Polynomial Algorithms for Subisomorphism of nD Open Combinatorial Maps

Guillaume Damiand, Christine Solnon, Colin De La Higuera, Jean-Christophe Janodet, Émilie Samuel

To cite this version:

Guillaume Damiand, Christine Solnon, Colin De La Higuera, Jean-Christophe Janodet, Émilie Samuel. Polynomial Algorithms for Subisomorphism of nD Open Combinatorial Maps. Computer Vision and Image Understanding, Elsevier, 2011, 115 (7), pp.996-1010. <10.1016/j.cviu.2010.12.013>. <hal-00597483>

HAL Id: hal-00597483

https://hal.archives-ouvertes.fr/hal-00597483

Submitted on 1 Jun 2011

Polynomial Algorithms for Subisomorphism of nD Open Combinatorial Maps

Guillaume Damiand[a], Christine Solnon[a], Colin de la Higuera[b], Jean-Christophe Janodet[c], Émilie Samuel[c]

[a] *Université de Lyon, CNRS*
Université Lyon 1, LIRIS, UMR5205, F-69622, France
[b] *Université de Nantes, CNRS, LINA, UMR6241, F-44000, France*
[c] *Université de Lyon, CNRS, Université de Saint-Etienne - Jean Monnet, Laboratoire Hubert Curien, UMR5516, F-42023, France*

Abstract

Combinatorial maps describe the subdivision of objects in cells, and incidence and adjacency relations between cells, and they are widely used to model 2D and 3D images. However, there is no algorithm for comparing combinatorial maps, which is an important issue for image processing and analysis. In this paper, we address two basic comparison problems, *i.e.*, *map isomorphism*, which involves deciding if two maps are equivalent, and *submap isomorphism*, which involves deciding if a copy of a pattern map may be found in a target map. We formally define these two problems for nD open combinatorial maps, we give polynomial time algorithms for solving them, and we illustrate their interest and feasibility for searching patterns in 2D and 3D images, as any child would aim to do when he searches Wally in Martin Handford's books.

Keywords: open combinatorial maps, isomorphism and subisomorphism, pattern detection, 2D and 3D images

1. Introduction

Graphs are used in many computer graphic applications to describe images (see, for example, [CFSV07] for a review of graph-based methods for pattern recognition and computer vision). In particular, Region Adjacency Graphs (RAGs) [Ros74] model images by means of vertices —corresponding to maximal homogeneous sets of connected pixels— and edges —corresponding to adjacency relationships. RAGs are used in many image processing applications like, for example, segmentation, object extraction or comparison, and image analysis [SC84, JB93, Saa94, GMBM95, LMV01].

However, RAGs cannot model some important information contained in images. In particular, they cannot model the order in which neighbor regions are encountered when turning around some given region, as the edges incident to a vertex are not ordered. Also, RAGs cannot represent multi-adjacency. Hence, two different images may be represented by the same RAG [Kov89].

To get round this default, the RAG model has been extended. For example, [KM95] defines the dual graph structure, which is a pair of multi-graphs that represent multi-adjacency relations, and [JB98] defines ordered graphs, such that edges incident to a vertex are uniquely ordered.

Several works have proposed some solutions in 2D [Dom92, Fio96, Bru96, DBF04] and in 3D [BDDW99, Dam08] based on combinatorial maps. Indeed, combinatorial maps [Lie91] have many advantages: they are defined in any dimension; they are based on a single element called *dart*; they describe the subdivision of objects in cells, and incidence and adjacency relations between cells; thus they describe the topology of

Email addresses: guillaume.damiand@liris.cnrs.fr (Guillaume Damiand), christine.solnon@liris.cnrs.fr (Christine Solnon), cdlh@univ-nantes.fr (Colin de la Higuera), janodet@univ-st-etienne.fr (Jean-Christophe Janodet), emilie.samuel@univ-st-etienne.fr (Émilie Samuel)

Author manuscript, published in Pattern Recognition Letters, 32(9), pp. 1374-1383, July 2011. Thanks to Elsevier. The original publication is available at http://dx.doi.org/doi:10.1016/j.patrec.2010.09.005

objects. Actually, many works have used combinatorial maps in 2D and 3D image processing algorithms [BDB97, BDD01, DR02, DD08]. However, there is no algorithm for comparing combinatorial maps, which is an important issue for image processing and image analysis.

Contribution and outline of the paper

In this paper, we address two comparison problems, *i.e.*, *map isomorphism*, which involves deciding if two maps are equivalent, and *submap isomorphism*, which involves deciding if a copy of a pattern map may be found in a target map. We formally define these two problems for nD open combinatorial maps. Then we develop polynomial time algorithms for solving them, and illustrate their efficiency for searching patterns in 2D and 3D images.

Part of this work was published in [DDLHJ$^+$09]. Nevertheless, our previous results were limited to 2D; moreover, the material presented in Sections 4 and 5 is totally new.

In Section 2, we recall basic definitions and notations for nD combinatorial maps [Lie91], and from open combinatorial maps [PABL07] thus the paper is self-contained. We use open maps to represent objects with boundaries, that allow us to model images that have blurred or that contain undefined regions.

In Section 3, we first extend the definition of map isomorphism of [Lie94] to open combinatorial maps and give a polynomial time algorithm for solving this problem. This algorithm is close to that of [Cor75], but we extend it to nD open maps. Then we tackle the submap isomorphism problem and again, we develop a polynomial time algorithm for nD open maps. We prove the correctness and study the complexity of both algorithms. Note that from a graph-theoretical perspective, these results imply that the subisomorphism problem for plane graphs (that is, planar graphs that are embedded in a plane) is solvable in polynomial time, whereas this problem is known intractable for general graphs. However, our algorithms are restricted to patterns which are connected maps, such that there exists a path of sewn darts between every pair of darts.

In Section 4, we introduce planar maps, that are 2D maps embedded in a plane. The goal is to design a model that is close to standard pictures: images are usually drawn on the plane, thus one region, called the external or infinite region, is distinguished and should play a particular role. Comparing planar maps ultimately returns to the problem of comparing 2D maps and then checking whether the external regions are matched or not. Therefore, we essentially use the algorithms developed in Section 3, adding new constraints. However, the knowledge of external regions allows us to optimize the efficiency of these algorithms in practice (although the worst-case complexity does not change).

Finally, in Section 5, we describe a large experimental study that proves the efficiency of our procedures in practice. We first show how to use submap isomorphism for searching for a 2D subimage into a database of 2D images. Then we tackle the searching of a 3D submesh into a database of 3D objects. These results show the interest of having generic definitions and algorithms based on nD open maps. Indeed, we use the same method and algorithm in different types of applications, with different dimensions, the corresponding maps being open or closed.

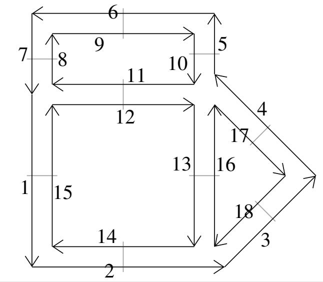
We conclude the paper on some related works and perspectives in Section 6.

2. Combinatorial Maps

An nD cellular complex is the subdivision of an nD object into cells of dimensions at most n (0D corresponding to vertices, 1D to edges, 2D to faces, 3D to volumes, *etc*), plus incidence and adjacency relationships between cells. Cellular complexes can be modeled with combinatorial maps, which provide a generic definition based on a single basic element called *dart*. In Section 2.1, we recall some definitions on combinatorial maps, and in Section 2.2, we give the definition of open combinatorial maps and related notions that we use to model cellular complexes with boundaries.

