HAL open science

Efficient Search of Combinatorial Maps using Signatures

Stéphane Gosselin, Christine Solnon, Guillaume Damiand

To cite this version:

Stéphane Gosselin, Christine Solnon, Guillaume Damiand. Efficient Search of Combinatorial Maps using Signatures. Theoretical Computer Science, 2011, 412 (15), pp.1392-1405. 10.1016/j.tcs.2010.10.029 . hal-00567332v2

HAL Id: hal-00567332

https://hal.science/hal-00567332v2

Submitted on 23 Jun 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Efficient Search of Combinatorial Maps using Signatures

Stéphane Gosselin, Guillaume Damiand, Christine Solnon

Université de Lyon, CNRS
Université Lyon 1, LIRIS, UMR5205, F-69622, France

Abstract

In this paper, we address the problem of computing canonical representations of n-dimensional combinatorial maps and of using them for efficiently searching for a map in a database. We define two combinatorial map signatures: the first one has a quadratic space complexity and may be used to decide of isomorphism with a new map in linear time whereas the second one has a linear space complexity and may be used to decide of isomorphism in quadratic time. We show that these signatures can be used to efficiently search for a map in a database.

Keywords:
Combinatorial map, signature, map isomorphism

1. Motivations

Combinatorial maps are nice data structures for modelling the subdivision of a space into cells. First defined in 2D [7, 15, 9, 3], they have been extended to nD [2, 11, 12] and model the subdivision of an object in cells, and all the adjacency and incidence relations in any dimension. Hence, combinatorial maps are often used to model the partition of an image in regions and to describe the topology of this partition (e.g., [1] for $2D$ images and [4] for $3D$ images). There exist efficient image processing algorithms using this topological information.

Our goal is to define new algorithms for classifying images modelled by combinatorial maps. More precisely, we propose to characterize image classes

Email addresses: `stephane.gosselin@liris.cnrs.fr` (Stéphane Gosselin), `guillaume.damiand@liris.cnrs.fr` (Guillaume Damiand), `christine.solnon@liris.cnrs.fr` (Christine Solnon)

Author manuscript, published in Journal of Theoretical Computer Science (TCS), Elsevier, 2011, 412, pp.1392-1405. Thanks to Elsevier.
The original publication is available at https://dx.doi.org/10.1016/j.tcs.2010.10.029

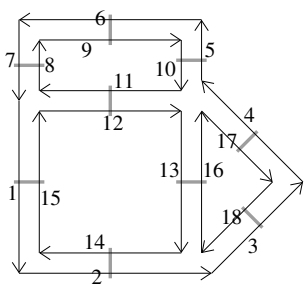
by extracting patterns (submaps) which occur frequently in these classes. Finding frequent patterns in large databases is a classical data mining problem, the tractability of which highly depends on the existency of efficient algorithms for deciding if two patterns are actually different or if they are two occurrences of a same object. Hence, if finding frequent subgraphs is intractable in the general case, it may be solved in incremental polynomial time when considering classes of graphs for which subgraph isomorphism may be solved in polynomial time, such as trees or outerplanar graphs [8].

In this paper, we address the problem of computing canonical representations of combinatorial maps which may be used to efficiently search for a map in a database. This work is related to [10], which introduces a polynomial algorithm for deciding of the isomorphism of ordered graphs (i.e., graphs such that the set of nodes adjacent to every node is ordered), based on a vertex labelling. Recently, this work has been extended to combinatorial maps by proposing a polynomial algorithm for map and submap isomorphism based on a traversal of the map [6].

Contribution. In this paper, we define canonical representations of combinatorial maps which are based on these traversal and labelling principles. More precisely, we define two map signatures. Both signatures may be computed in quadratic time with respect to the size of the map. The first signature (called Set Signature) is a set of words and is modelled by a lexicographical tree. It has a quadratic space complexity and it allows us to decide if a new map is isomorphic to a map modelled by this signature in linear time. The second signature (called Word Signature) is a word and has a linear space complexity. As a counterpart, isomorphism with a new map is quadratic.

We also show that these signatures can be used to efficiently search for a map into a database of maps. More precisely, each map of the database is modelled by its signature and the different signatures are merged into a tree. Space and time complexities depend on the considered signature (Set or Word Signature): the first one is faster but needs more space.

Outline. Basic definitions on combinatorial maps are recalled in section 2. The two map signatures are defined in section 3. We show how to use these signatures to model a database of maps in section 4. In sections 3 and 4, we only consider connected maps. We show how to extend this work to non connected maps in section 5. Finally, we experimentally evaluate our work in section 6.



d	1	2	3	4	5	6	7	8	9
$\beta_1(d)$	2	3	4	5	6	7	1	9	10
$\beta_2(d)$	15	14	18	17	10	9	8	7	6

d	10	11	12	13	14	15	16	17	18
$\beta_1(d)$	11	8	13	14	15	12	17	18	16
$\beta_2(d)$	5	12	11	16	2	1	13	4	3

Figure 1: $2D$ combinatorial map example. Darts are represented by numbered black arrows. Two darts 1-sewn are drawn consecutively, and two darts 2-sewn are concurrently drawn and in reverse orientation, with a little grey segment between the two darts.

2. Recalls on Combinatorial Maps

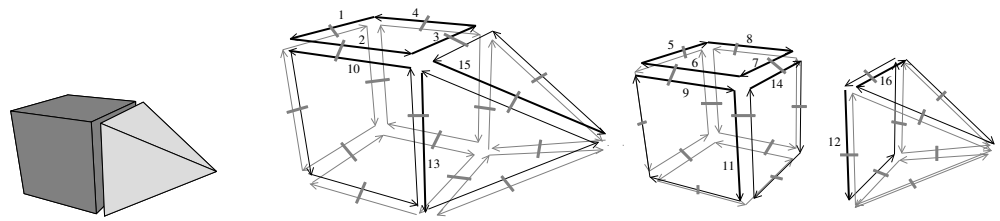
Definition 1 (Combinatorial map [12]). An nD combinatorial map (or n-map) is defined by a tuple $M = (D, \beta_1, \ldots, \beta_n)$ where

- D is a finite set of darts;
- β_1 is a *permutation* on D, *i.e.*, a one-to-one mapping from D to D;
- $\forall\, 2 \leq i \leq n$, β_i is an *involution* on D, *i.e.*, a one-to-one mapping from D to D such that $\beta_i = \beta_i^{-1}$;
- $\forall\, 1 \leq i \leq n-2$, $\forall\, i+2 \leq j \leq n$, $\beta_i \circ \beta_j$ is an *involution* on D.

A dart d is said to be i-sewn with another dart d' if $d = \beta_i(d')$. β_1 is a permutation which models edge successions when turning around $2D$ cells (i.e. faces) with respect to some given order. We note β_0 for β_1^{-1} so that β_0 models edge successions when turning around $2D$ cells with respect to the opposite order. Fig. 1 and 2 give examples of $2D$ and $3D$ combinatorial maps.

In some cases, it may be useful to allow some β_i to be partially defined, thus leading to open combinatorial maps. The basic idea is to add a new element ϵ to the set of darts, and to allow darts to be i-sewn with ϵ. By definition, $\forall 0 \leq i \leq n$, $\beta_i(\epsilon) = \epsilon$. Fig. 3 gives an example of open map (see [14] for precise definitions). In this paper, we always consider open combinatorial maps.

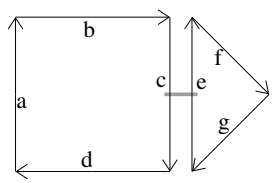
A map is connected if there exists a path of sewn darts between every pair of darts.



(a) (b)

d	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
$\beta_1(d)$	2	3	4	1	8	5	6	7	11	...	...	...	...	...	...	...
$\beta_2(d)$	...	10	...	...	...	9	14	...	6	2	...	...	...	7	...	...
$\beta_3(d)$	5	6	7	8	1	2	3	4	10	9	...	...	...	...	...	...

Figure 2: An example of a 3D combinatorial map. (a) A 3D object. (b) The corresponding 3D combinatorial map (external volume on the left; interior on the middle and the right). The graphical convention is the same as in 2D. β_3 is not drawn, but (partially) given in the array.



	a	b	c	d	e	f	g
β_1	b	c	d	a	f	g	e
β_2	ϵ	ϵ	e	ϵ	c	ϵ	ϵ

Figure 3: Open combinatorial map example. Darts a, b, d, f and g are not 2-sewn.

