Journal of Discrete Algorithms 12 (2012) 61–73

Contents lists available at SciVerse ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Towards a theory of patches [☆]

Amihood Amir [a,b,*,1], Haim Paryenty [a,2]

[a] Department of Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel
[b] Department of Computer Science, Johns Hopkins University, Baltimore, MD 21218, United States

ARTICLE INFO

Article history:
Received 2 March 2011
Received in revised form 1 December 2011
Accepted 15 December 2011
Available online 30 December 2011

Keywords:
Multi-dimensional indexing
Tiling
Patches

ABSTRACT

Many applications have a need for indexing unstructured data. It turns out that a similar ad-hoc method is being used in many of them – that of considering small particles of the data.
In this paper we formalize this concept as a tiling problem and consider the efficiency of dealing with this model in the pattern matching setting.
We present an efficient algorithm for the one-dimensional tiling problem, and the one-dimensional tiled pattern matching problem. We prove the two-dimensional problem is hard and then develop an approximation algorithm with an approximation ratio converging to 2. We show that other two-dimensional versions of the problem are also hard, regardless of the number of neighbors a tile has.

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 brings 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.

Dictionaries and concordances were in use by scholars for generations. By the late 1940's 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 [8,18,23]. 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.

1.1. Computational Biology

The three-dimensional structure of the protein plays an important role in its functionality and as F. Cohen [3] writes "...similar sequences yield similar structures, but quite distinct sequences can produce remarkably similar structures" [11].

[☆] A preliminary version of this paper received the *Best Paper Award* at the *String Processing and Information REtrieval* (*SPIRE*) Conference, at Saariselkä, Finland, August 2009.
[*] Corresponding author at: Department of Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel. Tel.: +972 3 531 8770.
E-mail addresses: amir@cs.biu.ac.il (A. Amir), haimpa@gmail.com (H. Paryenty).
[1] Partly supported by NSF grant CCR-09-04581, ISF grant 347/09 and an Israel–Korea bi-national research grant.
[2] This paper is part of this author's M.Sc. thesis, supported in part by an Israel–Korea bi-national research grant.

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 rate of development of 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 [15]) is taken, where the protein structure is represented as letter strings using structural alphabets.

1.2. 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 [14]. 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 [20].) 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?

1.3. 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 is it 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.

1.4. Audio indexing

Indexing large audio archives has emerged recently [17] 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.

2. Intuition of main idea

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. [6,29]), 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. [24]).

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 [25]. It has been used in Natural Language Processing as well (e.g. [14]).

The human genome projects [4,28] and other genome's 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.

In this paper, we seek to combinatorially define the idea of *patches*, and rigorously analyze its possibilities. This paper is devoted to the model definitions and initial technical results. When modeling real life applications, one needs to "clean up" and abstract many phenomena. For example, the shape, size, and dimension of the fragments, as well as the criteria for attaching adjacent pieces are application dependent.

For simplicity we assume an "exact match" in joining pieces. The major function necessary for a patch metric, is constructing the full object from its fragments. This task is interesting in its own right and has drawn attention in the pattern recognition community [31].

3. Combinatorial definition

Our first task developing a combinatorial definition for the "real world" problem of pattern matching in a patch model. The definition below abstracts the problem to the combinatorial realm.

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 $A = \{a_{0,0}, \ldots, a_{0,n-1}, \ldots\ldots, a_{n-1,0}, \ldots, a_{n-1,n-1}\}$ and $\forall i, j$ $a_{i,j} = \begin{bmatrix} m_{i,j}, m_{i,j+1} \\ m_{i+1,j}, m_{i+1,j+1} \end{bmatrix}$.

The problem we are concerned with is the inverse.

Definition 2. The problem of *constructing an image from patches* is defined as follows:
INPUT: $A = \{a_0, \ldots, a_{n^2-1}\}$ 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.

Example.

$$A = \left\{ \begin{bmatrix} a,b \\ a,a \end{bmatrix}, \begin{bmatrix} a,a \\ b,a \end{bmatrix}, \begin{bmatrix} a,a \\ a,a \end{bmatrix}, \begin{bmatrix} b,b \\ a,b \end{bmatrix}, \begin{bmatrix} b,b \\ a,a \end{bmatrix}, \begin{bmatrix} a,b \\ a,a \end{bmatrix}, \begin{bmatrix} a,a \\ b,b \end{bmatrix}, \begin{bmatrix} b,a \\ b,a \end{bmatrix}, \begin{bmatrix} b,a \\ a,b \end{bmatrix} \right\}.$$

The patches in set A can be constructed into text M,

$$M = \begin{pmatrix} a & b & b & a \\ a & a & a & b \\ b & b & a & a \\ a & b & a & a \end{pmatrix}.$$

4. One-dimensional tiling

We start by simplifying the problem to one dimension.
Let Σ be an alphabet. Denote by $\alpha = \langle x, y\rangle$ a pair over alphabet Σ^2, $(x, y \in \Sigma)$.
Let A be a set $A = \{\alpha_1, \ldots, \alpha_n\}$, $\alpha_j \in \Sigma^2$; $j = 1, \ldots, n$.

Definition 3. A is called a *tiling* if its elements can be arranged in order $\alpha_1, \ldots, \alpha_n$ such that $\forall i$, $1 \leqslant i \leqslant n-1$, $y_i = x_{i+1}$.

4.1. One-dimensional algorithm

The one-dimensional version of the algorithm is actually finding an Eulerian path in a de Bruijn graph [5,10,22].

Definition 4. An *n-dimensional de Bruijn graph over alphabet Σ* is a directed graph (V, E) defined as follows:
NODES: Strings of length n over Σ, i.e. $V \subseteq \Sigma^n$.
EDGES: Let $v, u \in V$. There is an edge $\overline{vu}$ if $v = s_1s_2s_3\cdots s_n$ and $u = s_2s_3\cdots s_nt$.