2.1. Recalls on combinatorial maps

Combinatorial maps were originally defined for 2D [Edm60, Tut63, Cor75], then extended to 3D [AK89, Spe91] and finally to nD [Lie91]. Below, we briefly recall the main definitions.



D	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
β_1	2	3	4	5	6	7	1	9	10	11	8	13	14	15	12	17	18	16
β_2	15	14	18	17	10	9	8	7	6	5	12	11	16	2	1	13	4	3

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

Definition 1 (Combinatorial map). An nD combinatorial map, (or n-map) is a tuple $M = (D, \beta_1, \dots, \beta_n)$ where

1. D is a finite set of darts;
2. β_1 is a *permutation*[1] on D;
3. $\forall i : 2 \leq i \leq n$, β_i is an *involution*[2] on D with no fixed point[3];
4. $\forall i : 1 \leq i \leq n-2$, $\forall j : i+2 \leq j \leq n$, $\beta_i \circ \beta_j$ is an involution on D.

We note β_0 for β_1^{-1}, and β_{ij} for $\beta_j \circ \beta_i$. Two darts d and d' such that $d = \beta_i(d')$ are said to be i-sewn. Let f be a function defined on a set E, and $X \subseteq E$, we denote $f(X) = \{f(x) | x \in X\}$.

Examples of 2D and 3D combinatorial maps are provided in Figures 1 and 2.

Intuitively, β_i defines adjacency relationships between cells of dimension i (e.g., edges for β_1, faces for β_2, volumes for β_3). β_1 is a permutation (so that $\beta_1(d)$ may be different from $\beta_1^{-1}(d)$) whereas all other β_i are involutions. Indeed, in 2D, a face is adjacent to at most one other face whereas an edge may be adjacent to two edges of the same face: $\beta_1(d)$ is the dart that follows d whereas $\beta_1^{-1}(d) = \beta_0(d)$ is the dart that precedes d. Note that β_1 may contain fixed points: in 2D, they correspond to loops, *i.e.* faces that are bordered by a single dart.

In combinatorial maps, cells are defined by means of *orbits*. Given a set E, a set $\{p_1, \dots, p_j\}$ of permutations on E and an element $e \in E$, the orbit $\langle p_1, \dots, p_j \rangle(e)$ is the set of all elements of E that can be reached from e by composing any $p_1, \dots, p_j$ and their inverses $p_1^{-1}, \dots, p_j^{-1}$.

The link between cells and orbits is given in Def. 2.

Definition 2 (i-cell). Let M be an n-map, and d a dart.

- The 0-cell incident to d is $\langle \beta_{02}, \dots, \beta_{0n} \rangle(d)$;
- $\forall i : 1 \leq i \leq n$, the i-cell incident to d is $\langle \beta_1, \dots, \beta_{i-1}, \beta_{i+1}, \dots, \beta_n \rangle(d)$.

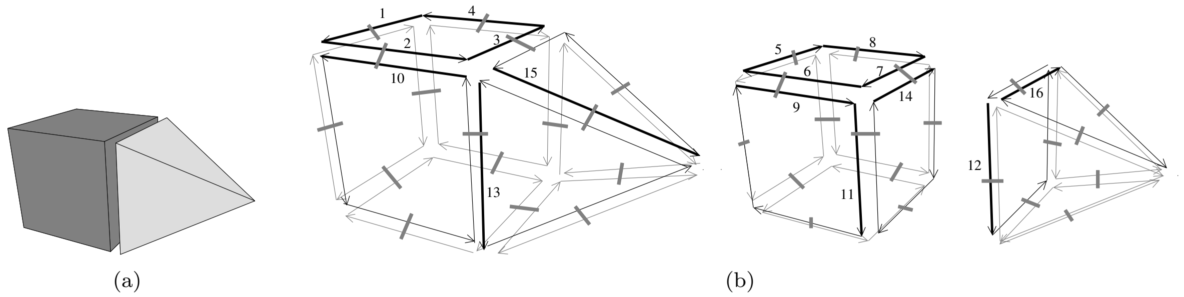
Hence, a cell is a set of darts.

- A 0-cell corresponds to a vertex and is a set of darts that share a same origin. In 2D, a 0-cell is obtained by the orbit $\langle \beta_{02} \rangle$ (e.g., $\langle \beta_{02} \rangle(8) = \{1, 8, 12\}$ in Fig. 1). In 3D, it is obtained by the orbit $\langle \beta_{02}, \beta_{03} \rangle$ (e.g., $\langle \beta_{02}, \beta_{03} \rangle(3) = \{3, 6, 10, 11, 12, 13, 14, 15, 16\}$ in Fig. 2);

[1] A permutation on D is a one-to-one mapping from D to D.

[2] An involution f on D is a one-to-one mapping from D to D such that $f = f^{-1}$.

[3] A fixed point of a function f is an element e such that $f(e) = e$.



D	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
β_1	2	3	4	1	8	5	6	7	11	...	...	...	...	...	...	...
β_2	...	10	...	...	...	9	14	...	6	2	...	...	...	7	...	...
β_3	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 1-cell corresponds to an edge and is a set of darts that share their endpoints. In 2D, it is obtained by the orbit $\langle\beta_2\rangle$ (e.g., $\langle\beta_2\rangle(8) = \{7, 8\}$ in Fig. 1). In 3D, it is obtained by the orbit $\langle\beta_2, \beta_3\rangle$ (e.g., $\langle\beta_2, \beta_3\rangle(2) = \{2, 6, 9, 10\}$ in Fig. 2);
- A 2-cell corresponds to a face. In 2D, it is obtained by the orbit $\langle\beta_1\rangle$ (e.g., $\langle\beta_1\rangle(8) = \{8, 9, 10, 11\}$ in Fig. 1). In 3D, it is obtained by the orbit $\langle\beta_1, \beta_3\rangle$ (e.g., $\langle\beta_1, \beta_3\rangle(1) = \{1, 2, 3, 4, 5, 6, 7, 8\}$ in Fig. 2);
- A 3-cell corresponds to a volume and is obtained in 3D by the orbit $\langle\beta_1, \beta_2\rangle$ (e.g., $\langle\beta_1, \beta_2\rangle(5)$ contains the 24 darts of the internal cube in Fig. 2).

We can define the incidence and adjacency relations between cells:

- Two cells are *incident* if their intersection is not empty. E.g., edge $\langle\beta_2\rangle(7) = \{7, 8\}$ is incident to face $\langle\beta_1\rangle(8) = \{8, 9, 10, 11\}$ in Fig. 1.
- Two i-cells are *adjacent* if there is an $(i-1)$-cell incident to both i-cells. E.g., faces $\langle\beta_1\rangle(8)$ and $\langle\beta_1\rangle(15)$ are adjacent because they are both incident to edge $\langle\beta_2\rangle(11)$ in Fig. 1.

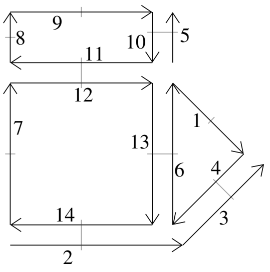
2.2. Open combinatorial maps

In [Lie91], Lienhardt has defined generalized maps, which can be used to model open or closed objects, thus allowing one to model objects with boundaries. A preliminary report [PABL07] extended the definition of nD combinatorial maps to open nD combinatorial maps.

Open maps may contain free darts, *i.e.*, darts that are not linked with other darts for some dimensions. In generalized maps [Lie91], a dart is i-free whenever it is i-sewn with itself (*i.e.*, $\beta_i(d) = d$). However, in combinatorial maps, β_1 may contain fixed points (in case of loops), thus such a trick is impossible to re-use. Therefore, to denote that a dart is 1-free, a new element $\varnothing$ is considered in addition to the set of darts, and darts can be 1-sewn with $\varnothing$. To make things similar in every dimensions, we use the same principle for i-free darts, with $i > 1$. Thus, a dart d is i-free, for $i \in \{0, \dots, n\}$, if $\beta_i(d) = \varnothing$.

