots, S_{c^4}\rangle$, where S_i is the number of input tiles of type i. Our task is to find a multiset of multisets $C \subseteq MS$, such that $\bigcup_{c\in C} c = S$, i.e. $\forall j$, $1 \le j \le c^4$, $\sum o_i \cdot w_{ij} = S_j$, where o_i is the number of occurrences of c_i in C.

The dynamic programming matrix

$T[1..x;\ 1..(n^2)^{c^4}]$ is a matrix with the following values:
Let $\langle b_1, \ldots, b_{c^4}\rangle$ be a multiset of size between 0 and n^2, and let i be a number $1 \le i \le x$.

If there does not exist a multiset which is the union of sets from $\{c_1, \ldots, c_i\}$, and whose elements are exactly the patches $\langle b_1, \ldots, b_{c^4}\rangle$, then $T(i, \langle b_1, \ldots, b_{c^4}\rangle) = \infty$.
Otherwise, let L be such a multiset where $\sum_{c\in L}(\text{entropy of } c) = E$ is smallest. Set $T(i, \langle b_1, \ldots, b_{c^4}\rangle) = E$.

Algorithm outline

The matrix is filled using dynamic programming:

Initialization: The first row can use only c_1, so $\forall\alpha \in \mathbb{N}_0$ every cell that represents $\alpha \cdot c_1$ will have the entropy $\alpha \cdot e_1$. The rest of the cells are infeasible and their entropies are ∞.

Filling the matrix: For each column $(b_1, \ldots, b_{c^4})$ and row i there are two possibilities:

1. The optimal solution does not use c_i at all in order to achieve $(b_1, \ldots, b_{c^4})$. In that case $T(i, \langle b_1, \ldots, b_{c^4}\rangle) = T(i-1, \langle b_1, \ldots, b_{c^4}\rangle)$
2. The optimal solution uses c_i at least one time. In that case $T(i, \langle b_1, \ldots, b_{c^4}\rangle) = T(i, \langle b_1, \ldots, b_{c^4}\rangle \backslash c_i) + e_i$

Therefore:

1. $T(i, \langle b_1, \ldots, b_{c^4}\rangle) \leftarrow \min(T(i, \langle b_1, \ldots, b_{c^4}\rangle), T(i-1, \langle b_1, \ldots, b_{c^4}\rangle))$
2. $T(i, \langle b_1, \ldots, b_{c^4}\rangle \cup c_i) \leftarrow \min(T(i, \langle b_1, \ldots, b_{c^4}\rangle \cup c_i), T(i, \langle b_1, \ldots, b_{c^4}\rangle) + e_i)$

Before presenting the algorithm in detail, we need to define an order on the multisets.

Definition 7. Let S', S'' be two multisets of tiles. We say that $\mathbf{S'} \le \mathbf{S''}$ if $\forall i\ 1 \le i \le c^4\ S'_i \le S''_i$. (The meaning is that for every type of tile, S'' has more patches of that type than S'.)

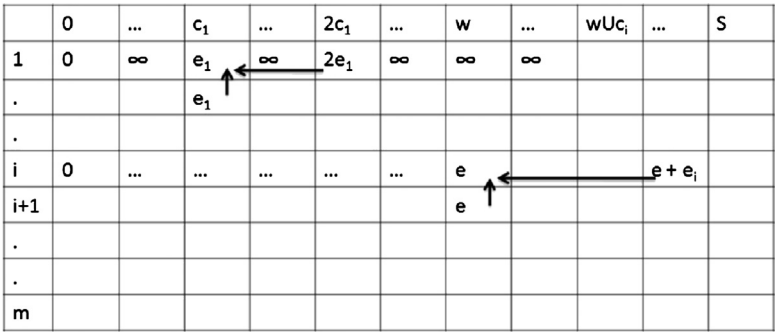

Fig. 1. MDK – dynamic programming.

Below is a pseudo-code for the dynamic programming algorithm.