De Bruijn graphs have been well studied and have many practical applications (see e.g. coding [13] and sequencing [21]).

Our tiles can be taken as nodes in a two-dimensional de Bruijn graph. An *Eulerian path* in a graph is a trail that visits every edge exactly once. Our problem is, then, finding an Eulerian path in a two-dimensional de Bruijn graph. While finding the number of Eulerian paths in an undirected graph is $\#\mathcal{P}$-complete [2], the B.E.S.T. theorem [26,27] shows that in directed graphs it can be done in polynomial time. In the special case we are considering, Euler [7] already proved a necessary condition.

The one-dimensional tiling problem is, therefore, a classic problem in Graph Theory. However, we use the techniques of this algorithm for the two-dimensional tiling approximation algorithm. Thus, for the sake of completeness, we provide the constructive proof of the algorithm that constructs a tiling.

Definition 5. Let $\alpha, \beta \in A$. There is a *connection between α and β* if there exists $A_1 \subseteq A$, such that A_1 is a tiling, and $\alpha, \beta \in A_1$.

Let $a \in \Sigma$. Denote by $\|a\|_x$ ($\|a\|_y$) the number of times a appears in the left (right) side of a pair in A.

Let $\alpha \in A$. Denote by $x_\alpha \in \Sigma$ ($y_\alpha \in \Sigma$) the symbol on the left (right) side of pair α.

A tiling $\alpha_1, \ldots, \alpha_n$ is *cyclic* if $x_{\alpha_1} = y_{\alpha_n}$.

Note that a cyclic tiling can actually start at any pair in the tiling, since the last pair can be connected to the first pair.

Example. $\langle a,b\rangle$, $\langle b,c\rangle$, $\langle c,d\rangle$, $\langle d,a\rangle$ can be tiled as $\langle b,c\rangle$, $\langle c,d\rangle$, $\langle d,a\rangle$, $\langle a,b\rangle$ and also as $\langle c,d\rangle$, $\langle d,a\rangle$, $\langle a,b\rangle$, $\langle b,c\rangle$.

The following lemma gives necessary and sufficient conditions for tiling.

Lemma 1. *Let Σ be the alphabet symbols occurring in A. A is a tiling iff*:

(a.1) *At least $|\Sigma| - 2$ characters a, fulfill the condition $\|a\|_x = \|a\|_y$.*
(a.2) $\forall a \in \Sigma$, $\|a\|_y - 1 \leqslant \|a\|_x \leqslant \|a\|_y + 1$.
(a.3) *If there is an $a \in \Sigma$, such that $\|a\|_x = \|a\|_y + 1$, then there exists $b \in \Sigma$ such that $\|b\|_x + 1 = \|b\|_y$.*
(b) *$\forall \alpha, \beta \in A$ there is a connection between α and β.*

Proof. $\Rightarrow$ simple.

$\Leftarrow$ Our problem is reducible to the directed version of Euler's *Königsberg Bridge problem*. We define the problem below. □

Definition 6. The *Directed Königsberg Bridge problem* is defined as follows:

INPUT: A directed multi-graph $G = (V, E)$.

OUTPUT: Find a path that traverses all edges of the graph, visiting every edge exactly once (it is permissible to visit a vertex multiple times).

Euler [7] showed a necessary condition for solving the undirected version of the problem, and Hierholzer [12] proved that the undirected version of the problem can be solved if and only if the graph is connected, and there are exactly two or zero nodes of odd degree.

The reduction is as follows: Construct a directed multi-graph $G = (V, E)$, whose vertices are the alphabet letters of Σ. For pair $\alpha = \langle x, y\rangle$ there is a directed edge from vertex x to vertex y.

For the sake of completeness, we provide the explicit tiling construction.

Assume the conditions hold. The algorithm below tiles A.

We construct the tiling from left to right.

When tiling A elements, 2 situations are possible.

1. The left symbol (x_1) of the first pair is equal to the right symbol (y_n) of the last pair. This condition implies a cyclic tiling, so it is not important which element is the first one.
2. The left symbol (x_1) of the first pair is different from the right symbol (y_n) of the last pair ($x_1 \neq y_n$). In this situation the first element can be found by seeking an alphabet symbol a for which $\|a\|_x = \|a\|_y + 1$.

By checking which of the above conditions hold, the first element can be found.

The algorithm then proceeds in a greedy fashion, to find a next matching pair. If all pairs are used, we are done. Otherwise, whenever it gets stuck, the algorithm starts a new tiling. As will be seen, all tilings except for, possibly, the first one are cyclic, so they all start and finish with the same character.

In the last phase the algorithm connects all the tilings created into a single tiling. Condition (b) makes this connection possible.

A pseudo-code of the algorithm is presented in Fig. 1.

```
Tiling (A)
1   $S \leftarrow A$
2   If $\exists a$, $\|a\|_x = \|a\|_y + 1$
3     Choose $\alpha$, such that $x_\alpha = a$
4   else
5     Choose any $\alpha$
6   First element $\leftarrow \alpha$
7   $\gamma \leftarrow \alpha$
8   $S \leftarrow S - \alpha$
9   While $S \neq \emptyset$
10    If $\exists \beta, x_\beta = y_\gamma$
11      Next element $\leftarrow \beta$
12    else
13      Choose any $\beta$
14      Start new temporary tiling with $\beta$
15    $\gamma \leftarrow \beta$
16    $S \leftarrow S - \beta$
17  end while
18  For every $a \in \Sigma$
19    If $\sharp$ tilings $= 1$
20      Stop
21    else
22      If $\sharp$ tilings containing $a \geqslant 2$
23        Connect all tilings containing $a$
24  end for
```

Fig. 1. Tiling A elements. Lines 1–8 are the first phase, finding the first element. Lines 9–17 are the second phase, making tilings from all A members. Lines 18—24 are the last phase, connecting all tilings.

```
Connecting tilings
1   Let $A = \alpha_1, \ldots, \alpha_n$ and $B = \beta_1, \ldots, \beta_m$ be two tilings, both containing $a$, and assume $B$ is cyclic.
2   If $a \neq x_{\beta_1}$
3     Rotate $B$ until $a = x_{\beta_1}$
4   Insert $B$ into $A$, in place where $A$ has $a$
```

Fig. 2. Connecting A and B elements.