Definition 2 (Connected map). An n-map $M = (D, \beta_1, \ldots, \beta_n)$ is connected if $\forall d \in D, \forall d' \in D$, there exists a path $(d_1, \ldots, d_k)$ such that $d_1 = d$, $d_k = d'$ and $\forall i \in \{1, \ldots, k\}, \exists j_i \in \{0, \ldots, n\}, d_{i+1} = \beta_{j_i}(d_i)$.

Lienhardt has defined isomorphism between two combinatorial maps as follows.

Definition 3 (Map isomorphism [13]). Two n-maps $M = (D, \beta_1, \ldots, \beta_n)$ and $M' = (D', \beta'_1, \ldots, \beta'_n)$ are isomorphic if there exists a one-to-one mapping $f : D \to D'$, called *isomorphism function*, such that $\forall d \in D, \forall i \in \{1, \ldots, n\}$ $f(\beta_i(d)) = \beta'_i(f(d))$.

This definition has been extended to open maps in [6] by adding that $f(\epsilon) = \epsilon$, thus enforcing that, when a dart is i-sewn with ϵ, then the dart matched to it by f is i-sewn with ϵ.

3. Signatures of connected maps

In this section, we introduce two different canonical representations of maps, called signatures. We only consider connected maps; the extension of this work to non connected maps is discussed in Section 5.

3.1. Labelling of a connected map

Our signatures are based on map labellings, which associate a different label with every different dart. By definition, the label associated with ϵ is 0.

Definition 4 (Labelling). Given an n-map $M = (D, \beta_1, \ldots, \beta_n)$ a labelling of M is a bijective function $l : D \cup \{\epsilon\} \rightarrow \{0, \ldots, |D|\}$ such that $l(\epsilon) = 0$.

Example 1. $l = \{\epsilon : 0,\ a : 3,\ b : 1,\ c : 5\ , d : 7,\ e : 2,\ f : 6,\ g : 4\}$ *is a labelling of the map displayed in Fig. 3.*

One may compute a labelling of a map by performing a map traversal and labelling darts with respect to the order in which they are discovered. Different labellings may be computed, depending on (i) the initial dart from which the traversal is started, (ii) the strategy used to memorize the darts that have been discovered but that have not yet been treated (e.g., FIFO or LIFO), and (iii) the order in which the β_i functions are used to discover new darts.

We define below the labelling corresponding to a breadth first traversal of a map where β_i functions are used in increasing order.

Definition 5 (Breadth first labelling (BFL)). Given a connected n-map $M = (D, \beta_1, \ldots, \beta_n)$ and a dart $d \in D$ the breadth first labelling associated with (M, d) is the labelling returned by the function $BFL(M, d)$ described in algorithm 1.

Example 2. *The breadth first labellings associated with the map of Fig. 3 for darts a and e respectively are*

$$\begin{aligned} BFL(M, a) &= \{\epsilon : 0,\ a : 1,\ b : 3,\ c : 4,\ d : 2,\ e : 5,\ f : 7,\ g : 6\} \\ BFL(M, e) &= \{\epsilon : 0,\ a : 7,\ b : 5,\ c : 4,\ d : 6,\ e : 1,\ f : 3,\ g : 2\} \end{aligned}$$

Proposition 1. Algorithm 1 returns a labelling.

Algorithm 1: $BFL(M, d)$

Input: a connected n-map $M = (D, \beta_1, \ldots, \beta_n)$, and a dart $d \in D$
Output: a labelling $l : D \cup \{\epsilon\} \to \{0, \ldots, |D|\}$

```
1  for each d' ∈ D do l(d') ← −1;
2  l(ε) ← 0
3  let Q be an empty queue
4  add d at the end of Q
5  l(d) ← 1
6  nextLabel ← 2
7  while Q is not empty do
8      remove d' from the head of Q
9      for i in 0...n do
10         if l(β_i(d')) = −1 then
11             l(β_i(d')) ← nextLabel
12             nextLabel ← nextLabel + 1
13             add β_i(d') at the end of Q
14 return l
```

Proof.

- $l(\epsilon)$ is set to 0 in line 2.
- $\forall d, d' \in D, d \neq d' \Rightarrow l(d) \neq l(d')$. Indeed, each time a label is assigned to a dart (line 11), *nextLabel* is incremented (line 12).
- $\forall d \in D, 1 \leq l(d) \leq |D|$. Indeed, each dart enters exactly once in the queue because (i) the map is connected and (ii) a dart enters the queue only if it has not yet been labelled, and it is labelled just before entering it. □

Proposition 2. The time complexity of algorithm 1 is $\mathcal{O}(n \cdot |D|)$

Proof. The while loop (lines 7-13) is iterated $|D|$ times as (i) exactly one dart d is removed from the queue at each iteration; and (ii) each dart $d \in D$ enters the queue exactly once. The for loop (lines 9-13) is iterated $n + 1$ times. □

Note that the for loop (lines 9-13) iterates for every $i \in \{0, \ldots, n\}$, including 0. Indeed, as we consider open maps, some darts may not be 1-sewn. In this case, some darts may not be reachable from the initial dart d without using β_0. Let us consider for example the open map displayed in Fig. 3, and let us suppose that dart b has been removed. If BFL is started from the initial dart a, then no dart will be discovered if we only use β_1 and β_2 to discover new darts (as $\beta_1(a) = \beta_2(a) = \epsilon$ in this case). However, if we use β_0, β_1, and β_2, we can actually discover all darts.

Given a map M and a labelling l, one may describe M (i.e., its functions β_1 to β_n) by a sequence of labels of l. The idea is to first list the n labels of the n darts which are i-sewn with the dart labelled by 1 (*i.e.*, $l(\beta_1(1)), \ldots, l(\beta_n(1))$), and then by 2 (*i.e.*, $l(\beta_1(2)), \ldots, l(\beta_n(2))$), etc. More formally, we define the word associated with a map and a labelling as follows.

Definition 6 (Word). Given a connected n-map $M = (D, \beta_1, \ldots, \beta_n)$ and a labelling $l : D \cup \{\epsilon\} \to \{0, \ldots, |D|\}$ the word associated with (M, l) is the sequence

$$W(M, l) =< w_1, \ldots, w_{n \cdot |D|} >$$

such that $\forall i \in \{1, \ldots, n\}$, $\forall k \in \{1, \ldots, |D|\}$, $w_{i \cdot k} = l(\beta_i(d_k))$ where d_k is the dart labelled with k, i.e., $d_k = l^{-1}(k)$.

Notation. The word associated with the breadth first labelling of a map M, starting from a dart d, is denoted $W_{BFL}(M, d)$, i.e.,

$$W_{BFL}(M, d) = W(M, BFL(M, d))$$

Example 3. *The words associated with the map of Fig. 3 for the two labellings of example 2 respectively are*

$$\begin{aligned} W_{BFL}(M, a) &=< 3, 0, 1, 0, 4, 0, 2, 5, 7, 4, 5, 0, 6, 0 > \\ W_{BFL}(M, e) &=< 3, 4, 1, 0, 2, 0, 6, 1, 4, 0, 7, 0, 5, 0 > \end{aligned}$$

Algorithm for building $W_{BFL}(M, l)$. Given an n-map $M = (D, \beta_1, \ldots, \beta_n)$, the word $W_{BFL}(M, l)$ is computed by calling $BFL(M, l)$ and considering every dart of D in increasing label order and enumerating the labels of its n i-sewn darts. Note that we do not have to sort darts with respect to their labels as we can save this order during the run of BFL. Hence, the time complexity of the construction of the word $W_{BFL}(M, l)$ is $\mathcal{O}(n \cdot |D|)$.

The key point which allows us to use words for building signatures is that two maps are isomorphic if and only if they share a word for a breadth first labelling, as stated in theorem 1.