However, if some darts are i-sewn with $\varnothing$, then β_i is no longer a permutation or an involution on D, but a partial permutation or partial involution such that only a subset $X \subseteq D$ of darts is linked to another subset of darts of D whereas other darts are i-free. We formally define a partial permutation as well as its inverse as follows.

Definition 3 (Partial permutation). Let E be a finite set. A mapping $p : E \cup \{\varnothing\} \to E \cup \{\varnothing\}$ is a *partial permutation* defined on E if:



	1	2	3	4	5	6	7	8	9	10	11	12	13	14
β_1	4	3	∅	6	∅	1	12	9	10	11	8	13	14	7
β_2	∅	14	4	3	10	13	∅	∅	∅	5	12	11	6	2

Figure 3: Open combinatorial map example. Darts 3 and 5 are 1-free, and darts 1, 7, 8 and 9 are 2-free.

1. $p(\varnothing) = \varnothing$;
2. $\forall e_1 \in E$, $\forall e_2 \in E$, $p(e_1) = p(e_2) \neq \varnothing \Rightarrow e_1 = e_2$.

The inverse p^{-1} of this partial permutation is also a partial permutation and is defined by:

1. $p^{-1}(\varnothing) = \varnothing$;
2. $\forall e \in E$, if it exists $a \in E$ such that $p(a) = e$, then $p^{-1}(e) = a$, otherwise $p^{-1}(e) = \varnothing$.

In other words, a partial permutation of E is a bijection between two subsets F and G of E such that the elements of E that do not belong to F or G are linked to $\varnothing$.

A partial involution is a specific partial permutation, satisfying the condition $f(f(e)) = e$ for any element e such that $f(e) \neq \varnothing$.

Definition 4 (Partial involution). A partial involution f on E is a partial permutation on E such that: $\forall e \in E$, $f(e) \neq \varnothing \Rightarrow f(f(e)) = e$.

Since a partial involution f is a partial permutation (with an additional property), its inverse is defined in Def. 3, and we have $f^{-1} = f$. Moreover, we can verify that, given two partial permutations f and g, $f \circ g$ is a partial permutation, and $(f \circ g)^{-1} = g^{-1} \circ f^{-1}$.

We can now define open combinatorial maps.

Definition 5 (Open combinatorial map). An open nD combinatorial map (or open n-map) is a tuple $M = (D, \beta_1, \ldots, \beta_n)$ where

1. D is a finite set of darts;
2. β_1 is a *partial permutation* on D;
3. $\forall i : 2 \leq i \leq n$, β_i is a *partial involution* on D with no fixed point;
4. $\forall i : 0 \leq i \leq n-2$, $\forall j : 3 \leq j \leq n$, $i + 2 \leq j$, β_{ij} is a partial involution.

Open n-maps differ from n-maps on three points: (1) β_1 is a partial permutation instead of a permutation; (2) other β_i are partial involutions instead of involutions; (3) Condition 4 is modified such that β_{ij} is a partial involution, and this condition must also be satisfied for $i = 0$. An example of open 2-map is represented in Fig. 3.

The first two differences directly come from the fact that we can have free darts. For the third difference, we need to add the condition on β_{0j} because β_{1j} is a partial involution does not imply that β_{0j} is a partial involution (contrary to the original definition of combinatorial maps where β_{1j} is an involution implies that β_{0j} is an involution).

A combinatorial map is said to be i-open (resp. i-closed), for $i \in \{1, \ldots, n\}$, if it contains at least one i-free dart (resp. no i-free dart). The map is said to be open (resp. closed) if it is i-open for at least one $i \in \{1, \ldots, n\}$ (resp. i-closed for all $i \in \{1, \ldots, n\}$). When nothing is specified, a combinatorial map may be either open or closed, and thus it is defined with respect to Def. 5.

From a mathematical standpoint, any combinatorial map denotes an nD cellular quasi-manifold [Lie94]. As every combinatorial map can easily be converted into a generalized map, this semantics still holds for our definition of open combinatorial map.

Now let us tackle the definition of cells.

The definition of orbits given in Section 2.1 is still valid for open combinatorial maps, since it uses "the set of all elements of E that can be reached…", which avoid to have $\varnothing$ in any orbit (since $\varnothing \notin E$ by partial permutation definition). For this reason, the Def. 2 of i-cells given for closed maps, is still valid for $i > 1$.

For instance, in the 2-map displayed in Fig. 3, the edge incident to dart 1 is $\langle\beta_2\rangle(1) = \{1\}$ (since dart 1 is 2-free) while the edge incident to dart 11 is $\langle\beta_2\rangle(11) = \{11, 12\}$. The face incident to dart 2 is $\langle\beta_1\rangle(2) = \{2, 3\}$.

However, we need to modify the definition of vertices (0-cells in Def. 6).

Definition 6 (0-cell). Let $M = (D, \beta_1, \ldots, \beta_n)$ be an n-map, and $d \in D$. The 0-cell incident to d is the set $\langle\beta_{02}, \ldots, \beta_{0n}, \{\beta_{ij} | \forall i, j : 2 \leq i < j \leq n\}\rangle(b)$.

The difference with the previous definition of 0-cells is that we need to add β_{ij} for all $2 \leq i < j \leq n$ in order not to miss some darts that cannot be reached due to some free darts. Let us consider, for example, the 0-cell incident to 3 in the closed 3-map of Fig. 2, that is, $\langle\beta_{02}, \beta_{03}\rangle(3) = \{3, 6, 10, 11, 12, 13, 14, 15, 16\}$. Some darts in this orbit are reached from 3 by composing some permutations, *e.g.*, $\beta_{02}(\beta_{03}(3)) = 11$. Let us now consider the 3-map obtained by removing darts 2 and 4 (*i.e.*, dart 3 is 0-free and 1-free). In this case, we have to use β_{23} and β_{23}^{-1} to reach other darts of the vertex incident to dart 3 (*e.g.*, $\beta_{23}(3) = 11$), since $\beta_{02}(3) = \beta_{03}(3) = \beta_{21}(3) = \beta_{31}(3) = \varnothing$.

In the example of the 2-map displayed in Fig. 3, the vertex incident to dart 12 is $\langle\beta_{02}\rangle(12) = \{8, 12\}$.

3. Map and Submap Isomorphism

(Sub)map isomorphism allows one to compare maps. In Section 3.1, we define map isomorphism —which allows one to decide the equivalence of two maps— and give a polynomial time algorithm for solving this problem. In Section 3.2, we consider submap isomorphism —which allows one to decide of the inclusion of a pattern map into a target map— and propose a polynomial time algorithm for solving this problem too.

In this paper, we only consider patterns which are connected maps, *i.e.* there exists a path of sewn darts between every pair of darts. Formally:

Definition 7 (Path). Let $M = (D, \beta_1, \ldots, \beta_n)$ be a combinatorial map. A sequence of darts $(d_1, \ldots, d_k)$ is a *path* between d_1 and d_k if
$\forall i : 1 \leq i < k, \exists j_i \in \{0, 1, \ldots, n\}, d_{i+1} = \beta_{j_i}(d_i)$.

Definition 8 (Connected map). A combinatorial map $M = (D, \beta_1, \ldots, \beta_n)$ is *connected* if $\forall d \in D, \forall d' \in D$, there exists a path between d and d'.