We now describe the implementation of the tiling connecting phase. The conditions of the lemma mean there is no more than one symbol a for which $\|a\|_x = \|a\|_y + 1$. So if such a character exists it is chosen in the first phase. If for every symbol a, $\|a\|_x = \|a\|_y$, then all tilings are cyclic. The following claim guarantees that for the case where there is an a, $\|a\|_x = \|a\|_y + 1$, the *first* tiling is not cyclic.

Claim. *If $\exists a$, $\|a\|_x = \|a\|_y + 1$ then the first tiling is not cyclic.*

Proof. The first pair α is chosen such that $x_\alpha = a$. Any last pair β where $y_\beta = a$ can be further extended, since there is one more a on the right than on the left. □

We, therefore, have that, at the end of second phase, all tilings, except for possibly the first one, must be cyclic.

The following pseudo-code connects two tilings that both contain the symbol a. (See Fig. 2.)

The idea behind this subroutine is as follows. B is cyclic, thus it can be rotated to start with a pair whose left symbol is a. It also will end with a pair whose right symbol is a. Therefore, B can be inserted into A between two pairs α_i, α_{i+1} where $y_{\alpha_i} = x_{\alpha_{i+1}}$.

Example. Let $A = \langle a,b\rangle, \langle b,c\rangle, \langle c,d\rangle, \langle d,e\rangle$, and $B = \langle f,g\rangle, \langle g,h\rangle, \langle h,c\rangle, \langle c,k\rangle, \langle k,f\rangle$.

c is in both tilings. B is cyclic thus it can be rotated to start with c, i.e. $B = \langle c,k\rangle, \langle k,f\rangle, \langle f,g\rangle, \langle g,h\rangle, \langle h,c\rangle$.

B can now be inserted between the second and third pairs of A resulting in the single tiling: $\langle a,b\rangle$, $\langle b,c\rangle$, $\langle c,k\rangle$, $\langle k,f\rangle$, $\langle f,g\rangle$, $\langle g,h\rangle$, $\langle h,c\rangle$, $\langle c,d\rangle$, $\langle d,e\rangle$.

Checking the conditions. We now present the implementation of checking the conditions.

Condition (a) is easy to check.

The following pseudo-code checks condition (b) (see Fig. 3).

Claim. *The algorithm's complexity is $O(n\log n + |\Sigma|n)$.*

Proof. Condition (b) is checked in time $O(n\log n)$. If $|\Sigma|$ is polynomial in n it can be done in time $O(n)$. The space complexity is $O(n)$.

```
CHECK CONDITION (B)
1  Sort all pairs in $A$ by their left characters.
2  Put in a bin all pairs whose left symbol is $a$. Push $a$ into queue $q1$,
   and push into queue $q2$ every symbol appearing to the right of $a$ in some pair.
3  Pop from $q2$ a symbol $b$. If it is not in $q1$ then repeat stage 2) with $b$.
4  If in the end, all pairs are in the bin, then there is a connection between all characters in $A$.
```

Fig. 3. Checking if there is a connection between all elements of A.

The algorithm for tiling A takes time $O(n \log n)$ if implemented as follows: First, sort all pairs. Then convert the alphabet to $\Sigma = \{1, \dots, n\}$, allowing direct access when seeking pairs. Tilings can be connected in time $O(|\Sigma| n)$. The total is $O(n \log n + |\Sigma| n)$ time.

Claim *(Correctness). If A has a tiling then the algorithm finds one.*

Proof. The algorithm finds a tiling for A: We have established that except for, possibly, the first tiling all tilings are cyclic. The only thing left to prove is that we can connect the tilings. Every set of tilings has at least a pair with a common alphabet symbol, otherwise condition (b) is not satisfied. Therefore, inductively, we can connect all tilings until only one is left. $\square$

4.2. Pattern matching

The *tiled pattern matching problem* is defined as follows.

Definition 7. *Tiled pattern matching of P in A* is:
Input: $A = \{\alpha_1, \dots, \alpha_n\}$, $P = \{\tau_1, \dots, \tau_m\}$, $\alpha_i, \tau_j \in \Sigma^2$, $i = 1, \dots, n;\ j = 1, \dots, m$.
Output: Decide if $\exists$ tiling of A and tiling of P such that P tiling is a substring of A tiling.

Since P must be a substring of A, then $P \subseteq A$. In the first stage check if P itself is a tiling.

If P is cyclic, we can check A with all possible beginnings of P. However, another strategy is better. It suffices to check if $A \setminus P$ is a tiling and if there is a common alphabet symbol to P and $A \setminus P$. If P is not cyclic, let τ_1, τ_k be the first and last pairs of P, respectively. Let $\alpha_1, \dots, \alpha_j$ be a tiling of $A \setminus P$. There is a tiled matching of P in A if either $x_{\alpha_1} = y_{\tau_k}$, $y_{\alpha_j} = x_{\tau_1}$, or if there exist two pairs $\alpha_i, \alpha_{i+1} \in A \setminus P$, where $y_{\alpha_i} = x_{\tau_1}$, and $x_{\alpha_{i+1}} = y_{\tau_k}$.

5. Two-dimensional tiling

We show that two-dimensional tiling is hard, consider similar problems, and find an approximation algorithm.

5.1. History

5.1.1. Wang tiles

The patches we defined are called Wang tiles. Wang introduced these tiles in 1961 [30] as a class of formal system. The questions he grappled with are tessellation, and the tiles are forms rather than single elements. Therefore, while our patches are Wang tiles, our model is very different.