Theorem 1. *Two connected n-maps $M = (D, \beta_1, \ldots, \beta_n)$ and $M' = (D', \beta'_1, \ldots, \beta'_n)$ are isomorphic iff there exist $d \in D$ and $d' \in D'$ such that $W_{BFL}(M, d) = W_{BFL}(M', d')$*

Proof. $\Rightarrow$ Let us first consider two isomorphic n-maps $M = (D, \beta_1, \ldots, \beta_n)$ and $M' = (D', \beta'_1, \ldots, \beta'_n)$, and let us show that there exist two darts d and d' such that $W_{BFL}(M, d) = W_{BFL}(M', d')$. If M and M' are isomorphic then there exists $f : D \to D'$ such that $\forall d \in D, \forall i \in \{1, \ldots, n\}, f(\beta_i(d)) = \beta'_i(f(d))$ (Def. 3). Let d_1 be a dart of D, and let us note l (resp. l') the labellings returned by $BFL(M, d_1)$ (resp. $BFL(M', f(d_1))$). <u>Claim 1</u>: l and l' are such that $\forall d_i \in D, l(d_i) = l'(f(d_i))$. This is true for the initial dart d_1 as both d_1 and $f(d_1)$ are labelled with 1 at the beginning of each traversal. This is true for every other dart $d_i \in D$ as the traversals of M and M' performed by BFL are completely determined by the fact that (i) they consider the same FIFO strategy to select the next labelled dart which will be used to discover new darts and (ii) they use the β_i functions in the same order to discover new darts from a selected labelled dart. <u>Claim 2</u>: $\forall k \in \{1, \ldots, |D|\}, f(l^{-1}(k)) = l'^{-1}(k)$. This is a direct consequence of Claim 1. <u>Conclusion</u>: $\forall i \in \{1, \ldots, n\}, \forall k \in \{1, \ldots, |D|\}$, the $i.k^{th}$ element of $W_{BFL}(M, d_1)$ is equal to the $i.k^{th}$ element of $W'_{BFL}(M', f(d_1))$, i.e., $l(\beta_i(l^{-1}(k))) = l'(\beta'_i(l'{-}1(k)))$. Indeed,

$$\begin{aligned}
l(\beta_i(l^{-1}(k))) &= l'(f(\beta_i(l^{-1}(k)))) && \text{(because of Claim 1)}\\
&= l'(\beta'_i(f(l^{-1}(k)))) && \text{(because } f \text{ is an isomorphism function)}\\
&= l'(\beta'_i(l'^{-1}(k))) && \text{(because of Claim 2)}
\end{aligned}$$

$\Leftarrow$ Let us now consider two n-maps $M = (D, \beta_1, \ldots, \beta_n)$ and $M' = (D', \beta'_1, \ldots, \beta'_n)$ and two darts d and d' such that $W_{BFL}(M, d) = W_{BFL}(M', d')$, and let us show that M and M' are isomorphic. Let us note l (resp. l') the labellings returned by $BFL(M, d)$ (resp. $BFL(M', d')$), and let us define the function $f : D \to D'$ which matches darts with same labels, i.e., $\forall d_j \in D, f(d_j) = l'^{-1}(l(d_k))$. Note that this implies as well that $l(d_j) = l'(f(d_j))$. <u>Claim 3</u>: $\forall i \in \{1, \ldots, n\}, \forall k \in \{1, \ldots, |D|\}, l(\beta_i(l^{-1}(k))) = l'(\beta'_i(l'{-}1(k)))$. This comes from the fact that $W_{BFL}(M, d) = W_{BFL}(M', d')$ so that the $i.k^{th}$ element of $W_{BFL}(M, d_1)$ is equal to the $i.k^{th}$ element of $W'_{BFL}(M', f(d_1))$.

Conclusion: $\forall i \in \{1, \ldots, n\}, \forall d_j \in D$,

$$\begin{aligned}
f(\beta_i(d_j)) &= l'^{-1}(l(\beta_i(d_j))) && \text{(by definition of } f\text{)} \\
&= l'^{-1}(l'(\beta'_i(l'^{-1}(l(d_j))))) && \text{(because of Claim 3)} \\
&= \beta'_i(l'^{-1}(l(d_j))) && \text{(by simplification)} \\
&= \beta'_i(l'^{-1}(l'(f(d_j)))) && \text{(by definition of } f\text{)} \\
&= \beta'_i(f(d_j)) && \text{(by simplification)}
\end{aligned}$$

Hence, f is an isomorphism function and M and M' are isomorphic. □

3.2. Set Signature of a connected map

A map is characterized by the set of words associated with all possible breadth first labellings. This set defines a signature.

Definition 7 (Set Signature). Given an n-map $M = (D, \beta_1, \ldots, \beta_n)$, the Set Signature associated with M is $SS(M) = \{W_{BFL}(M, d) | d \in D\}$

Fig. 4 shows the Set Signature of the map of Fig. 3. Note that a Set Signature may contain less than $|D|$ words as there may exist different darts d and d' such that $W_{BFL}(M, d) = W_{BFL}(M, d')$ (in case of automorphisms).

Theorem 2. *$SS(M)$ is a signature, i.e., two connected maps M and M' are isomorphic if and only if $SS(M) = SS(M')$*

Proof. $\Rightarrow$ Let us consider two isomorphic maps $M = (D, \beta_1, \ldots, \beta_n)$ and $M' = (D', \beta'_1, \ldots, \beta'_n)$, and let us show that $SS(M) = SS(M')$. This is a direct consequence of theorem 1, which ensures that given an isomorphism function f between M and M' we have, for every dart $d \in D$, $W_{BFL}(M, d) = W_{BFL}(M', f(d))$. Hence, every word of $SS(M)$, computed from any dart of D, necessarily belongs to $SS(M')$ (and conversely).

$\Leftarrow$ Let us consider two maps $M = (D, \beta_1, \ldots, \beta_n)$ and $M' = (D', \beta'_1, \ldots, \beta'_n)$ such that $SS(M) = SS(M')$, and let us show that M and M' are isomorphic. Indeed, there exist two words $W \in SS(M)$ and $W' \in SS(M')$ such that $W = W'$, thus M and M' are isomorphic due to theorem 1. □

Note that a direct consequence of theorem 1 and theorem 2 is that for two non isomorphic maps M and M', $SS(M) \cap SS(M') = \emptyset$.

The Set Signature of a map M may be represented by a lexicographical tree which groups common prefixes of words.

$$\{ < 3,0,1,0,4,0,2,5,7,4,5,0,6,0 >,$$
$$< 3,0,1,0,4,5,2,0,7,3,5,0,6,0 >,$$
$$< 3,0,1,4,2,0,6,2,4,0,7,0,5,0 >,$$
$$< 3,4,1,0,2,0,6,1,4,0,7,0,5,0 >,$$
$$< 3,4,1,0,5,0,7,1,2,0,4,0,6,0 >,$$
$$< 3,0,1,0,2,4,6,3,4,0,7,0,5,0 >,$$
$$< 3,0,1,5,4,0,2,0,7,2,5,0,6,0 > \}$$

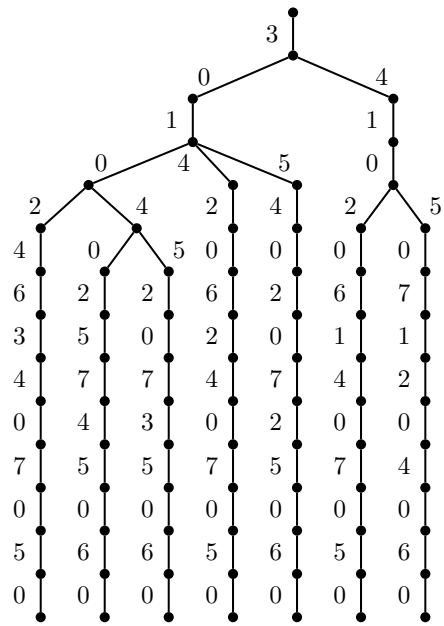


Figure 4: Set Signature of the map of Fig. 3. On the left, the set of words $SS(M)$, on the right, the tree $T_{SS}(M)$ (for each node u, $l(u)$ is displayed above u).