3.1. Map isomorphism

Lienhardt [Lie94] has defined isomorphism between two closed combinatorial maps. We extend this definition to open combinatorial maps as follows.

Definition 9 (Map isomorphism). Two n-maps $M = (D, \beta_1, \ldots, \beta_n)$ and $M' = (D', \beta'_1, \ldots, \beta'_n)$ are *isomorphic* if there exists a bijection $f : D \cup \{\varnothing\} \to D' \cup \{\varnothing\}$, called *isomorphism function*, such that $f(\varnothing) = \varnothing$ and $\forall d \in D, \forall i : 1 \leq i \leq n, f(\beta_i(d)) = \beta'_i(f(d))$.

The only difference with the definition of isomorphism between closed n-maps is that we have added that $f(\varnothing) = \varnothing$. Indeed, if a dart is i-sewn with $\varnothing$, then the dart matched to it by f must also be i-sewn with $\varnothing$.

As already mentioned in [Cor75] for 2D maps, an algorithm for deciding of the isomorphism of two maps can easily be derived from this definition. Indeed, consider Algorithm 1. We first fix a dart $d_0 \in D$ and, for every dart $d'_0 \in D'$, we call Algorithm 2 to build a candidate matching function f and then we check whether f actually is an isomorphism function. Algorithm 2 basically performs a traversal of M, starting from d_0 and using the β_i functions to discover new darts from darts that have already been discovered: initially, $f[d_0]$ is set to d'_0 whereas $f[d]$ is set to *nil* for all other darts, thus stating that d has not yet been discovered; then, each time a dart $d \in D$ is discovered, from another dart $d_k \in D$ such that d is i-sewn with d_k, then $f[d]$ is set to the dart $d' \in D'$ which is i-sewn with $f[d_k]$.

```
Algorithm 1: checkIsomorphism($M, M'$)
   Input: a connected $n$-map $M = (D, \beta_1, \ldots, \beta_n)$, and an $n$-map $M' = (D', \beta'_1, \ldots, \beta'_n)$
   Output: returns true iff $M$ and $M'$ are isomorphic
 1 choose $d_0 \in D$
 2 foreach $d'_0 \in D'$ do
 3     $f \leftarrow$ traverseAndBuildMatching($M, M', d_0, d'_0$)
 4     if $f$ is an isomorphism function then
 5         return True
 6 return False
```

```
Algorithm 2: traverseAndBuildMatching($M, M', d_0, d'_0$)
   Input: a connected $n$-map $M = (D, \beta_1, \ldots, \beta_n)$, an $n$-map $M' = (D', \beta'_1, \ldots, \beta'_n)$ and an initial
          couple of darts $(d_0, d'_0) \in D \times D'$
   Output: returns an array $f : D \cup \{\varnothing\} \to D' \cup \{\varnothing\}$
 1 foreach $d \in D$ do $f[d] \leftarrow nil$
 2 $f[d_0] \leftarrow d'_0$
 3 let $S$ be an empty stack; push $d_0$ in $S$
 4 while $S$ is not empty do
 5     pop a dart $d$ from $S$
 6     foreach $i \in \{0, \ldots, n\}$ do
 7         if $d$ is not $i$-free and $f[\beta_i(d)] = nil$ then
 8             $f[\beta_i(d)] \leftarrow \beta'_i(f[d])$
 9             push $\beta_i(d)$ in $S$
10 $f[\varnothing] \leftarrow \varnothing$
11 return $f$
```

We get the following result:

Theorem 1. *A connected n-map M is isomorphic to a map M'*
iff checkIsomorphism(M, M') *returns true.*

Proof. This corresponds to proving that Algorithm 1 is correct.

($\Leftarrow$) If checkIsomorphism(M, M') returns true, then M and M' are isomorphic since true is returned only if the test in line 4 succeeds.

($\Rightarrow$) Let us suppose that M and M' are isomorphic, so that there exists an isomorphism function $\varphi : D \to D'$, and let us show that checkIsomorphism(M, M') returns true. Let $d_0 \in D$ be the dart chosen at line 1 of Algorithm 1. As the loop lines 2-5 iterates on every dart $d'_0 \in D'$, there exists an iteration of this loop for which $d'_0 = \varphi(d_0)$. Let us show that for this iteration traverseAndBuildMatching(M, M', d_0, d'_0) returns the array f such that $\forall d \in D, f[d] = \varphi(d)$ so that true is returned line 5 of Algorithm 1:

- Claim 1: *When pushing a dart d in S, $f[d] = \varphi(d)$.* By induction. The claim holds for the push of line 3 as $f[d_0]$ is set to $d'_0 = \varphi(d_0)$ at line 2. The claim also holds for the push at line 9 as (1) $f[\beta_i(d)]$ is set to

$\beta'_i(f[d])$ in line 8 and (2) $f[d] = \varphi(d)$ (by induction hypothesis) and (3) $\varphi(d) = d' \Rightarrow \varphi(\beta_i(d)) = \beta'_i(d')$ (by definition of an isomorphism function).

- Claim 2: *Every dart $d \in D$ is pushed once in S.* Indeed, M is connected, so that there exists at least one path of sewn darts $(d_0, \ldots, d_k)$ such that $d_k = d$. Therefore, each time a dart d_i of this path is popped from S (line 5), d_{i+1} is pushed in S (line 9) if it has not been pushed before (through another path). □

The time complexity of Algorithm 1 is $\mathcal{O}(n \cdot |D| \cdot |D'|)$

Let us first show that the time complexity of Algorithm 2 is $\mathcal{O}(n \cdot |D|)$. Indeed, the **for** loop (lines 6-9) is iterated n times, and the **while** loop (lines 4-9) is iterated $|D|$ times as (1) exactly one dart d is removed from the stack S at each iteration, and (2) each dart $d \in D$ enters S at most once (it enters S only if $f[d] = nil$, and before entering S, $f[d]$ is set to a dart of D'). Let us then note that the test from line 4 of Algorithm 1 may be performed in $\mathcal{O}(n \cdot |D|)$. As Algorithm 2 and the test of line 4 are performed at most $|D'|$ times (once for each dart of M'), the overall time complexity of Algorithm 1 is $\mathcal{O}(n \cdot |D| \cdot |D'|)$.

Note that Algorithm 1 may be optimized, without changing its worst-case complexity. In particular, we can detect failure while building matchings by checking between lines 7 and 8 of Algorithm 2 that there does not exist a dart $d_j \in D$ that has already been matched to $\beta'_i(f[d])$: if this is the case, one can stop the current traversal as f will not be a bijection.

The condition that M is a connected map is necessary to ensure that all darts of M are discovered during the traversal, whatever the initial dart d_0 is. This condition is not necessary for the second map M' since the traversal is only guided by M (but of course if M is connected while M' not, Algorithm 1 return false).

3.2. Submap isomorphism

We now tackle the problem of submap isomorphism, the goal of which is to decide if there exists a copy of a pattern map in a target map.

Basically, a submap is obtained from a map by keeping only a subset X of its darts, and updating β_i functions so that every dart of X that is i-sewn with a dart that does not belong to X becomes i-free.

Definition 10 (Submap). Given a combinatorial map $M = (D, \beta_1, \ldots, \beta_n)$ and a subset of darts $X \subseteq D$, the *submap* of M induced by X, denoted $M_{\downarrow X}$ is the combinatorial map $(X, \gamma_1, \ldots, \gamma_n)$ such that:
$\forall d \in X$, $\forall i : 1 \leq i \leq n$, if $\beta_i(d) \notin X$ then $\gamma_i(d) = \varnothing$; else $\gamma_i(d) = \beta_i(d)$.