Algorithm for constructing multisets entropy

Initialization:

1 Initialize all the elements of T to ∞
2 $T[1, \vec{0}_{c^4}].entropy \leftarrow 0$
3 $\alpha \leftarrow 1$
4 while $(\alpha \cdot c_1 \leq S)$
4.1 $T[1, \alpha \cdot c_1].entropy \leftarrow \alpha \cdot c_1$
4.2 $T[1, \alpha \cdot c_1].prev \leftarrow T[1, (\alpha - 1) \cdot c_1]$
4.3 $\alpha \leftarrow \alpha + 1$

Filling the Matrix:

5 for $i \leftarrow 2 \ldots x$ in $n\frac{\log n}{\epsilon}$ hops:
5.1 for $size \leftarrow 0 \ldots x^2$
5.1.1 $\forall \langle b_1, \ldots, b_{c^4} \rangle s.t. \sum b_i = size$ and $T[i, \langle b_1, \ldots, b_{c^4} \rangle].entropy < \infty$
5.1.1.1 if $i \neq x$ then
5.1.1.1.1 $T[i+1, \langle b_1, \ldots, b_{c^4} \rangle].entropy \leftarrow T[i, \langle b_1, \ldots, b_{c^4} \rangle].entropy$
5.1.1.1.2 $T[i+1, \langle b_1, \ldots, b_{c^4} \rangle].prev \leftarrow T[i, \langle b_1, \ldots, b_{c^4} \rangle]$
5.1.1.2 if $(\langle b_1, \ldots, b_{c^4} \rangle \cup c_i \leq S)$ then
5.1.1.2.1 if $(T[i, \langle b_1, \ldots, b_{c^4} \rangle \cup c_i].entropy > T[i, \langle b_1, \ldots, b_{c^4} \rangle].entropy + e_i)$ then
5.1.1.2.1.1 $T[i, \langle b_1, \ldots, b_{c^4} \rangle \cup c_i].entropy \leftarrow T[i, \langle b_1, \ldots, b_{c^4} \rangle].entropy + e_i$
5.1.1.2.1.2 $T[i, \langle b_1, \ldots, b_{c^4} \rangle \cup c_i].prev \leftarrow T[i, \langle b_1, \ldots, b_{c^4} \rangle]$
6 The multisets participating in the result can be reconstructed by following the pointers from $T[x, S]$

Example. A schematic of the dynamic programming matrix for the multidimensional knapsack problem can be seen in Fig. 1.

Theorem 2. *The dynamic programming algorithm's running time is polynomial.*

Proof. There are $m \cdot (n^2)^{c^4}$ cells in matrix M. The algorithm only handles columns representing multisets of size divisible by $n\frac{\log n}{\epsilon}$. The time to fill each cell is constant. Therefore, the total complexity is $O((n \cdot \frac{\log n}{\epsilon})^{c^4-1} \cdot (n^2)^{c^4}/n\frac{\log n}{\epsilon})$ □

Theorem 3. *Let S be a set and let s be the number of correct seams in the dynamic programming construction of S. Let s_{max} be the maximum number of correct seams for any square tiling of S. Then $s \geq (1 - \frac{\epsilon}{2 \log n}) s_{max}$.*

Proof. Let M be an optimal tiling. Let X be the number of correct vertical seams, and Y the number of correct horizontal seams in M. $s_{max} = X + Y$. (If M has no errors, then $s_{max} = 2n^2 - 2n$).

Without loss of generality we may assume that $X \geq Y$ (otherwise, we rotate all tiles by 90°). Since the dynamic programming algorithm provides the optimum tiling within the strips of size $\frac{\log n}{\epsilon} \times n$, and the only errors may occur when "putting together" these strips, then it is clear that the number of correct vertical seams decided by our algorithm is no less than the number of vertical seams in the optimum tiling. We need to consider only Y — the number of correct horizontal seams, since our algorithm makes no effort to match the rows between the strips.

We start by identifying the total number of correct horizontal seams at the bottom of the $\frac{\log n}{\epsilon} \times n$ strips. Call that number Y_0 (see Fig. 2). Next we consider the strips as moved down by an offset of 1, i.e., assume the first strip is of only

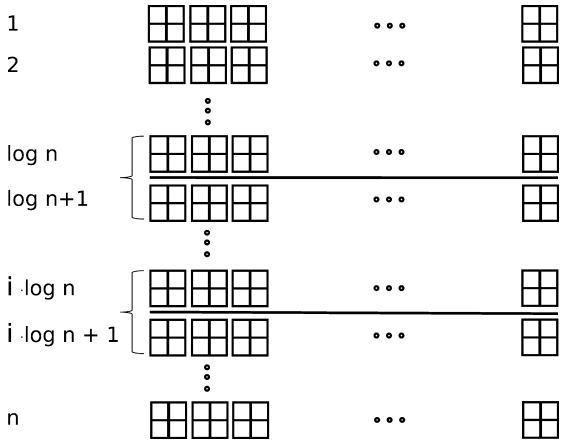

Fig. 2. Example – Y_0.