Definition 8 (Set Signature Tree of a map). Given an n-map $M = (D, \beta_1, \ldots, \beta_n)$, the tree associated with the Set Signature of M is the tree $T_{SS}(M)$ such that

- every node u except the root has a label $l(u)$ which is an integer ranging between 0 and $|D|$; we note $w(u)$ the word obtained by concatening all these labels along the path from the root to u;
- for every node u, all the children of u have different labels;
- there are $|SS(M)|$ leaves and for every leaf u, we have $w(u) \in SS(M)$.

For example, the Set Signature Tree of the map displayed in Fig. 3 is displayed in Fig. 4.

Property 1. The space complexity of the Set Signature Tree $T_{SS}(M)$ of a map M is $\mathcal{O}(n \cdot |D|^2)$.

Proof. The tree contains one leaf for each word in the signature, i.e., at most $|D|$ leaves, and the length of each path from the root to a leaf is $n \cdot |D|$.

Hence, the tree has $\mathcal{O}(n \cdot |D|^2)$ nodes. At each node u of the tree, we use a list to memorize all its children. The sum of the sizes of all lists is equal to the number of edges of the tree which is equal to the number of its nodes minus one. $\square$

Property 2. The time complexity for building the Set Signature Tree $T_{SS}(M)$ of a map M is $\mathcal{O}(n \cdot |D|^2)$.

Proof. The tree can be built in an incremental way: starting from the tree which only contains the root, we iteratively add $W_{BFL}(M, d)$ to it, for every dart $d \in D$. The time complexity for computing a word is $\mathcal{O}(n \cdot |D|)$. The time complexity for adding it to the current tree is also $\mathcal{O}(n \cdot |D|)$. Indeed, the length of a word is $n \cdot |D|$. For each label x of the word, we mainly have to decide if the current node of the tree has a child u such that $l(u) = x$. This is done in linear time with respect to the number of children of the current node as we use lists to memorize node children. A node has at most $|D| + 1$ children. However, the sum of the number of children of all nodes between the root and a leaf is bounded by the length of the path between the root and a leaf plus the number of leaves, i.e., $n \cdot |D| + |D|$. $\square$

Property 3. Given an n-map $M = (D, \beta_1, \ldots, \beta_n)$ and the Set Signature Tree $T_{SS}(M')$ of another map M', the complexity of deciding of the isomorphism between M and M' is $\mathcal{O}(n \cdot |D|)$.

Proof. To decide of the isomorphism, we build a breadth first labelling, starting from any dart $d \in D$, and decide if $W_{BFL}(M, d)$ corresponds to a path from the root to a leaf of the tree. This is done in linear time with respect to the length of the word (again, if a node of the tree may have up to $|D| + 1$ children, the sum of the number of children of all nodes between the root and a leaf is bounded by $n \cdot |D| + |D|$). $\square$

Note that we can check that the word corresponds to a path in the tree during the construction of the word so that we can stop the construction as soon as it does not match a branch in the tree. Note also that this algorithm is optimal. Indeed, to decide of the isomorphism between two maps we have to check if $f(\beta_i(d)) = \beta'_i(f(d))$ for every dart $d \in |D|$ and every dimension $i \in \{1, \ldots, n\}$. This cannot be done in less than $\mathcal{O}(n \cdot |D|)$.

3.3. Word Signature of a connected map

The lexicographical order is a strict total order on the words of a Set Signature, and we have shown that if two Set Signatures share one word, then they are equal. Hence, we may define a map signature by considering the smallest word of the Set Signature.

Definition 9 (Word Signature). Given a map M, the Word Signature of M is, $WS(M) = min(SS(M))$.

Example 4. *The Word Signature of the map displayed in Fig. 3 is*

$$WS(M) =< 3, 0, 1, 0, 2, 4, 6, 3, 4, 0, 7, 0, 5, 0 >$$

Property 4. The space complexity of a Word Signature is $\mathcal{O}(n \cdot |D|)$.

Algorithm for computing $WS(M)$. The Word Signature of a map M is built by calling $BFL(M, d)$ for each dart $d \in D$, and keeping the smallest word with respect to the lexicographical order. The time complexity for computing the Word Signature is $\mathcal{O}(n \cdot |D|^2)$. Note that this process may be improved (without changing the worst case complexity) by incrementally comparing the word in construction with the current smallest word and stopping the construction whenever it becomes greater.

Property 5. Given an n-map $M = (D, \beta_1, \ldots, \beta_n)$ and the Word Signature $WS(M')$ of another map M', we can decide of the isomorphism between M and M' in $\mathcal{O}(n \cdot |D|^2)$.

Proof. To decide of the isomorphism, we have to build breadth first labellings, starting from every different dart $d \in D$, until either $W_{BFL}(M, d) = WS(M')$ (M is isomorphic to M') or all darts have been tried (M is not isomorphic to M'). In the worst case, we have to build $|D|$ labellings so that the overall time complexity is $\mathcal{O}(n \cdot |D|^2)$. □

4. Signatures of databases of connected maps

Map signatures can be used to decide of the isomorphism of two maps. However, in many cases, we have to compare a map M not only with one other map but with a whole database of maps in order to search for maps isomorphic to M. To this aim, we define the signature of a database of maps. This signature actually merges all the signatures of the maps of the database into a single tree. We can either consider Set or Word Signatures.

In this section, we only consider connected maps; the extension of this work to non connected maps is discussed in Section 5.

4.1. Set Signature of a database of maps

We can model a database $B = \{M^1, \ldots, M^k\}$ of k n-maps by a list of k independent Set Signature Trees. Given a new n-map M, we can search for an isomorphic map in B in $\mathcal{O}(k \cdot n \cdot |D|)$ by iteratively searching for M in each of these trees. This complexity can be improved by merging the k trees into a single tree.

If all maps in the database have the same number of darts, then all branches (between the root and a leaf) have the same length in all trees. In this case, all branches also have the same length in the merged tree and we simply have to memorize, for every leaf u, the set of maps such that $w(u)$ belongs to the Set Signature of the map (the database may contain different maps which are isomorphic so that a same leaf may correspond to several maps).

However, if the number of darts is different from a map to another, then we have to merge trees whose branches have different lengths. In this case, it may happen that the word associated with a leaf in a tree is a prefix of the word associated with another leaf of another tree so that, when merging these two trees, a word may end on a node which is not a leaf. Hence, for each node u of the tree, we memorize the set $m(u)$ of maps such that $w(u)$ belongs to the Set Signature of the map.

Definition 10 (Set Signature Tree of a database of maps). Let $B = \{M^1, \ldots, M^k\}$ be a database of k n-maps and t be the maximum number of darts of the maps of B. The Set Signature Tree of B is the tree $T_{SS}(B)$ such that

- every node u except the root has a label $l(u)$ such that $l(u)$ is an integer ranging between 0 and t; we note $w(u)$ the word obtained by concatening every label $l(v)$ of every node v along the path from the root to u;
- for every node u, all the children of u have different labels;
- every node u except the root is associated with a set $m(u)$ defined by

$$m(u) = \{i \in \{1, \ldots, k\} \mid w(u) \in SS(M^i)\}$$

Figure 5 displays an example of Set Signature Tree for a database composed of 3 maps.

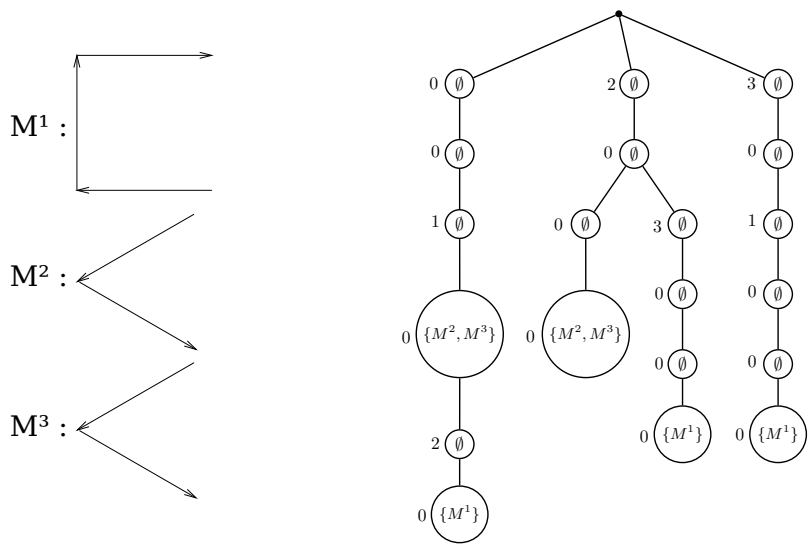


Figure 5: A database composed of 3 maps (left part) and its Set Signature Tree (right part). For each node u, the label $l(u)$ is displayed on the left of u whereas $m(u)$ is displayed in the node.