5.1.2. Square Tiling

A very similar issue to our work is *Square Tiling*. In this model one is given a set C of "colors", collection $T \subseteq C^4$ of tiles, where $\langle a, b, c, d\rangle$ denotes a tile whose top, right, bottom, and left sides are colored by a, b, c, d respectively, and a positive integer $N \leqslant |C|$. One needs to decide if there is a tiling of an $N \times N$ square using the tiles in T, i.e., an assignment $f : N \times N \to T$ of a tile $f(i, j) \in T$ to each ordered pair i, j, $1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant N$, such that

1. if $f(i, j) = \langle a, b, c, d\rangle$ and $f(i+1, j) = \langle a', b', c', d'\rangle$, then $a = c'$, and
2. if $f(i, j) = \langle a, b, c, d\rangle$ and $f(i, j+1) = \langle a', b', c', d'\rangle$, then $b = d'$.

The above problem can be shown to be $\mathcal{NP}$-hard, using transformation from directed Hamiltonian path [9].

Another Square Tiling problem is defined in [19]. Given a set of square tile types $T = \{t_0, \dots, t_k\}$, together with two relations $H, V \subseteq T \times T$ (the horizontal and vertical compatibility relations, respectively). Also given an integer n in binary. An $n \times n$ tiling is a function $f : \{1, \dots, n\} \times \{1, \dots, n\} \to T$ such that (a) $f(1, 1) = t_0$, and (b) for all $i, j (f(i, j), f(i+1, j)) \in H$, and $(f(i, j), f(i, j+1)) \in V$. *TILING* is the problem of deciding, given T, H, V and n, whether an $n \times n$ tiling exists. In [19] Papadimitriou claims the following:

1. Tiling is $\mathcal{NEXP}$-complete.
2. Tiling becomes $\mathcal{NP}$-complete if n is given in unary.
3. It is undecidable to tell, given T, H, and V, whether an $n \times n$ tiling exists for all $n > 0$.

The Papadimitriou version of the tiling problem is more general than the Garey and Johnson version, because H, V, does not have the restraints defined in the previous version. In any event, our problem is more restrictive than both, because the tiles are actual entities rather than elements of a range, so we had hoped it is polynomially solvable. The next subsection dashes these hopes.

5.2. Two-dimensional tiling is $\mathcal{NP}$-hard

Theorem 1. *The problem of constructing an image from patches as defined in Definition 2, is hard.*

Proof. By reduction from Levin's tiling problem [16]. □

Definition 8. A patch A may be *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.

A *tiled square descriptor* f is a function that, given a square of patches placed next to each other, outputs its first row and the list of patches used.

Levin's tiling problem is inverting the tiled square descriptor, i.e.

INPUT: A row R of n patches placed next to each other, and a multiset S of $(n-1)n$ patches.

DECIDE: Whether there exists a square of patches placed next to each other, whose first row is R and where the rest of the tiles used are exactly those in the multiset S. We call such a square a *Levin Square*.

Levin showed [16] that Levin's tiling problem is $\mathcal{NP}$-complete.

We polynomially reduce Levin's tiling problem to our two-dimensional image construction problem as follows.

Given the input for Levin's tiling problem, and let the first row R be the n patches $\{x_0, \ldots, x_n\}$.

We have two tasks ahead of us. The first is to show that the problem of placing patches *next to each other* in Levin's tiling problem is equivalent to that of constructing an image from *overlapping* patches. The second challenge is to force the patches in row R to be the first row in the image constructed from our patches.

Definition 9. The *Square Tiling problem* is defined as follows:

INPUT: A multiset S of n^2 patches.

DECIDE: Whether there exists a square of patches placed next to each other, whose patches are exactly those in the multiset S.

The equivalence of Square Tiling and our Image Construction problem is established by the following lemma.

Lemma 2. *n^2 patches can be placed in an $n \times n$ Levin Square iff those n^2 patches can be used to construct an $(n+1) \times (n+1)$ image. (Note that the size of the Levin Square is given in patches and the size of the image is given in pixels.)*

Proof. An $n \times n$ Levin Square constructs a $2n \times 2n$ matrix of symbols. Erasing all even columns and rows, excluding the last even column and row, produces an $(n+1) \times (n+1)$ image constructed from the given patches.

Conversely, given an $(n+1) \times (n+1)$ image, doubling all columns and rows excluding the first and last, produces a Levin Square constructed from the given patches. □

We now reduce Levin's tiling problem to the Square Tiling problem. Assume the first row is $\{x_0, \ldots, x_n\}$. Replace this set of patches by n new patches $\{x'_0, \ldots, x'_n\}$ where the symbols on the top of $x_0, \ldots, x_n$ are changed to new symbols which do not exist in the alphabet, but that cause $x'_0, \ldots, x'_n$ to be placed next to each other in that order. Specifically, let $\{a_1, \ldots, a_{n+1}\}$ be new symbols not in Σ, replace the top symbols of patch x_i by a_i, a_{i+1}.

Example. If the first row is:

$$\left\{\begin{bmatrix}1,2\\3,4\end{bmatrix}, \begin{bmatrix}2,5\\4,6\end{bmatrix}, \begin{bmatrix}5,3\\6,3\end{bmatrix}, \begin{bmatrix}3,7\\3,0\end{bmatrix}\right\}$$

then change it to:

$$\left\{\begin{bmatrix}a_1,a_2\\3,4\end{bmatrix}, \begin{bmatrix}a_2,a_3\\4,6\end{bmatrix}, \begin{bmatrix}a_3,a_4\\6,3\end{bmatrix}, \begin{bmatrix}a_4,a_5\\3,0\end{bmatrix}\right\}$$

where $a_1, a_2, a_3, a_4, a_5 \notin \Sigma$.