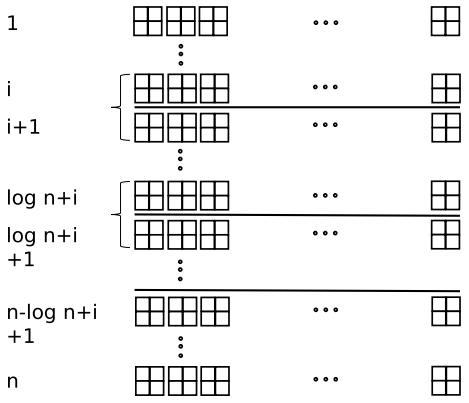

Fig. 3. Example – Y_i.

one row, and the following strips are of size $\frac{\log n}{\epsilon} \times n$. The total number of correct horizontal seams at the bottom of the $\frac{\log n}{\epsilon} \times n$ strips (with offset 1) we call Y_1. In general, let Y_i the total number of correct horizontal seams at the bottom of the $\frac{\log n}{\epsilon} \times n$ strips with offset i (i.e., the first strip has only i rows, followed by $\frac{\log n}{\epsilon} \times n$ strips.) Formally, $\forall 0 \leq i \leq \frac{\log n}{\epsilon} - 1$, define

$$Y_i = \sum_{j=1}^{\frac{n \cdot \epsilon}{\log n}} \left(\text{number of matches between row } j \cdot \frac{\log n}{\epsilon} + i \text{ and the row below it} \right).$$

For an example, see Fig. 3.

Let Y_{min} be such that $Y_{min} \leq Y_i \forall i$, $0 \leq i \leq \frac{\log n}{\epsilon} - 1$. We will assume that the worst happened, and all the horizontal seams at the bottom of the $\frac{\log n}{\epsilon} \times n$ strips of the dynamic programming tiling of S are erroneous. However, this number of errors cannot exceed Y_i, since the dynamic programming vertical tiling within the strips is *superior* to the optimal tiling. Therefore $s \geq X + Y - Y_{min}$.

However, because of averaging considerations, it is clear that $Y_{min} \leq Y / \frac{\log n}{\epsilon}$. Therefore

$$s \geq s_{max} - Y_{min} \geq s_{max} - \frac{Y}{\frac{\log n}{\epsilon}} \geq \left(1 - \frac{\epsilon}{\log n}\right) s_{max}. \qquad \square$$

In this section we used multisets of size $n * \log n/\epsilon$ as the building blocks. Using techniques taken from the Multi-dimensional Knapsack Problem and exploiting the uniqueness of our problem, we managed to construct a solution that approximates the objective within a factor of $(1 - \epsilon / \log n)$.

5. Rectangle tiling over infinite alphabet

As proven above, the Rectangle Tiling Problem over a finite alphabet is polynomial-time computable for input of size $n \log n$. We now show that over an infinite alphabet, even the decision version for rectangles of size $3n$ is $\mathcal{NP}$-Complete.

Definition 8. The *Rectangle Tiling Problem* is defined as follows:

INPUT: A multiset S of $n \cdot c$ tiles. Each tile is a 2×2 matrix over alphabet Σ

DECIDE: Whether there exists a rectangle of size $n \times c$ of tiles correctly placed next to each other, whose tiles are exactly those in the multiset.

Theorem 4. *Rectangle Tiling with dimensions $n \times 3$ over an infinite alphabet is $\mathcal{NP}$-Complete.*

Proof. The problem is in NP because given an arrangement of the tiles, it's easy to check whether all tiles are correctly placed.

Reduction from 3-dimensional matching

Definition 9. The *3-dimensional matching Problem* is defined as follows:

Let X, Y, and Z be finite, disjoint sets. Let T be a subset of $X \times Y \times Z$. That is, T consists of triples (x, y, z) such that $x \in X$, $y \in Y$, and $z \in Z$. $M \subseteq T$ is a 3-dimensional matching if the following holds: for any two distinct triples $(x1, y1, z1) \in M$ and $(x2, y2, z2) \in M$, we have $x1 \neq x2$, $y1 \neq y2$, and $z1 \neq z2$.

The decision problem is: given a set T and an integer k, decide whether there exists a 3-dimensional matching $M \in T$ with $|M| \geq k$.

We'll refer to a special case that $k = |X| = |Y| = |Z|$, which is also a known $\mathcal{NP}$-Complete problem. In this case, a 3-dimensional matching is an exact cover: the set M covers each element of X, Y, and Z exactly once.

Construction

Let $D \equiv (X, Y, Z, T)$ be the input for the 3D-Matching problem. We construct a set of tiles S of size $3n$, such that $D \in$ 3DM $\Leftrightarrow S \in$ Rectangle Tiling.