Property 6. The space complexity of the Set Signature Tree $T_{SS}(B)$ of a database B is $\mathcal{O}(k \cdot n \cdot t^2)$.

Proof. The tree contains at most $k \cdot t$ leaves (one for each different word in a Set Signature of a map), and the length of each path from the root to a leaf is bounded by $n \cdot t$. Hence, the number of nodes of the tree is bounded by $k \cdot n \cdot t^2$. At each node u of the tree, we use a list to memorize all its children. □

Property 7. The time complexity for building the Set Signature Tree $T_{SS}(B)$ of a database B is $\mathcal{O}(k \cdot n \cdot t^3)$.

Proof. The tree can be built in an incremental way by iteratively adding each word of each Set Signature. There are $\mathcal{O}(k \cdot t)$ words to be added to the tree and the length of a word is in $\mathcal{O}(n \cdot t)$. For each label x of the word, we mainly have to decide if the current node of the tree has a child u such that $l(u) = x$. As we use lists to memorize node children, this is done in linear time with respect to the number of children of the current node, and a node may have at most $t + 1$ children. □

This complexity is based on a worst case: if a node may have at most $t+1$ children in the Set Signature Tree, we shall experimentally show in Section 4.3 that, on randomly generated maps, a node has only very few children.

Property 8. Given an n-map $M = (D, \beta_1, \ldots, \beta_n)$ and the Set Signature Tree $T_{SS}(B)$ of a database B, the complexity of searching for all maps of B which are isomorphic to M is $\mathcal{O}(n \cdot t^2)$, where t is the maximum number of darts of the maps of B.

Proof. To search for all maps isomorphic to M, we build a breadth first labelling, starting from any dart $d \in D$. This is done in $\mathcal{O}(n \cdot t)$. Then, we search for a node u of the tree such that $w(u) = W_{BFL}(M, d)$: if this node exists, then $m(u)$ gives the set of maps of B which are isomorphic to M; otherwise, no map of B is isomorphic to M. To search for a node u such that $w(u) = W_{BFL}(M, d)$, we have to search for each symbol w_i of $W_{BFL}(M, d)$ if the corresponding node in the tree has a child labelled by w_i. As each node has at most $t + 1$ children, and $W_{BFL}(M, d)$ has at most $n \cdot t$ symbols, the whole step is done in $\mathcal{O}(n \cdot t^2)$. □

4.2. Word Signature of a database of maps

We can also model a database of n-maps by a tree which contains the Word Signatures of the maps.

Definition 11 (Word Signature Tree of a database of maps). Let $B = \{M^1, \ldots, M^k\}$ be a database of k n-maps and let t be the maximum number of darts of the maps of B. The Word Signature Tree of B is the tree $T_{WS}(B)$ such that

- every node u except the root has a label $l(u)$ such that $l(u)$ is an integer ranging between 0 and t; we note $w(u)$ the word obtained by concatening every label $l(v)$ of every node v along the path from the root to u;
- for every node u, all the children of u have different labels;
- every node u except the root is associated with a set $m(u)$ defined by

$$m(u) = \{i \in \{1, \ldots, k\} \mid w(u) = WS(M^i)\}$$

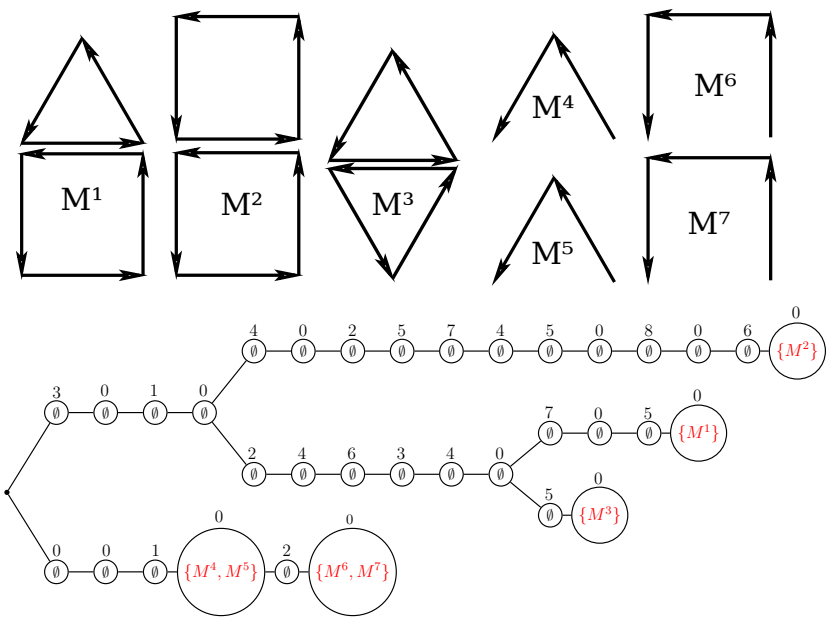


Figure 6: Example of a Word Signature Tree representing the maps M^1,M^2,M^3,M^4,M^5,M^6, and M^7.

Figure 6 shows an example of a database of maps represented by a Word Signature Tree.

Property 9. The space complexity of the Word Signature Tree $T_{WS}(B)$ of a database B is $\mathcal{O}(k \cdot n \cdot t)$.

Proof. The tree contains at most k leaves (one for each different map of the database), and the length of each path from the root to a leaf is bounded by $n \cdot t$. Hence, the number of nodes of the tree is bounded by $k \cdot n \cdot t$. At each node u of the tree, we use a list to memorize all its children. □

Property 10. The time complexity for building the Word Signature Tree $T_{SS}(B)$ of a database B is $\mathcal{O}(k \cdot n \cdot t^2)$.

Proof. The tree can be built in an incremental way by iteratively adding each word of each Set Signature. There are $\mathcal{O}(k)$ words to be added to the tree and each word is computed in $\mathcal{O}(n \cdot t^2)$. The length of a word if in $\mathcal{O}(n \cdot t)$. For each label x of the word, we mainly have to decide if the current node of

the tree has a child u such that $l(u) = x$. As we use lists to memorize node children, this is done in linear time with respect to the number of children of the current node, and a node may have at most $t + 1$ children. □

Property 11. Given an n-map $M = (D, \beta_1, \ldots, \beta_n)$ and the Word Signature Tree $T_{WS}(B)$ of a database B, the complexity of searching for all maps of B which are isomorphic to M is $\mathcal{O}(n \cdot t^2)$.

Proof. To search for all maps isomorphic to M, we build the Word Signature $WS(M)$. This is done in $\mathcal{O}(n \cdot t^2)$. Then, we search for a node u of the tree such that $w(u) = WS(M)$ just like in the Set Signature Tree. This is done in $\mathcal{O}(n \cdot t^2)$. □

4.3. Number of children of the nodes of Set and Word Signature Trees

The time complexity of the construction of Set and Word Signature Trees as well as the time complexity of searching for a map in a tree depend on the number of children of the nodes of the trees. Indeed, the different children of a node are memorized in a list so that the search for a child with a given label is done in linear time with respect to the number of children. A simple upper bound on the number of children of a node in a signature tree is given by the number of different labels (i.e., $t+1$ if the largest map of the database has t darts) as all children must have a different label.

In this section, we first show that it is possible to build a database of maps such that the number of children of nodes are of the same order as this upper bound. Then, we experimentally show that the number of children is much lower for randomly generated databases of maps.

Worst case study. For a database of k n-maps of t darts, the Set Signature Tree has $\mathcal{O}(t \cdot k)$ leaves and the Word Signature Tree has $\mathcal{O}(k)$ leaves; in both cases, the length of branches from the root to leaves is in $\mathcal{O}(n \cdot t)$.