Note that, depending on X, the submap $M_{\downarrow X}$ may not satisfy condition 4 of Def. 5 so that it may not be a combinatorial map. Hence, we shall verify, when using Def. 10, that the subset X is such that $M_{\downarrow X}$ satisfies this condition.

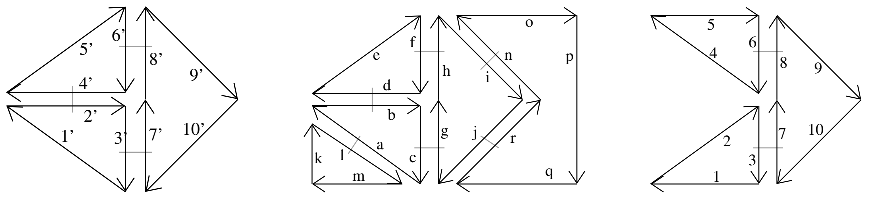
For example, if we consider the map shown in Fig. 2, and $X = D \setminus \{1\}$. In this case, $M_{\downarrow X}$ is not a combinatorial map since the condition that $\beta_1 \circ \beta_3$ must be an involution is not satisfied for dart 5 (we have $\beta_1 \circ \beta_3(5) = 4$ and $\beta_1 \circ \beta_3(5) = \varnothing$).

We can now define submap isomorphism as follows.

Definition 11 (Submap isomorphism). Let $M = (D, \beta_1, \ldots, \beta_n)$ and $M' = (D', \beta'_1, \ldots, \beta'_n)$ be two n-maps. M is *isomorphic to a submap* of M' if there exists a subset of darts $X \subseteq D'$ such that $M'_{\downarrow X}$ is isomorphic to M.

The subset X obviously satisfies condition 4 of Def. 5 since M is a valid combinatorial map, and $M'_{\downarrow X}$ is isomorphic to M.

The existency of a submap isomorphism implies the existency of a subisomorphism function which matches every dart of the pattern map to a different dart of the target map, so that pattern darts that are i-sewn are matched to target darts that are also i-sewn, like an isomorphism function. However, when a pattern dart d is i-free the target dart matched to d must either be i-free, or it must be i-sewn with a target dart which is not matched to another pattern dart (see example in Fig. 4). This is more formally stated in Theorem 2.



(a) M

(b) M'

(c) M''

Figure 4: Submap isomorphism example. M is a submap of M' as it is obtained from M' by deleting darts k to r. M'' is not isomorphic to a submap of M' as the injection $f : X \to D$ that respectively matches darts 1 to 10 to darts a to j does not verify Theorem 2: for example, dart 4 is 2-free and it is matched to dart d which is 2-sewn with a dart (b) which is itself matched (*i.e.*, $f^{-1}(b) = 2$).

Theorem 2. *M is isomorphic to a submap of M' iff there exists an injection $f : D \cup \{\varnothing\} \to D' \cup \{\varnothing\}$, called a* subisomorphism function*, such that:*

1. *$f(\varnothing) = \varnothing$ and*
2. *$\forall d \in D, \forall i : 1 \leq i \leq n$,*
   - *if d is not i-free, then $\beta'_i(f(d)) = f(\beta_i(d))$;*
   - *otherwise, either $f(d)$ is i-free, or $\forall d_k \in D, f(d_k) \neq \beta'_i(f(d)))$.*

Proof. ($\Rightarrow$) If M is isomorphic to a submap of M', then there exists a subset $X \subseteq D'$ such that M is isomorphic to $M'_{\downarrow X}$. By Def. 9, there exists a bijection $f : D \cup \{\varnothing\} \to X \cup \{\varnothing\}$ such that $f(\varnothing) = \varnothing$ and $\forall d \in D, \forall i : 1 \leq i \leq n, f(\beta_i(d)) = \beta'_i(f(d))$. Let us define function $g : D \cup \{\varnothing\} \to D' \cup \{\varnothing\}$ such that $\forall d \in D, g(d) = f(d)$ and $g(\varnothing) = \varnothing$, and let us show that g is a subisomorphism function. Obviously, $\forall i : 1 \leq i \leq n$, if d is not i-free, then $\beta'_i(g(d)) = g(\beta_i(d))$. The key point is to show that if d is i-free, then either $g(d)$ is i-free or $\forall d_k \in D, f(d_k) \neq \beta'_i(f(d)))$. Let us suppose that $g(d)$ is not i-free and there exists $d_k \in D$ such that $g(d_k) = \beta'_i(g(d))$. As f is an isomorphism function and g is the restriction of f to X, we have $g(d_k) = \beta'_i(g(d)) = g(\beta_i(d))$ so that $d_k = \beta_i(d)$ which is in contradiction with the fact that d is i-free.

($\Leftarrow$) Let us suppose that there exists a subisomorphism function $f : D \cup \{\varnothing\} \to D' \cup \{\varnothing\}$. We must show that M is isomorphic to a submap of M', *i.e.*, that there exists a subset $X \subseteq D$ such that M is isomorphic to $M'_{\downarrow X}$. Obviously, we define $X = \{f(d) | d \in D\}$ and function $g : D \to X$ such that $g(\varnothing) = \varnothing$ and $\forall d \in D, g(d) = f(d)$. □

Algorithm 3 determines if there is a submap isomorphism between a connected map and a map. It is based on the same principle as Algorithm 1; the only difference is the test of line 4, which succeeds if f is a subisomorphism function instead of an isomorphism function.

Algorithm 3: checkSubIsomorphism(M, M')

```
Input: a connected map $M = (D, \beta_1, \ldots, \beta_n)$, and a map $M' = (D', \beta'_1, \ldots, \beta'_n)$
Output: returns true iff $M$ is isomorphic to a submap of $M'$
1 choose $d_0 \in D$
2 foreach $d'_0 \in D'$ do
3     $f \leftarrow$ traverseAndBuildMatching($M, M', d_0, d'_0$)
4     if $f$ is a subisomorphism function then
5         return True
6 return False
```

Correctness proofs and evidences given for isomorphism are still valid: we solve the submap isomorphism problem with the same method as before, except that function f is now an injection instead of a bijection.

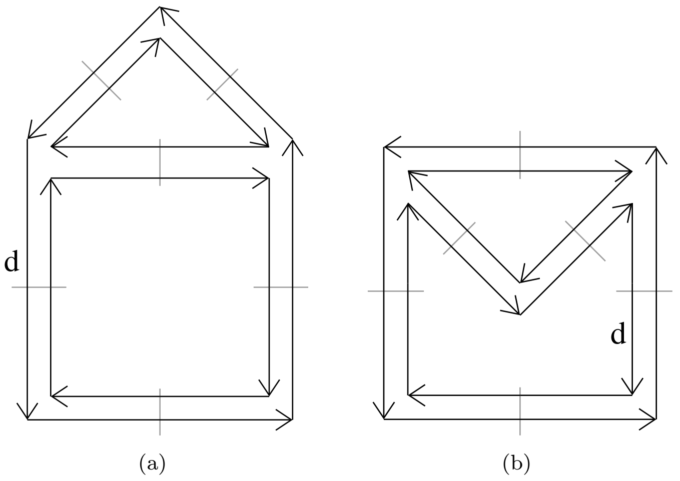


Figure 5: Two maps that are isomorphic but not planar-isomorphic.

The time complexity of this algorithm is $\mathcal{O}(n \cdot |D| \cdot |D'|)$ as TRAVERSEANDBUILDMATCHING is called at most $|D'|$ times and its complexity is $\mathcal{O}(n \cdot |D|)$. Note that the subisomorphism test may be done in linear time.

Also, one may optimize Algorithm 3 in a similar way as proposed for Algorithm 1, without changing its worst-case complexity, by checking between lines 7 and 8 of Algorithm 2 that there does not exist a dart $d_j \in D$ that has already been matched to $\beta'_i(f[d])$: if this is the case, one can stop the current traversal as f will not be an injection.