Now, every construction of a Levin Square is forced to put on top in the given order, the patches $x'_1, \dots, x'_n$, and is thus a solution to Levin's tiling problem. No solution to our problem means no solution to Levin problem.

5.3. *Matching equal symbols*

Lemma 2 allows us to consider the problem of matching patches when their borders match, without recourse to an overlap. We have seen that the image reconstruction problem is $\mathcal{NP}$-hard. It is therefore natural to ask whether relaxing the condition that allows placing two patches next to each other yields a more tractable problem.

Definition 10. Let $\Sigma = \{1, 2, \dots, |\Sigma|\}$. A patch A may be *closely placed* to the right (left, top, bottom) of patch B if the pair of symbols $\langle b_1, b_2 \rangle$ on the right (left, top, bottom) side of B satisfy $|a_1 - b_1| \leqslant k$ and $|a_2 - b_2| \leqslant k$, where $\langle a_1.a_2 \rangle$ is the pair on the left (right, bottom, top) of patch A, and k is a given fixed constant.

The *Near Square Tiling problem* is defined as follows:

INPUT: A multiset S of n^2 patches.

DECIDE: Whether there exists a square of patches *closely placed* next to each other, whose patches are exactly those in the multiset S.

Theorem 2. *The Near Square Tiling problem is $\mathcal{NP}$-hard.*

Proof. Given input n patches over alphabet Σ, multiply every symbol in the alphabet by k getting a new alphabet $\Sigma' = \{1, \dots |\Sigma|k\}$. Now the only solution to the Near Square Tiling problem is the solution to the Square Tiling problem. □

5.4. *An approximation algorithm*

Having established the hardness of the image reconstruction problem, we now seek ways to approximate the image. We need to decide first what it is we are trying to approximate. Granted that we cannot efficiently reconstruct the image, we can try to develop an efficient algorithm that reconstructs the largest square it can from the patches. Another possibility, and this is the one we took, is reconstructing an image from *all* the patches, but allowing *errors*, where two patches that are placed next to each other but whose symbols don't match, introduce an error. The square tiling of n patches actually has $2n - 2\sqrt{n}$ matches. The algorithm below constructs a square with at least n matches, thus we have an algorithm that approximates the matches within a factor that converges to $\frac{1}{2}$ as n grows to infinity.

We would like to construct $\sqrt{n}$ matching rows. If that were done, we would have $n - \sqrt{n}$ matches. Unfortunately, we will not be able to guarantee matching rows. Rather, there will be an additional $\sqrt{n}$ errors within the rows.

For simplicity of exposition, we consider a pair of symbols $\langle x, y \rangle$, $x, y \in \Sigma$, as a single new symbol in a new alphabet $\Pi = \Sigma \times \Sigma$. Each column of two symbols in a two-dimensional patch becomes a new symbol, converting every two-dimensional patch into a one-dimensional patch.

Example. $\begin{bmatrix} a,b \\ c,d \end{bmatrix}$ will be written as $[\,ac\ bd\,]$.

The idea of the algorithm is similar to the one presented in Lemma 1, except everything is done for each of the $\sqrt{n}$ rows. As in Lemma 1 we will give treat differently the symbols which occur the same number of times on the right and left side of a tile, which we call *circular symbols*, and symbols for which $\|a\|_x \neq \|a\|_y$, which we call *uncircular*.

5.4.1. *Algorithm's idea*

The algorithm has four stages:

1. **Finding Uncircular Symbols Phase**: Find the set U of uncircular symbols in Π. Because of Lemma 1 we know that for such symbols $\|a\|_x = \|a\|_y + i_a$, where $\sum_{a \in U} i_a \leqslant \sqrt{n}$. Furthermore, we also know that $\sum i_a = s \leqslant \sqrt{n}$.
2. **Uncircular Rows Construction Phase**: For each symbol $a \in U$ for which $\|a\|_x = \|a\|_y + i_a$, start constructing i_a row in a greedy fashion as in Lemma 1 until it is impossible to continue. We now have x rows, $x \leqslant \sqrt{n}$, whose first (and possibly last) element are not circular. If no patches remain we go to the Row Length Adjusting Phase. If there are remaining patches, they are all circular. We go to the Circular Rows Construction Phase.
3. **Circular Rows Construction Phase**: As in Lemma 1 construct, in a greedy fashion, rows starting in circular symbols. Because of the reasons proven in Lemma 1, all these rows are *circular*, i.e., begin and end in the same symbol. At this point, insert all circular rows, wherever possible, into other rows, until no more such insertions are possible. We prove below that the total number of remaining rows, both circular and uncircular, are no greater than $\sqrt{n}$. The problem is that some of them may have length greater than $\sqrt{n}$ and some may have a shorter length.
4. **Row Length Adjusting Phase**: For each row whose length exceeds $\sqrt{n}$, cut it to as many rows of length $\sqrt{n}$ as possible. All these rows are now complete. The remaining sub-row of length less than $\sqrt{n}$, is added to one of the incomplete rows whose length is less than $\sqrt{n}$ (possibly introducing a mismatch).

```
Approximation algorithm
Input: set $A$ of $n$ patches
1  $B \leftarrow$ List of $s$ patches beginning with uncircular symbols.
2  While $B \neq \emptyset$
3    $x \leftarrow$ a patch from $B$
4    Greedily construct the longest row, starting with $x$ and using patches from $B$.
5    $B \leftarrow B-$ the used patches
6  end while
7  $A' \leftarrow A - B$
8  While $A' \neq \emptyset$
9    $x \leftarrow$ a patch from $A'$
10   Greedily construct the longest row, starting with $x$ and using patches from $A'$.
11   $A' \leftarrow A'-$ the used patches
12   If possible, insert the row just constructed into one of the exist rows
13 end while
14 Place all rows constructed, $L_1, \ldots, L_m$, vertically one above one in any order.
15 If $\forall i$, $1 \leqslant i \leqslant m$, $|L_i| = \sqrt{n}$ then stop.
16 else
17   While $\exists i, 1 \leqslant i \leqslant m$, $|L_i| > \sqrt{n}$
18   Choose the top $L_j$, $|L_j| > \sqrt{n}$, and remove from it the first $\sqrt{n}$ patches,
     calling this removed row $L_{cut}$
19   Choose a $L_k$, $|L_k| < \sqrt{n}$, and attach $L_j - l_{cut}$ to the right of $L_k$
20   end while
```

Fig. 4. Approximation algorithm. Lines 1–7 are phases 1 and 2. Lines 7–13 are phase 3. Lines 14–19 are phase 4.