Let us consider a node u labelled by dart $l(u)$ and let i be the depth of u in the tree (i.e., the length of the path between the root and u). If the number of children of u is bounded by the number of different labels (i.e., $t + 1$), it is also bounded by the number of darts that have been discovered when dart $l(u)$ has been removed from the queue Q (line 8 of algorithm 1). This number is equal to $i \cdot n$. Indeed, every dart $l(v)$ associated with a node v between the root and u may discover n new darts. Hence, the number of children of u is bounded by $min(t, i \cdot n)$.

This bound may be slighly improved by taking into account the fact that, for every dimension $j \in \{1, \ldots, n\}$, β_j is a permutation so that every dart appears at most n times in a word. More precisely, if node u is at depth i, then, for every dimension, the number of different darts that have been used in $w(u)$ is i/n, so that the number of children of u should be decreased by i/n. Hence, a more precise bound on the number of children of a node at depth i is $B(i) = min(t, i \cdot n) - i/n$.

We now show that it is possible to build a database of maps such that the number of children of a node at depth i in the signature tree reaches this bound $B(i)$. To do so, the number of visited darts must increase faster than the number of used darts. Thus, as much as possible, a dart must be linked with only one visited dart and with new darts in all the other dimensions. Then, for a given depth, we can create as many branches as there are free darts for a given dimension that are already discovered but not yet used. Figure 7 shows an example of a database where the number of children of a node is equal to 6. In this example, we can 1-sew dart 18 either to ϵ or to darts 17, 19, 21, or 23 or to a new dart. Each case will create a new branch at depth $(18 * 2)$. But we cannot 1-sew dart 18 to darts 3, 5, 9, or 11 because it will change the labelling and create a new branch before depth $(18 * 2)$. Then, for each branch we can generate height new branches, we can 2-sew dart 18 to ϵ or darts 19, 20, 21, 22, 23, or 24 or a new dart. In this way it is possible to create as many branches as we want which will have large numbers of children.

Number of children in randomly generated maps. If it is possible to build a pathological database for which the number of children reaches the upper bound, it is worth studying the evolution of this number in practice on randomly generated databases. Hence, Table 1 displays the maximum and the average number of children of a node depending on (1) the depth i of the node in the tree; (2) the dimension n of the maps; and (3) the number k of different maps in the database. For all maps, the number of darts t has been fixed to 500.

This table shows us that the number of children of a node is much smaller than the worst case bound: in all the generated databases (which have up to 10000 different maps of 500 darts), the maximum number of children of a node is 4 with the Word Signature and 5 with the Set Signature. Note that the observed degree depends on the depth of the node in the tree: when i is greater than $100n$, nodes nearly always have only one child. The observed

n	k	Word Signature Tree						Set Signature Tree					
		$i \in [1..10n]$		$i \in]10n..100n]$		$i \in]100n..500n]$		$i \in [1..10n]$		$i \in]10n..100n]$		$i \in]100n..500n]$	
		max	avg	max	avg	max	avg	max	avg	max	avg	max	avg
2	10	3	1.7	1	1	1	1	4	3.3	3	1.11	1	1
	100	4	2.4	2	1.04	1	1	5	3.45	3	1.22	1	1
	1000	4	2.55	3	1.12	1	1	5	3.85	4	1.35	-	-
	10000	4	2.65	3	1.25	1	1	5	3.95	-	-	-	-
3	10	3	1.57	1	1	1	1	3	2.9	2	1.03	1	1
	100	3	2.2	2	1	1	1	4	3.03	3	1.13	1	1
	1000	3	2.6	2	1.06	1	1	4	3.1	3	1.25	-	-
	10000	3	2.63	3	1.15	1	1	4	3.57	-	-	-	-
4	10	3	1.53	1	1	1	1	3	2.78	3	1.13	1	1
	100	3	1.95	2	1.02	1	1	3	2.78	3	1.28	1	1
	1000	3	2.38	3	1.12	1	1	3	2.78	3	1.52	-	-
	10000	3	2.5	3	1.29	1	1	3	2.8	-	-	-	-
8	10	3	1.28	1	1	1	1	3	2.15	2	1.22	1	1
	100	3	1.55	2	1	1	1	3	2.33	3	1.6	2	1.02
	1000	3	2.13	3	1.08	1	1	3	1.95	2	1.98	-	-
	10000	3	2.29	3	1.3	1	1	-	-	-	-	-	-

Table 1: Maximum and average number of children of a node depending on the depth i in the tree, the dimension n of the maps and the number of different maps k in the database. Maps are randomly generated. For Set Signature Tree, - corresponds to databases that cannot be represented in 16 GB RAM.

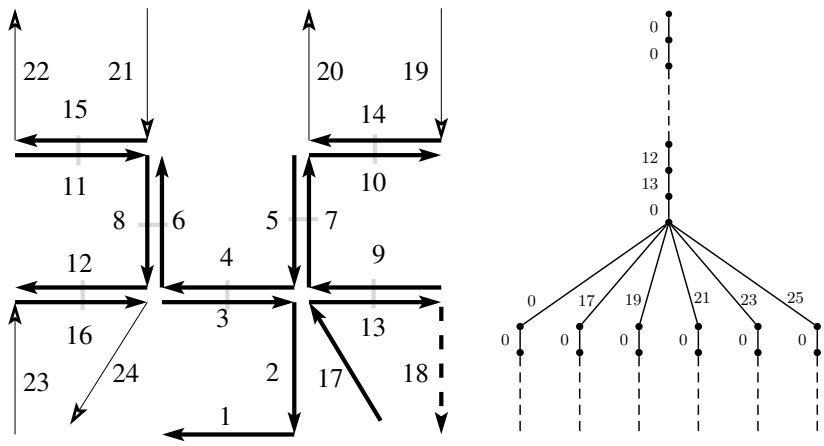


Figure 7: Example of worst case for the number of nodes of a Word Signature Tree. On the left: a 2-map M. On the right : part of the Word Signature Tree which contains the six maps obtained from M by 1-sewing dart 18 either to ϵ or to darts 17, 19, 21, or 23 or to a new dart.

degree also depends on the number of maps in the database and on the dimension of the maps: when k increases and/or n decreases, the number of children slightly increases. Finally, the observed degree is slightly higher in Set Signature Trees than in Word Signature Trees. This reflects the fact that the Set Signature Tree may contain t times more branches than the Word Signature Tree to represent the same database.

5. Signatures of non-connected maps

Signatures introduced in the previous sections are defined for connected maps. Indeed, if a map is not connected, then the labelling process described in algorithm 1 cannot label all darts as some darts cannot be reached from the initial dart.

However, a non connected n-map may be decomposed in a set of disjoint connected maps in $\mathcal{O}(n \cdot |D|)$ by performing successive map traversals until all darts have been discovered. The signature of a non-connected map is built from the signatures of its different connected components in a very similar way as we build signatures for databases of maps.

More precisely, given a non connected map M such that M is composed of k disjoint connected maps $M^1, \ldots, M^k$. We define the database $B_M =$

$\{M^1, \ldots, M^k\}$. The Set Signature of M is defined by the Set Signature Tree $T_{SS}(B_M)$ and its Word Signature is defined by the Word Signature Tree $T_{WS}(B_M)$.

Property 12. Given a new n-map $M' = (D', \beta'_1, \ldots, \beta'_n)$ and the Set Signature Tree $T_{SS}(B_M)$, we can decide of the isomorphism between M and M' in $\mathcal{O}(n \cdot |D'|^2)$.

Proof. Indeed, to decide of isomorphism, we can proceed as follows:

1. We decompose M' into a set of disjoint connected maps. Let k' be the number of connected components. If $k' \neq k$ then the two maps are trivially not isomorphic.
   This step is done in $\mathcal{O}(n \cdot |D'|)$.
2. If $k' = k$, let $M'^1, \ldots, M'^k$ be the different connected components of M'. For every connected component M'^i, we build a breadth first labelling, starting from any dart d of M'^i, and decide if $W_{BFL}(M'^i, d)$ corresponds to a path from the root to a node u of the tree. If this is not the case, then the two maps are not isomorphic. Otherwise, let us note $m^i = m(u)$.
   This step is done in $\mathcal{O}(n \cdot |D'^i|)$ for every connected component i (see Property 8).
3. We have to check, for every connected component M'^i, that the number of connected components of M' which are isomorphic to M'^i is equal to the number of connected components of M which are isomorphic to M'^i, i.e., that $|m^i| = |\{j \in \{1, \ldots, k\}, |m^i| = |m^j|\}|$.
   This step may be done in $\mathcal{O}(k)$ by using counters.