1. $\forall x_i \in X$ construct 2 tiles. Call them "type I" tiles:

*	x_i
*	A_1

x_i	*
A_1	*

   Respectively, do the same $\forall y_i \in Y$ and $\forall z_i \in Z$.
2. For each triple $(x, y, z) = t_i \in T$, construct 12 tiles. Call them "type II" tiles:

*	A_2
*	t_i^1

A_2	A_2
t_i^1	t_i^2

A_2	A_2
t_i^2	t_i^3

A_2	A_2
t_i^3	t_i^4

A_2	A_2
t_i^4	t_i^5

A_2	*
t_i^5	*

*	t_i^1
*	x

t_i^1	t_i^2
x	*

t_i^2	t_i^3
*	y

t_i^3	t_i^4
y	*

t_i^4	t_i^5
*	z

t_i^5	*
z	*

3. Let $n = |X| = |Y| = |Z|$ and let $m = |T|$. Construct $6(m - n)$ "type III" tiles.

   $4(m - n)$ tiles of the form:

A_3	A_3
A_2	A_2

   and $(m - n)$ of the form:

*	A_3
*	A_2

A_3	*
A_2	*

Correctness

$(X, Y, Z, T) \in$ 3DM $\Rightarrow S \in$ Rectangle Tiling

We want to show an arrangement of the tiles without any errors.

We know there are exactly n triples in the matching. W.L.O.G we'll assume those triples are $t_1, \ldots t_n$.

Step 1 – For each triple t_i, $i = 1 \ldots n$ we arrange the tiles in the following way:

*	A_2	A_2	A_2	A_2	A_2	A_2	A_2	A_2	A_2	A_2	A_2	A_2	*
*	t_i^1	t_i^1	t_i^2	t_i^2	t_i^3	t_i^3	t_i^4	t_i^4	t_i^5	t_i^5	*		
*	t_i^1	t_i^1	t_i^2	t_i^2	t_i^3	t_i^3	t_i^4	t_i^4	t_i^5	t_i^5	*		
*	x	x	*	*	y	y	*	*	z	z	*		
*	x	x	*	*	y	y	*	*	z	z	*		
*	A_1	A_1	A_1	A_1	A_1	A_1	A_1	A_1	A_1	A_1	A_1		

This arrangement is possible because all of these triples are part of the matching, so they are disjoint and no element from X, Y or Z participates more than a single time.

Step 2 — Now we have left only "type II" and "type III" tiles. For each triple t_i, $i = n+1 \ldots m$ we arrange the tiles in the following way:

*	A_3	A_3	A_3	A_3	A_3	A_3	A_3	A_3	A_3	A_3	*
*	A_2	A_2	A_2	A_2	A_2	A_2	A_2	A_2	A_2	A_2	*
*	A_2	A_2	A_2	A_2	A_2	A_2	A_2	A_2	A_2	A_2	*
*	t_i^1	t_i^1	t_i^2	t_i^2	t_i^3	t_i^3	t_i^4	t_i^4	t_i^5	t_i^5	*
*	t_i^1	t_i^1	t_i^2	t_i^2	t_i^3	t_i^3	t_i^4	t_i^4	t_i^5	t_i^5	*
*	x	x	*	*	y	y	*	*	z	z	*

The right side of rightmost column and the left side of the leftmost column of the rectangles from steps 1 and 2 are composed of the single symbol *. Attach those rectangles one next to the other and get an arrangement without errors.

$(X, Y, Z, T) \in$ 3DM $\Leftarrow S \in$ Rectangle Tiling

It's easy to conclude from the construction 3 things:

1. "Type I" tiles have to be on the bottom row.
2. "Type III" tiles have to be on the top row.
3. For each triple t_i, there exists a single arrangement without errors for its 12 "type II" tiles. We call each such arrangement rectangle i.

From 1 and 2 we can see that "type I" tiles can be attached only to "type II" tiles. When there is a common seam between a "type I" tile and a rectangle i, the whole bottom part of the rectangle has to be attached with "type I" tiles. Those tiles define a match in M. It's known that $S \in$ Rectangle Tiling, so all "type I" tiles are attached to any rectangle, so we got n elements in M.

Conclusion: We have shown the decision problem Rectangle Tiling is $\mathcal{NP}$-Complete even for the dimensions $n \times 3$. Therefore, it's obvious that discovering the entropy for such a multiset is an $\mathcal{NP}$-Hard problem. □

6. Conclusions and open problems

As proven by Amir and Parienty [2], for infinite alphabet the one-dimensional tiling problem can be solved in $O(n \log n)$. We have shown that the two-dimensional tiling problem is $\mathcal{NP}$-Complete even for rectangles of width 3. An open problem is knowing what is the time complexity of the two-dimensional tiling problem for rectangles of width 2.