4. Constrained Isomorphisms

In 2D image processing, the type of maps one wishes to consider have the property that they are to be drawn on the plane, not on the sphere. We call this type of map a *planar combinatorial map.* To understand this point, let us consider the typical situation represented in Fig. 5. These two maps are isomorphic on the sphere, which corresponds to the definitions introduced in Section 3.1, but are not isomorphic in the plane. In other words, for these two maps, Algorithm 3 returns true, but that is probably not what is intended in an image processing task.

The property of being planar isomorphic is a geometrical property, *i.e.* it cannot be characterized for maps without geometry. Indeed, two maps are planar isomorphic when they are isomorphic and when the image of the exterior of the first map is equal to the exterior of the second map.

Let us consider, for example, the two maps displayed in Fig. 6(b) and Fig. 6(c). These maps are isomorphic but not planar isomorphic. Indeed, the image of the exterior darts of the map in Fig. 6(b) are not exterior darts in Fig. 6(c). This means that we have reversed the first map which is not a possible operation in the plane. This is similar for planar submap isomorphism. In our example, the map of Fig. 6(a) is a submap of the two maps of Fig. 6(b) and Fig. 6(c), but it is only a planar submap of the map of Fig. 6(b) since only in this case, the image of the exterior darts of Fig. 6(a) are still at the exterior of the second map.

In Section 4.1, we formally define planar maps by introducing the notion of exterior and infinite darts of a map. In Section 4.2, we extend (sub)map isomorphism to planar (sub)map isomorphism by adding constraints on exterior and infinite darts. In Section 4.3, we show how to compute exterior and infinite darts in the case of 1-closed combinatorial maps (*i.e.* when all faces are closed).

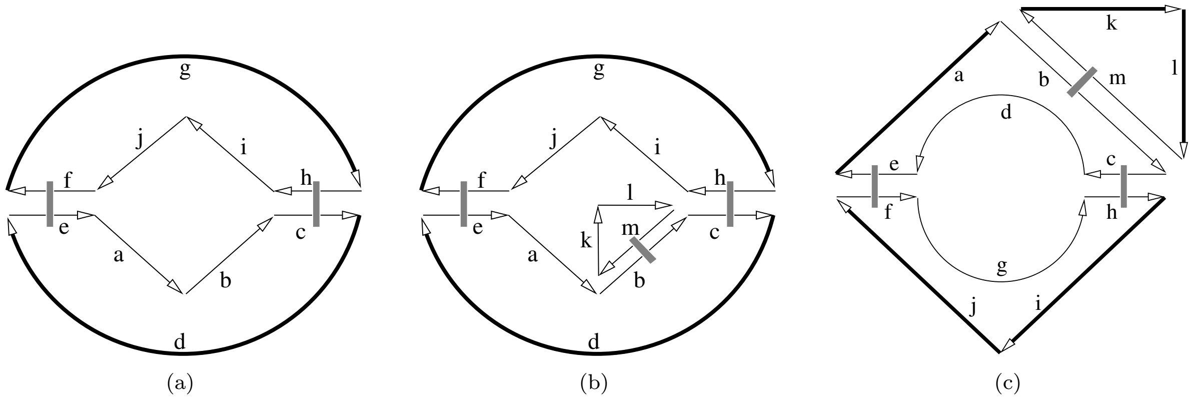


Figure 6: Isomorphisms of maps with undefined parts on a sphere *versus* in the plane: on a sphere, (b) and (c) are isomorphic, and (a) is a submap of both (b) and (c). However, in the plane (b) and (c) are not isomorphic, and (a) is a submap of (b) but not of (c). Exterior darts are drawn in bold.

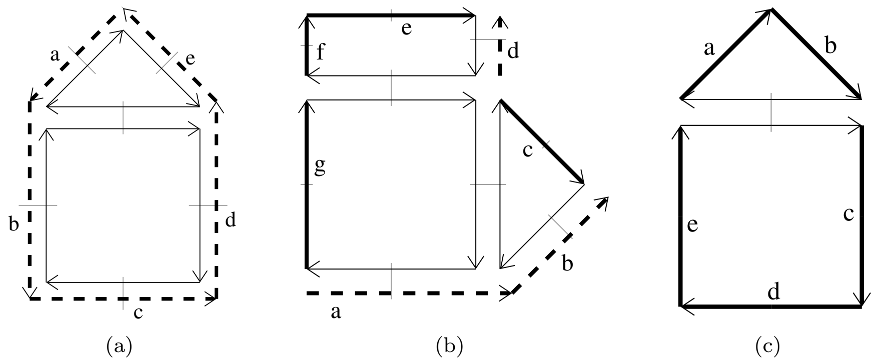


Figure 7: Exteriors darts are drawn in bold, and infinite darts in dash. Darts in bold and dash are both exterior and infinite. (a) A closed map: $Inf = Ext = \{a, b, c, d, e\}$. (b) A map with an open infinite face: $Inf = \{a, b, d\}$ and $Ext = \{a, b, c, d, e, f, g\}$. (c) A map without infinite face: $Inf = \emptyset$ and $Ext = \{a, b, c, d, e\}$.

4.1. Planar maps

When considering a map drawn on the plane, a part of the plane corresponds to the exterior of the modeled object. There are 3 cases to consider:

- If there exists an infinite (or unbounded) face, and if this infinite face is closed, then the exterior is defined by the darts of this infinite face, as displayed in Fig. 7(a).
- If there exists an infinite face, but this face is open, then the exterior is defined by the darts of this open infinite face, plus the set of 2-free darts that border the exterior, as displayed in Fig. 7(b);
- If there does not exist an infinite face, then the exterior is defined by the set of 2-free darts that border the exterior, as displayed in Fig. 7(c).

Hence, a planar map is defined by a map, together with a set Ext of exterior darts, and a set $Inf \subseteq Ext$ of darts that belong to the infinite face (called infinite darts).

The last consideration to take into account is face orientation. Indeed, there are two possible orientations of each planar combinatorial map: by taking β_1 such that faces are oriented clockwise (like Fig. 8(b)) or counterclockwise (like Fig. 8(a)). Note that all the faces of a map have the same orientation, except the

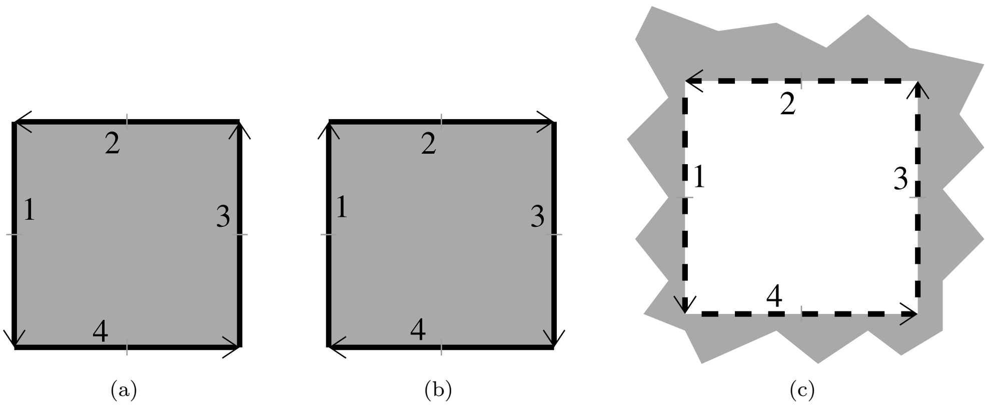


Figure 8: Three maps which are isomorphic, but not planar isomorphic. (a) and (b) are two maps with different orientations. (b) and (c) have the same orientation but (b) has no infinite face while (c) has one.

infinite face which has a reversed orientation. This orientation is once again a geometrical property, and it is required to use the same orientation for all the considered planar maps. Thus, in this section, we suppose that the same orientation is chosen for all the planar maps.