Property 13. Given a new n-map $M' = (D', \beta'_1, \ldots, \beta'_n)$ and the Word Signature Tree $T_{WS}(B_M)$, we can decide of the isomorphism between M and M' in $\mathcal{O}(n \cdot |D'|^2)$.

Proof. To decide of isomorphism with a map modelled by a Word Signature, we proceed like with a Set Signature. The only difference is in step 2: we must first compute the Word Signature of each connected component M'^i before looking for a node u in the tree such that $w(u)$ is equal to the Word Signature. The computation of the Word Signature of a connected component M'^i is done in $\mathcal{O}(n \cdot |D'^i|)$ so that the complexity of step 2 is not changed.

Finally, we can also define Set and Word Signature Trees of databases of non-connected maps in a rather straightforward way: we mainly have to store, for each different non-connected map, the set of its connected components.

6. Experimental evaluation

In this section we report some experiments which demonstrate the interest of using signatures. All the reported experiments have been performed on a 2.26 GHz Intel Xeon E5520 with 16GB RAM.

Using map signatures to represent images. Maps may be extracted from segmented images by using the linear algorithm described in [5]. We obtain the same map whatever we submit the image to a rotation or a scale-up. Hence, map signatures may be used to identify images even if they have been rotated or scaled-up. Table 2 displays 5 images and the number of darts and faces of the maps extracted from these images. It compares the CPU time needed to compute Set and Word Signatures of these maps. It shows us that signatures are very quickly computed, even for rather large maps that have more than 6000 darts.

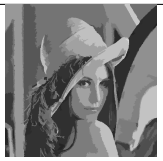
Image					
Darts	3410	6060	1728	4224	1590
Faces	590	1044	295	716	275
$SS(M)$	0.83	2.21	0.26	1.14	0.26
$WS(M)$	0.26	0.53	0.15	0.32	0.16

Table 2: From images to signatures: the first line displays images, the next two lines give the number of darts and faces in the corresponding maps; the last two lines give the CPU time in seconds for computing the Set and Word Signatures of these maps.

Scalability of signature constructions. To compare scale-up properties of Set and Word Signatures, we have performed experiments on randomly generated maps with exponentially growing sizes (with 1000, 2000, 4000 and 8000 darts). Table 3 first compares time complexities for constructing Set and

Word Signatures. To build the Set Signature, one has to perform a complete breadth first traversal for each dart so that the total number of visited darts is always equal to $|D|^2$ and the time complexity does not depend on the initial dart chosen to start the traversal. To build a Word Signature, one also has to perform a breadth first traversal for each dart but each traversal may be stopped as soon as the corresponding word is greater than the smallest word computed so far. Hence, if the worst case complexity is quadratic, Table 3 shows that the CPU time needed to compute a Word Signature is sub quadratic in practice. Indeed, the average number of darts visited for each traversal varies from 19 or so for the map with 1000 darts to 27 or so for the map with 8000 darts. Note that, if the number of visited darts actually depends on the order in which initial darts are chosen, standard deviations are rather low.

	Set Signature		Word Signature		
$\|D\|$	Time avg	$\frac{\|Visited\ darts\|}{\|D\|}$	Time avg	$\frac{\|Visited\ darts\|}{\|D\|}$ avg	(sdv)
1000	0.054	1000	0.047	19.48	(3.24)
2000	0.228	2000	0.084	19.27	(3.71)
4000	1.056	4000	0.262	23.78	(5.31)
8000	4.088	8000	0.352	26.91	(4.88)

Table 3: Comparison of time complexities for computing Set and Word Signatures of a map. Each line successively gives the number of darts $|D|$ of the map and, for each signature, the CPU time (in seconds) and the ratio between the number of visited darts and $|D|$. For the Word Signature, we give average results (and standard deviations) obtained with different orders over the set of darts.

Scale-up properties of signatures for deciding of isomorphism. We now compare Set and Word Signatures for deciding if a new map M' is isomorphic to a map M described by its signature.

When using the Set Signature $SS(M)$, the worst case complexity is $\mathcal{O}(n \cdot |D|)$. Table 4 shows that, when M' and M are isomorphic (when the percentage of different darts is 0%), the algorithm visits each dart exactly once. However, when M and M' are not isomorphic, the breadth first traversal of M' may be stopped as soon as no branch of the lexicographical tree matches the word under construction. Table 4 shows that the more different M and M', the smaller the number of visited darts.

When using the Word Signature $WS(M)$, the worst case complexity is $\mathcal{O}(n \cdot |D|^2)$ as one has to perform a breadth first traversal starting from every dart of M'. However, one may stop each breadth first traversal as soon as the word under construction is different from the signature. Hence, Table 4 shows that the more different M and M', the smaller the number of visited darts. In practice, each dart is visited from 2 to 4 times. Interestingly, this ratio does not significantly vary when increasing the size of the map.

Table 4 also compares these results with the isomorphism algorithm of [6]. This algorithm decides of isomorphism of 2 maps in $\mathcal{O}(n \cdot |D|^2)$. It builds a matching by performing a traversal of M and M' starting from a dart d of M and every dart d' of M'. The Word Signature and the isomorphism algorithm have the same time complexities and they exhibit very similar CPU times. The main interest of the Word Signature lies in the fact that the signatures of the maps in a database can be merged into a Word Signature Tree, thus allowing us to very efficiently search for a map in a database (whatever the size of the database is).

Scale-up properties for searching for a map in a database of maps. To compare scale-up properties of Set and Word Signature Trees for representing a database of maps, we have randomly generated seven databases of 2-maps. Each database contains 100 non-isomorphic 2-maps and each 2-map respectively has 100, 200, 500, 1000, 2000, 4000 and 8000 darts. Table 5 shows us the interest of merging the signatures of the different maps into a single tree (column *Tree*) compared to keeping every signature independently (column *Indpt*). Merging the different signatures into a single tree allows us to save memory, but the gain is small (less than 10%). The main interest lies in the speed-up of the process of searching for a map into the database: CPU times are from 50 to 100 times as small when merging all signatures into a single tree. Indeed, searching for a map in the merged tree depends on the size of the map but not on the number of maps in the database. As a comparison, when storing each signature independently, the searched map must be compared with every signature of the database independently so the more maps are in the database, the longer the search lasts.

Table 5 also shows us that searching for a map in a Set Signature Tree is much faster than searching for a map in a Word Signature Tree. However, the space complexity of the Set Signature Tree is also an order higher: with our computer (16GB RAM), we cannot store the Set Signature Tree of a database of 100 maps of 2000 darts, while there is no problem with a Word

Signature Tree. Hence, Set Signature Trees should be used only for rather small databases, when CPU time is a critical issue. Otherwise, one has better use Word Signature Trees. Note that the time spent to search for a map in a Set Signature Tree is much shorter when the map is not in the database. Indeed, we check that the word associated with the searched map corresponds to a path in the tree during the construction of the word so that we stop the construction as soon as it does not match a branch in the tree. When using the Word Signature Tree, times are not significantly different whenever the searched map belongs to the database or not. Indeed, in both cases, we have to search for the smallest word (by calling BFL for every dart of the searched map) before searching for this word in the tree.

Finally, table 6 allows us to study scale-up properties of the Word Signature Tree when increasing the number k of maps in the database (we do not report results with the Set Signature Tree as memory is rather quickly exceeded). It shows us that the size of the tree increases linearly with the number of maps, and that the time to search for a map in the database is rather constant and does not depend on the number of maps in the database. Hence, Word Signature Trees allow us to find a map of 1,000 darts into a database of 100,000 maps of 1,000 darts in 0.003 second.

7. Conclusion

In this paper, we have introduced a Set and a Word Signature of combinatorial maps. The space complexity of the Set Signature is quadratic and it allows us to decide of isomorphism in linear time in the worst case. The space complexity of the Word Signature is linear and it allows us to decide of isomorphism in quadratic time in the worst case. Experimental results on randomly generated maps have shown us that CPU times actually grow sub-linearly for the Set Signature and linearly for the Word Signature.