Iterate on this phase as long as there remain incomplete rows. Note that there are no more than $\sqrt{n}$ iterations, and each one introduces at most one mismatch.
At the end of this phase we have exactly $\sqrt{n}$ rows of length exactly $\sqrt{n}$ each, and at most $\sqrt{n}$ mismatches within these rows. Placing these rows one on top of the other in any sequence gives a $\sqrt{n} \times \sqrt{n}$ matrix with at most $\sqrt{n}$ mismatches within the rows and $(\sqrt{n}-1)\sqrt{n}$ mismatches between the rows for a total of $\sqrt{n}^2 = n$ mismatches.

A pseudo-code of the algorithm is presented in Fig. 4.

5.4.2. Time

Phase 1 can be done in time $O(n \log n)$ by soring. Phase 2 can be done in linear time. Phase 3's most complex part is a *union-find* which can be implemented in time $O(n\alpha(n))$, where $\alpha(x)$ is the inverse Ackerman function. Phase 4 can be implemented in linear time. Thus the total algorithm time is $O(n \log n)$.

5.4.3. Correctness

The correctness hinges on Lemma 1, and on the fact that the patches were produced from a $\sqrt{n} \times \sqrt{n}$ image. This guarantees that there is equal number of circular symbols on the left side and the right side of the patches, and thus an equal number of uncircular symbols on the left and right sides of the patches. Therefore, all rows constructed during Phase 2, start and end with uncircular symbols, and there are no more than $\sqrt{n}$ such rows.
Lemma 3 proves that the total number of rows after Phase 3 is no larger than $\sqrt{n}$. Now Phase 4 is clear.

Lemma 3. *Given k one-dimensional rows, the greedy algorithm reconstructs at most k rows.*

Proof. For $k = 1$ it is the case of Lemma 1. Assume, for a general k, that the greedy algorithm produced rows:

$$\begin{array}{c} a_1 \cdots b_1 \\ a_2 \cdots b_2 \\ \vdots \\ a_m \cdots b_m \end{array}$$

where $m > k$.
Note the following observations:

1. It is impossible that all the a_i and b_i, $i = 1, \ldots, m$ are different pairwise, because that would make them all uncircular symbols, contradicting the fact that there are originally only k rows.
2. It is impossible that any $b_i = a_j$ for some $i, j \in \{1, \ldots, m\}$, otherwise the greedy algorithm would have joined those rows.
3. Therefore we have no more than k rows where $a_\ell \neq b_\ell$ and the rest are circular rows.
4. Each circular row has symbols that do not appear in any of the other rows, otherwise the greedy algorithm would have merged them.

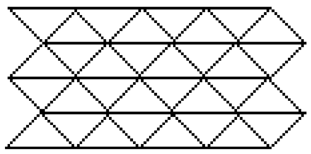


Fig. 5. Triangles tiling the plane.

This means that each of the constructed circular rows, and each of the constructed non-circular rows corresponds to a different input row. Conclude that $m \leqslant k$. $\square$

5.5. *Largest common image*

So far we have discussed the problem of constructing an image from patches. Patches can be used as a method for indexing images. In this context it is necessary to find the largest sub-image common to two given images.

Definition 11. Let A and B be two sets of patches, each of size n. The *largest common image* of A and B is the largest set of patches in the intersection of A and B that can be placed in a square.

Theorem 3. *Computing the largest common image is $\mathcal{NP}$-hard.*

Proof. By reduction from the Square Tiling problem. Let S be the set of n patches that are input to the Square Tiling problem, take $A = B = S$. The Largest Common Image of A and B is A iff there is a Square Tiling of S. $\square$

5.6. *Three neighbors*

We have seen that a one-dimensional image, where each patch connects to two neighbors, one on its right and one on its left, can be efficiently reconstructed. We have seen that a two-dimensional image, where a patch connects to four neighbors – top, bottom, left and right – is $\mathcal{NP}$-hard. What happens if each patch connects to *three* neighbors? Is the hardness a result of the number of neighbors or of the dimension?

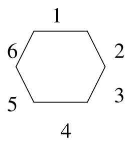
The case of a patch connecting to three neighbors is possible if the patches are equilateral triangles, rather than squares. Equilateral triangles can still tile the plane (see Fig. 5) but there are only three neighbors to every patch. Does the reduction of a degree of freedom make the tiling efficiently computable, or is it still $\mathcal{NP}$-hard?

It turns out that there is still one other equilateral polygon that can tile the plane – the hexagon. Since the hexagon has 6 neighbors, we expect that reconstructing an image from hexagons is hard. We will, indeed, see that this is the case. However, the hexagon tiling will help us solve the triangular tiling case.

5.6.1. *Hexagonal tiling*

We start by proving the intuitively true claim that hexagonal tiling is also $\mathcal{NP}$-hard. We first need to define a "square tiling" using hexagonal patches, which are non-square. This creates a somewhat "rotated" square, giving the impression of a stairway.

Definition 12. Let h be a hexagon. We *number* its top side *side* 1, and then go through the other sides in a clockwise order and number them 2 to 6.