The idea of using patches for indexing, presented by Amir and Parienty [2], is not viable if tiling cannot be done efficiently. In this paper we showed a PTAS for the square tiling problem over fixed finite alphabets. An interesting open question is whether square tiling over an infinite alphabet is also approximable.

An intriguing direction is, perhaps, using rectangles, rather than squares for indexing, since we have shown that rectangle tiling is polynomially computable for "long and skinny" rectangles over a finite fixed alphabet. Indeed, for indexing purposes, the entire square will rarely be sought. Thus the results of this paper bring encouraging evidence to the proposal of utilizing patches for indexing.

References

[1] A. Amir, A. Apostolico, G.M. Landau, G. Satta, Efficient text fingerprinting via Parikh mapping, J. Discrete Algorithms 1 (5–6) (2003) 409–421.
[2] A. Amir, H. Parienty, Towards a theory of patches, in: J. Kalgren, J. Tarhio, H. Hyyrö (Eds.), Proc. 16th Symposium on String Processing and Information Retrieval (SPIRE), in: LNCS, vol. 5721, Springer, 2009, pp. 254–265 (best paper award).
[3] J.F. Blinn, M.E. Newel, Texture and reflection in computer generated images, ACM Commun. 19 (10) (1976) 542–547.
[4] E.E. Catmull, Computer display of curved surfaces, in: Proceedings of the IEEE Conference on Computer Graphics, Pattern Recognition, and Data Structure, 1975, pp. 11–17.
[5] F.E. Cohen, Folding the sheets: using computational methods to predict the structure of proteins, in: E.S. Lander, M.S. Waterman (Eds.), Calculating the Secrets of Life: Contributions of the Mathematical Sciences to Molecular Biology, National Academy Press, 1995, pp. 236–271.
[6] P.D. Dobson, Y.-D. Cai, B.J. Stapley, A.J. Doig, Prediction of protein function in the absence of significant sequence similarity, Curr. Med. Chem. 11 (16) (2004) 2135–2142.
[7] B. Epshtein, S. Ullman, Identifying semantically equivalent object fragments, in: Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), vol. 1, 2005, pp. 2–9.
[8] Collins, et al., International human genome sequencing consortium. Initial sequencing and analysis of the human genome, Nature 409 (6822) (2001) 860–921.
[9] J.C. Vender, et al., The sequence of the human genome, Science 291 (5507) (2001) 1304–1351.
[10] W.B. Frakes, R. Baeza-Yates, Information Retrieval Data Structure and Algorithms, Prentice Hall, 1992.
[11] Dan Gusfield, Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology, Cambridge University Press, 1997.
[12] F. Karlsson, A. Voutilainen, J. Heikkilä, A. Anttila, Constraint Grammar. A Language Independent System for Parsing Unrestricted Text, Mouton de Gruyter, 1995.

[13] R. Kolodny, P. Koehl, L. Guibas, M. Levitt, Small libraries of protein fragments model native protein structures accurately, J. Mol. Biol. 323 (2) (2002) 297–307.
[14] G. Lu, Indexing and retrieval of audio: a survey, Multimed. Tools Appl. 15 (3) (2001) 269–290.
[15] M.J. Magazine, M.-S. Chern, A note on approximation schemes for multidimensional knapsack problems, Math. Oper. Res. 9 (2) (1984) 244–247.
[16] C.D. Manning, P. Raghavan, H. Schütze, Introduction to Information Retrieval, Cambridge University Press, 2008.
[17] R.J. Parikh, On context-free languages, J. ACM 14 (4) (1966) 570–581.
[18] M. Vidal-Naquet, S. Ullman, E. Sali, A fragment-based approach to object representation and classification, in: Proc. 4th International Workshop on Visual Form (IWVF), in: LNCS, vol. 2059, Springer, 2001, pp. 85–102.
[19] G. Salton, M.J. McGill, Introduction to Modern Information Retrieval, Computer Series, McGraw-Hill, New York, 1983.
[20] P. Shilane, T.A. Funkhouser, Distinctive regions of 3d surfaces, ACM Trans. Graph. 26 (2) (2007).
[21] C. Srisuwannapa, P. Chamsethikul, An exact algorithm for the unbounded knapsack problem with minimizing maximum processing time, J. Comput. Sci. 3 (3) (2007) 138–143.
[22] M. Stricker, M. Swain, The capacity of color histogram indexing, in: Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1994, pp. 704–708.