Definition 12 (Planar map). A *planar map* is a triple $PM = \langle M, Ext, Inf \rangle$ such that (1) $M = (D, \beta_1, \beta_2)$ is a combinatorial map drawn on the plane with a given orientation; (2) $Ext \subseteq D$ is the set of exterior darts; (3) $Inf \subseteq Ext$ is the set of infinite darts.

Note that when we consider a connected map, there is at most one infinite face. This is not the case for non-connected maps that may have several infinite faces (at most one for each connected component).

4.2. Planar map isomorphism

When comparing two planar maps in order to decide if they are isomorphic, one has to check that the isomorphism function actually matches exterior and infinite darts, as stated in Def. 13.

Definition 13 (Planar map isomorphism). Two planar maps $PM = \langle M, Ext, Inf \rangle$ and $PM' = \langle M', Ext', Inf' \rangle$ are planar isomorphic if there exists an isomorphism function f between M and M' such that $f(Ext) = Ext'$ and $f(Inf) = Inf'$.

The test on exterior darts is not enough to process all the possible cases. Let us consider for example the maps of Fig. 8(b) and Fig. 8(c). These two maps are isomorphic and have the same set of exterior darts, but they are not planar isomorphic since the first one represents a square face while the second one represents the plane minus a square. The condition on the infinite face allows to distinguish these two cases since $Inf = \emptyset$ in the first map, whereas $Inf \neq \emptyset$ in the second map.

The definition of a submap is also extended to planar maps as follows.

Definition 14 (Planar submap). Given a planar map $PM = \langle M, Ext, Inf \rangle$ with $M = (D, \beta_1, \beta_2)$, and a subset of connected darts $X \subseteq D$. The planar submap of PM induced by X is the planar map $PM_{\downarrow X} = \langle M_{\downarrow X}, Ext', Inf' \rangle$ such that $M_{\downarrow X}$ is the submap of M induced by X, Ext' is the set of exterior darts of $M_{\downarrow X}$ and Inf' is the set of infinite darts of $M_{\downarrow X}$.

The way to compute Ext' and Inf' depends on how the combinatorial map is linked with some geometrical elements. In particular, we show in Section 4.3 how these sets can be computed when combinatorial maps are 1-closed.

Planar submap isomorphism is now defined in a straightforward way with respect to planar submap and planar map isomorphism.

Definition 15 (Planar submap isomorphism). A planar map PM is planar submap isomorphic to a planar map PM' if there exists a planar submap of PM' which is planar isomorphic to PM.

4.3. The case of 1-closed combinatorial maps

In the field of image processing, faces are either entirely represented or not at all. Technically, this means that β_1 should be a permutation and not a partial permutation. We use this property in this section to simplify handling of exterior and infinite darts, and to show how to compute Ext and Inf sets for the planar submap definition.

If a map is 1-closed, we have only two cases to consider: the infinite face is either present and closed or totally absent. Therefore, we have either $Inf = \emptyset$, or $Inf = Ext$. Thus, we do not have to explicitly list the set of infinite darts Inf, but we can simply use a boolean $infinite$ which is true if the infinite face is present and false otherwise.

For each of these two cases, the subset Ext may be defined by giving only one dart d_{ext} of the set:

- When the infinite face is defined, given a dart d_{ext} belonging to this face, the whole set of exterior darts corresponds to the orbit $\langle\beta_1\rangle(d_{ext})$. For example, in Fig. 7(a), $Ext = \langle\beta_1\rangle(a) = \{a, b, c, d, e\}$.
- When the infinite face is not defined, the exterior darts correspond to the set of 2-free darts that border the exterior of the map. In this case, given a 2-free dart d_{ext} that borders the exterior of the map, the whole set of exterior darts may be computed by Algorithm 4. For example, in Fig. 7(c), $Ext = \text{BUILDEXTERIOR}(M, a, false) = \{a, b, c, d, e\}$.

Note that in both cases, there may exist 2-free darts that are not exterior darts. These 2-free darts border undefined parts of the map. Let us consider, for example, the map displayed in Fig. 6(a): the set of exterior darts is $\{d, g\}$; the other 2-free darts, *i.e.*, $\{a, b, i, j\}$, border an undefined part inside the map.

```
Algorithm 4: BUILDEXTERIOR($M, d_{ext}, infinite$)
  Input: a connected map $M = (D, \beta_1, \beta_2)$, an initial exterior dart $d_{ext} \in D$, and a boolean $infinite$
         which is true if $d_{ext}$ belongs to the infinite face, false otherwise
  Output: the set of all exterior darts
1 if $infinite$ is true then return $\langle\beta_1\rangle(d_{ext})$
2 $Ext \leftarrow \emptyset$; $d_{curr} \leftarrow d_{ext}$
3 repeat
4     $Ext \leftarrow Ext \cup \{d_{curr}\}$
5     $d_{curr} \leftarrow \beta_1(d_{curr})$
6     while $d_{cur}$ is not 2-free do
7         $d_{curr} \leftarrow \beta_{21}(d_{curr})$
8 until $d_{curr} = d_{ext}$ ;
9 return $Ext$
```

Theorem 3. *Given* $(M, d_{ext}, infinite)$*,* BUILDEXTERIOR *computes all the exterior darts of* M*.*

PROOF. As the map is 1-closed, there are only two possible cases: the infinite face is present and thus $Ext = Inf = \langle\beta_1\rangle(d_{ext})$, or this face is not present and in this case, the exterior darts can all be reached starting from d_{ext}, and keeping all the 2-free darts that border the exterior of the map. For that, given an exterior dart, we go to the next dart of the same face (line 5), then we jump over darts inside the map (*i.e.* which are not 2-free) by using $\beta_{21}(d_{curr})$ as many time as necessary to reach a 2-free dart (lines 6-7). We are sure that such a dart exists because the map is connected and 1-closed. □

This algorithm has a linear time complexity with respect to the number of darts.

We can now simplify the notation of planar maps by $PM = \langle M, d_{ext}, infinite\rangle$ since we can retrieve Ext and Inf given d_{ext} and $infinite$, for 1-closed maps.

Note that two planar maps may be planar isomorphic only if the two sets of exterior darts have the same cardinality and either both correspond to an infinite face, or both correspond to a set of 2-free darts that border the exterior of the maps. If this is not the case, one can trivially conclude that the two planar maps are not isomorphic; otherwise, one has to search for an isomorphism function that matches Ext and Ext'. In this later case, one may use the fact that darts of Ext must be matched to darts of Ext' to boost the research, as described in Algorithm 5.