Given n^2 hexagonal tiles with labels on their sides, a *square hexagonal tiling* of the given tiles has n columns $C_1, \dots, C_n$, where column C_i is composed of tiles $h_{j,i}$, $j = 1, \dots, n$, where the label of side 4 of $h_{j,i}$ is equal to the label of side 1 of $h_{j+1,i}$, $j = 1, \dots, n-1$. In addition, the label of side 3 of $h_{j,i}$ equals the label of side 6 of $h_{j,i+1}$, $j = 1, \dots, n$; $i = 1, \dots, n-1$, and the label of side 2 of $h_{j,i}$ equals the label of side 5 of $h_{j-1,i+1}$, $j = 2, \dots, n$; $i = 1, \dots, n-1$. See Fig. 6.

Theorem 4. *The Hexagonal Tiling problem is $\mathcal{NP}$-hard.*

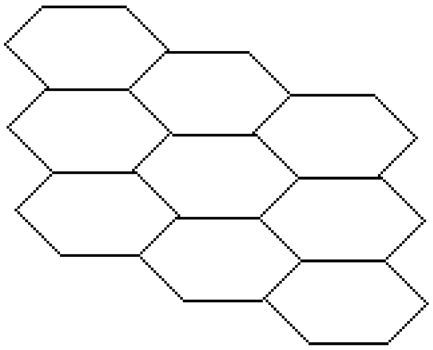
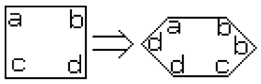
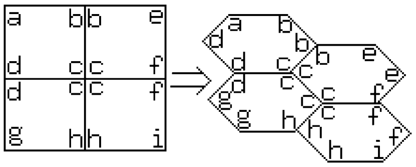


Fig. 6. Square hexagonal tiling.

Proof. We reduce the square tiling problem to the square hexagonal tiling problem as follows. Given an input of n squares patches, convert every square tile to a hexagonal tile in the following manner.



Note that there is a matching between squares iff there is a matching between hexagons.



Therefore if it is possible to organize the squares in $\sqrt{n} \times \sqrt{n}$ picture, iff it is possible to organize the hexagons. □

5.6.2. Triangle Tiling is hard

We will now reduce square hexagonal tiling to triangle tiling, proving that tiling is hard even with three neighbors.

Definition 13. The *Triangle Tiling problem* is defined as follows. Assume we have an image composed of contiguous triangles with labels on their angles, such that the labels on every two adjacent angles are equal.

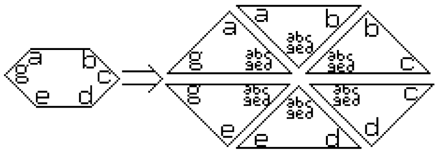
INPUT: The triangles composing the image, and the outline of the image.

OUTPUT: A tiling using the input triangles that fills out the outline of the image preserving the matching label requirement for adjacent angles.

Theorem 5. *Triangle Tiling is $\mathcal{NP}$-hard.*

Proof. We reduce the Square Hexagonal Tiling problem to the Triangle Tiling problem.

Given an input of n labeled hexagons, construct 6 labeled triangles from each hexagon in the following manner.

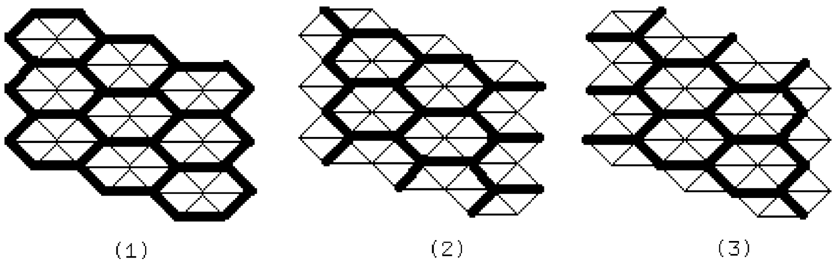


Where the letter $\substack{\text{abc}\\ \text{ged}}$ does not belong to the original alphabet. The role of the special letters is to insure that every algorithm that tiles the triangles must connect an original hexagon, yet is incapable of connecting to a hexagon triangles that originated from different hexagons. □

Lemma 4. *If it is possible to connect the triangles to a "stairway image", it is possible to construct a square hexagonal tiling from the hexagons.*

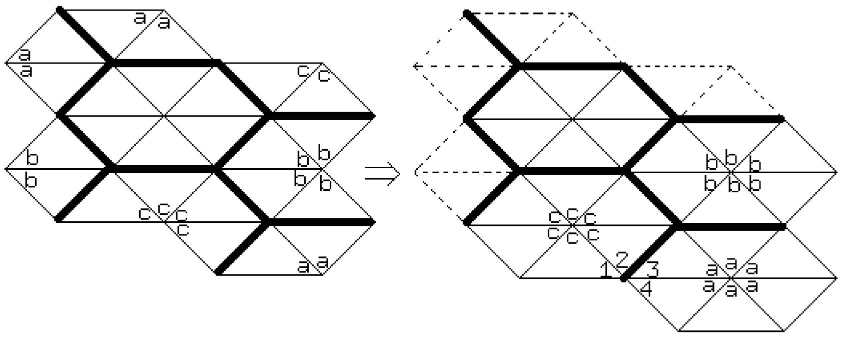
Proof. Every triangle has two letters belonging to the original alphabet, and one special letter. The special letters can only fit when the triangles have been created from the same hexagon. Therefore, given a reconstructed "stairway image" put together from the triangles, every special letter that is not located on the image outline must connect the 6 triangles that comprised an original hexagon.

There are altogether 3 different possible cases:



If the triangle reconstruction results in the first case, then we are done.

For the second case, cut the pairs of triangles from the left side and insert them in the fitting place on the right. Next, cut the pairs of triangles from the upper edge and insert them in the appropriate places on the bottom edge. For example:



Now we have a square hexagonal tiling, provided that one can always find an appropriate location to match the cutoff pairs of triangles.