We have also shown that these signatures can be merged into trees in order to efficiently search for a map into a database of maps. Experimental results on randomly generated maps have shown us that merging Set Signatures allows us to find a map quicker, but it also requires more memory so that it cannot be used for large databases.

Further work mainly concern the use of these signatures to search for frequent submaps in a database of maps. Our goal is to use these frequent submaps to characterize classes of maps modeling 2D or 3D images.

[1] J.-P. Braquelaire and L. Brun. Image segmentation with topological maps and inter-pixel representation. 9(1):62–79, march 1998.

[2] E. Brisson. Representing geometric structures in d dimensions: topology and order. In *Proc.* 5th *Annual ACM Symposium on Computational Geometry*, pages 218–227, Saarbrücken, Germany, 1989.

[3] R. Cori. Un code pour les graphes planaires et ses applications. In *Astérisque*, volume 27. Soc. Math. de France, Paris, France, 1975.

[4] G. Damiand. Topological model for 3d image representation: Definition and incremental extraction algorithm. *Computer Vision and Image Understanding*, 109(3):260–289, March 2008.

[5] G. Damiand, Y. Bertrand, and C. Fiorio. Topological model for two-dimensional image representation: definition and optimal extraction algorithm. *Computer Vision and Image Understanding*, 93(2):111–154, February 2004.

[6] G. Damiand, C. De La Higuera, J.-C. Janodet, E. Samuel, and C. Solnon. Polynomial Algorithm for Submap Isomorphism: Application to searching patterns in images. In *Graph-based Representation for Pattern Recognition (GbR)*, LNCS 5534, pages 102–112. Springer, May 2009.

[7] J. Edmonds. A combinatorial representation for polyhedral surfaces. *Notices of the American Mathematical Society*, 7, 1960.

[8] T. Horvath, J. Ramon, and S. Wrobel. Frequent subgraph mining in outerplanar graphs. In *KDD 2006*, pages 197–206, 2006.

[9] A. Jacques. Constellations et graphes topologiques. In *Combinatorial Theory and Applications*, volume 2, pages 657–673, 1970.

[10] X. Jiang and H. Bunke. Optimal quadratic-time isomorphism of ordered graphs. *Pattern Recognition*, 32(7):1273–1283, 1999.

[11] P. Lienhardt. Subdivision of n-dimensional spaces and n-dimensional generalized maps. In *Proc.* 5th *Annual ACM Symposium on Computational Geometry*, pages 228–236, Saarbrücken, Germany, 1989.

[12] P. Lienhardt. Topological models for boundary representation: a comparison with n-dimensional generalized maps. *Computer-Aided Design*, 23(1):59–82, 1991.

[13] P. Lienhardt. N-dimensional generalized combinatorial maps and cellular quasi-manifolds. *International Journal of Computational Geometry and Applications*, 4(3):275–324, 1994.

[14] M. Poudret, A. Arnould, Y. Bertrand, and P. Lienhardt. Cartes combinatoires ouvertes. Research Notes 2007-1, Laboratoire SIC E.A. 4103, F-86962 Futuroscope Cedex - France, October 2007.

[15] W.T. Tutte. A census of planar maps. *Canad. J. Math.*, 15:249–271, 1963.

		Set Signature			Word Signature			Direct Isomorphism		
$\|D\|$		Time	$\frac{\|Visited\ darts\|}{\|D\|}$		Time	$\frac{\|Visited\ darts\|}{\|D\|}$		Time	$\frac{\|Visited\ darts\|}{\|D\|}$	
		avg	avg	(sdv)	avg	avg	(sdv)	avg	avg	(sdv)
1000	0%	0.000099	1.000	(0.000)	0.035	2.13	(0.64)	0.030	2.09	(0.72)
	1%	0.000091	0.298	(0.214)	0.060	3.71	(1.48)	0.058	3.58	(1.51)
	10%	0.000086	0.026	(0.021)	0.059	3.41	(1.34)	0.058	3.42	(1.53)
	50%	0.000072	0.015	(0.006)	0.056	1.88	(1.19)	0.056	1.64	(1.08)
	99%	0.000068	0.011	(0.004)	0.050	1.59	(0.90)	0.055	1.55	(0.93)
2000	0%	0.000215	1.000	(0.000)	0.084	2.59	(1.47)	0.076	2.55	(1.62)
	1%	0.000161	0.069	(0.081)	0.095	3.08	(1.79)	0.102	3.22	(1.64)
	10%	0.000130	0.019	(0.032)	0.076	2.92	(1.76)	0.084	3.01	(1.72)
	50%	0.000098	0.006	(0.005)	0.073	1.77	(1.40)	0.067	1.56	(1.43)
	99%	0.000097	0.006	(0.003)	0.069	1.38	(0.83)	0.066	1.38	(0.93)
4000	0%	0.000341	1.000	(0.000)	0.262	2.46	(1.30)	0.212	2.31	(1.41)
	1%	0.000292	0.015	(0.037)	0.434	3.09	(1.89)	0.451	3.45	(1.58)
	10%	0.000222	0.005	(0.005)	0.329	2.57	(1.81)	0.331	3.02	(1.43)
	50%	0.000178	0.005	(0.006)	0.286	2.03	(1.41)	0.305	2.54	(1.20)
	99%	0.000164	0.005	(0.003)	0.273	1.43	(0.85)	0.265	1.28	(1.09)
8000	0%	0.000697	1.000	(0.000)	0.352	2.23	(1.04)	0.450	2.62	(0.98)
	1%	0.000556	0.032	(0.178)	1.451	3.11	(1.86)	1.397	2.99	(1.62)
	10%	0.000439	0.003	(0.009)	1.343	3.05	(1.81)	1.263	2.89	(1.45)
	50%	0.000296	0.002	(0.003)	1.042	2.44	(1.25)	1.101	2.46	1.31
	99%	0.000353	0.003	(0.003)	0.993	1.53	(1.02)	0.910	1.48	(1.14)

Table 4: Comparison of scale-up properties of Set and Word Signatures for deciding if a new map M' is isomorphic to a map M given the signature of M. M and M' have the same number of darts, but M' is obtained from M by removing and then adding a given percentage of darts. When this percentage is 0%, M and M' are isomorphic. Each line successively gives: the number of darts of M, the percentage of different darts between M and M', and, for each signature, the time and the ratio between the number of visited darts and the number of darts of M. We give average results (and standard deviations) obtained when changing the initial dart of M'. The last 3 columns give results when using the isomorphism algorithm of [6].

	Number of nodes				Time			
	Set Signature		Word Signature		Set Signature		Word Signature	
$\|D\|$	Indpt	Tree	Indpt	Tree	Indpt	Tree	Indpt	Tree
100	1,810,207	1,666,663	20,000	16,832	0.000207	0.000004	0.032033	0.000097
200	7,558,050	7,221,234	40,000	36,551	0.000244	0.000007	0.020617	0.000225
500	48,675,443	47,735,334	100,000	95,929	0.000312	0.000015	0.751917	0.000956
1,000	197,031,371	195,086,897	200,000	195,771	0.000451	0.000030	0.303131	0.003803
2,000	–	–	400,000	394,745	–	–	1.337401	0.014879
4,000	–	–	800,000	794,896	–	–	4.608052	0.053965
8,000	–	–	1,600,000	1,594,375	–	–	15.93316	0.216157

Table 5: Comparison between word and Set Signature Trees to search for a map in a database. For each signature we compare performences obtained when the different map signatures are independent (column *Indpt* and when the signatures have been merged into a single tree (column *Tree*). Each database contains 100 different maps randomly generated. Each line displays: the number of darts of each map of the database and, for each signature, the memory used (number of nodes in the tree) and the time in seconds to search for a map in the database (average on 200 searches, where 100 maps are in the database and 100 aren't) .

Maps	Number of nodes	Time
100	195,771	0.003046
1,000	1,933,461	0.003053
10,000	19,311,582	0.003053
100,000	192,171,529	0.003046

Table 6: Scale-up properties of the Word Signature Tree for searching for a map in a database when increasing the number of maps in the database. Each line displays: the number of maps in the database (each map has 1000 darts), the memory used (number of nodes in the tree) and the time in seconds to search for a map in the database (average on 200 searches).