Consider the above diagram. It is clear that the angle marked 1) has the same label as the angle marked 2), since they belong to the same original hexagon. For the same reason, the label of angle 3) is the same as that of angle 4). Now, 2) has the same label as 3) because their triangles are neighbors. Conclude that the labels of all angles 1), 2), 3), and 4) are equal. A similar argument applies to the right edge.

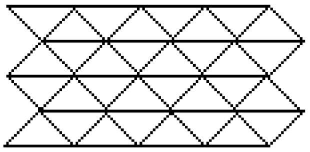
For the third case, note that it is a 180° rotation of the second case, thus it can be treated in a similar manner. □

6. Conclusion and open problems

We formalized an indexing paradigm for pattern matching as a combinatorial tiling problem. This introduced the rich field of tiling to pattern matching problems, and gives a motivation to study tiling problems where the tiles are actual counted entities.

The one-dimensional tiling problem turned out to be a classic Graph Theory problem and has an efficient solution. The two-dimensional problem is hard. We developed an approximation algorithm with an approximation ratio converging to 2. We have barely begun to scratch the surface of these problems, and much research with potential applications is still left to be done.

Some interesting examples of open problems are approximating the largest contiguous sub-image of a given set of patches, and studying the tiling problem of non-square tiles for special forms, for example, approximating an unrotated square as much as possible, as in the image below.



References

[1] A. Amir, A. Apostolico, G.M. Landau, G. Satta, Efficient text fingerprinting via Parikh mapping, Journal of Discrete Algorithms 1 (5–6) (2003) 409–421.
[2] G. Brightwell, P. Winkler, Conting Eulerian circuits is #p-complete, in: Proc. 2nd Workshop on Analytic Algorithms and Combinatorics (ANALCO), 2005, pp. 259–262.

[3] 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.
[4] F. Collins, et al., International Human Genome Sequencing Consortium, Initial sequencing and analysis of the human genome, Nature 409 (6822) (2001) 860–921.
[5] N.G. de Bruijn, A combinatorial problem, Koninklijke Nederlandse Akademie v. Wetenschappen 49 (1946) 758–764.
[6] 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.
[7] L. Euler, Solutio problematis ad geometriam situs pertinentis, Commentarii Academiae Scientiarum Petropolitanae 8 (1741) 128–140.
[8] W.B. Frakes, R. Baeza-Yates, Information Retrieval Data Structure and Algorithms, Prentice Hall, 1992.
[9] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Co., New York, 1979.
[10] I.J. Good, Normal recurring decimals, Journal of the London Mathematical Society 21 (3) (1946) 167–169.
[11] Dan Gusfield, Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology, Cambridge University Press, 1997.
[12] C. Hierholzer, Über die möglichkeit, einen linienzug ohne wiederholung und ohne unterbrechnung zu umfahren, Mathematische Annalen 6 (1873) 30–32.
[13] M.F. Kaashoek, D.R. Karger, Koorde: A simple degree-optimal distributed hash table, in: M.F. Kaashoek, I. Stoical (Eds.), Proc. 2nd International Workshop Peer-to-Peer Systems II (IPTPS), in: Lecture Notes in Computer Science, vol. 2735, Springer, 2003, pp. 98–107.
[14] F. Karlsson, A. Voutilainen, J. Heikkilä, A. Anttila, Constraint Grammar. A Language Independent System for Parsing Unrestricted Text, Mouton de Gruyter, 1995.
[15] R. Kolodny, P. Koehl, L. Guibas, M. Levitt, Small libraries of protein fragments model native protein structures accurately, Journal of Molecular Biology 323 (2) (2002) 297–307.
[16] L.A. Levin, Universal sorting problems, Problemy Peredachi Informatsii 9 (3) (1973) 265–266 (in Russian).
[17] G. Lu, Indexing and retrieval of audio: A survey, Multimedia Tools and Applications 15 (3) (2001) 269–290.
[18] C.D. Manning, P. Raghavan, H. Schütze, Introduction to Information Retrieval, Cambridge University Press, 2008.
[19] C.H. Papadimitriou, Computational Complexity, Addison–Wesley, 1994.
[20] R.J. Parikh, On context-free languages, Journal of the ACM 14 (4) (1966) 570–581.
[21] P.A. Pevzner, H. Tang, M.S. Waterman, An Eulerian path approach to DNA fragment assembly, Proc. National Academy of Sciences of the USA (PNAS) 98 (17) (2001) 9748–9753.
[22] F. Sainte-Marie, Question 48, L'Interm'ediaire Math. 1 (1894) 107–110.
[23] G. Salton, M.J. McGill, Introduction to Modern Information Retrieval, Computer Series, McGraw–Hill, New York, 1983.
[24] P. Shilane, T.A. Funkhouser, Distinctive regions of 3D surfaces, ACM Transactions on Graphics 26 (2) (2007), http://doi.acm.org/10.1145/1243980.1243981.
[25] 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.
[26] W.T. Tutte, C.A.B. Smith, On unicursal paths in a network of degree 4, American Mathemathical Monthly 48 (1941) 233–237.
[27] T. van Aardenne-Ehrenfest, N.G. de Bruijn, Circuits and trees in oriented linear graphs, Simon Stevin (Bul. Belgian Math. Soc.) 28 (1951) 203–217.
[28] J.C. Vender, et al., The sequence of the human genome, Science 291 (5507) (2001) 1304–1351.
[29] 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: Lecture Notes in Computer Science, vol. 2059, Springer, 2001, pp. 85–102.
[30] H. Wang, Proving theorems by pattern recognition, II, Bell System Technical Journal 40 (1) (1961) 1–41.
[31] L. Zhu, Z. Zhou, D. Hu, Globally consistent reconstruction of ripped-up documents, IEEE Transactions on Pattern Analysis and Machine Intelligence 30 (1) (2008) 1–13.