Algorithm 5: CHECKPLANARISOMORPHISM(PM, PM')

```
Input: a planar connected map $PM = \langle M, d_{ext}, infinite\rangle$, and a planar map
       $PM' = \langle M', d'_{ext}, infinite'\rangle$
Output: returns true if and only if the maps are planar-isomorphic
if $infinite \neq infinite'$ then return false
$Ext \leftarrow$ buildExterior($M, d_{ext}, infinite$)
$Ext' \leftarrow$ buildExterior($M', d'_{ext}, infinite'$)
if $|Ext| \neq |Ext'|$ then return false
foreach $d'_0 \in Ext'$ do
    $f \leftarrow$ traverseAndBuildMatching($M, M', d_{ext}, d'_0$)
    if $f$ is an isomorphism function between $M$ and $M'$ then
        return true
return false
```

Theorem 4. CHECKPLANARISOMORPHISM(PM, PM') *returns true iff PM and PM' are planar isomorphic.*

PROOF. First, if PM and PM' are planar isomorphic, there is an isomorphism function f such that $f(Ext) = Ext'$ and $f(Inf) = Inf'$. Thus, we have necessarily $infinite = infinite'$ and $f(d_{ext}) = d'_0 \in Ext'$. By testing all the darts $d'_0 \in Ext'$, we are sure to find this case and thus the algorithm will return true. Second, if our algorithm returns true, we have an isomorphism function between M and M' such that $infinite = infinite'$ and $f(d_{ext}) = d'_0 \in Ext'$. Since we are in the case of 1-closed maps, this implies that $f(Inf) = Inf'$ and $f(Ext) = Ext'$ and thus PM and PM' are planar isomorphic. □

Remember that we are able to decide of the isomorphism of 2-maps in $\mathcal{O}(|D| \cdot |D'|)$ time. Having an information about the set of exterior darts and an efficient way to compute these darts allows us to reduce the complexity to $\mathcal{O}(|Ext'| \cdot |D|)$. In many cases, $|Ext'|$ is significantly smaller than $|D'|$. Typically, $|Ext'| = \mathcal{O}(\sqrt{|D'|})$ for the lattice $\mathbb{Z}^2$.

Now let us re-consider the planar submap problem. By using the fact that maps are 1-closed, we are now able to compute combinatorially Ext' and Inf' in the submap definition. First, we need to add a constraint in the submap definition. Indeed, given a planar map $PM = (M, Ext, Inf)$, the subset of darts $X \subseteq D$ must ensure that the submap $PM_{\downarrow X}$ is always 1-closed. For this reason, for each face f of the initial map, X must either contain all the darts of f, or no dart of f, but it cannot contain only some darts of f, otherwise some darts are 1-free.

If the submap is 1-closed, then the infinite face is either totally kept in the submap, or totally removed. This allows us to compute simply the new boolean $infinite'$ of the submap. Indeed, if $infinite$ is false, the initial map M does not have an infinite face, and thus the submap also. Otherwise, we have $Ext = Inf$ and thus $d_{ext} \in Inf$. Now there are again two cases: if $d_{ext} \in X$, then the infinite face is totally selected and thus is present in the submap, otherwise the infinite face is not selected and thus the submap does not have an infinite face.

For d'_{ext}, there are also two cases to consider. If $d_{ext} \in X$, then this dart is present in the submap and thus it belongs to the exterior of the submap, so that $d'_{ext} = d_{ext}$. Otherwise, we need to find a new exterior dart of the submap. This can be achieved easily by searching a dart $d \in X$ which can be reached from d_{ext} by a path of darts that uses only darts that do not belong to X (except d of course). Indeed, such a dart belongs necessarily to the exterior of the submap induced from X due to the fact that the pattern map is connected. Such a path may be found by a simple traversal of the darts of D starting from d_{ext} and avoiding darts of X. The key point is to ensure that we are always able to compute the dart d'_{ext} by using such a path. Suppose it is not the case, then there is no path between d_{ext} and one dart of X and this implies that the map is not connected.

More formally, we define planar submap for 1-closed maps as follows.

Definition 16 (1-closed planar submap). Given a 1-closed connected planar map $PM = \langle M, d_{ext}, infinite \rangle$ with $M = (D, \beta_1, \beta_2)$, and a subset of connected darts $X \subseteq D$. The planar submap of PM induced by X is the planar map $PM_{\downarrow X} = \langle M_{\downarrow X}, d_{ext'}, infinite' \rangle$ such that $M_{\downarrow X}$ is 1-closed, with:

- $infinite' =$ true if $infinite = true$ and $d_{ext} \in X$;
  $infinite' =$ false otherwise;
- $d'_{ext} = d_{ext}$ if $d_{ext} \in X$;
  $d'_{ext} = d \in X$ with a path $(d_{ext}, \ldots, d)$ of darts $\notin X$ (except d) otherwise.

Algorithm 6: CHECKPLANARSUBISOMORPHISM(PM, PM')

```
Input: a connected planar map $PM = \langle M, d_{ext}, inf \rangle$, and a planar map $PM' = \langle M', d'_{ext}, inf' \rangle$
Output: returns true iff $PM$ is planar isomorphic to a submap of $PM'$
foreach dart $d'_0$ of $M'$ do
    $f \leftarrow$ TRAVERSEANDBUILDMATCHING($M, M', d_{ext}, d'_0$)
    if $f$ is a subisomorphism function then
        $PM'' = PM'_{\downarrow f(D)}$
        if CHECKPLANARISOMORPHISM($PM, PM''$) then
            return True
return False
```

Algorithm 6 shows how to decide if a planar map is isomorphic to a submap of another planar map. The basic idea is to compute subisomorphism functions and check if they satisfy the additional planarity constraint. We may have to compute several subisomorphism functions since it is possible that there are different subisomorphisms, and that some of them are planar while others are not.

Theorem 5. CHECKPLANARSUBISOMORPHISM(PM, PM') *returns true iff PM is planar isomorphic to a submap of PM'.*

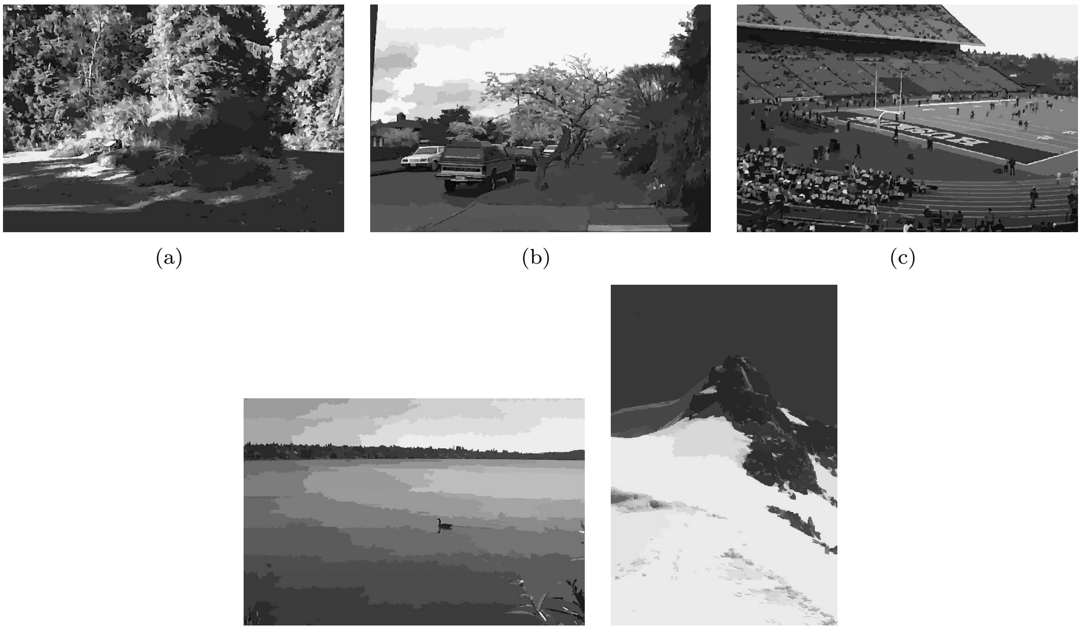
PROOF. First, if PM is planar isomorphic to a submap of PM', as we compute all the subisomorphism functions between M and M', we are sure to find the one satisfying planarity constraints and thus the algorithm returns true. Second, if our algorithm returns true, there is a subisomorphism satisfying the planarity constraints and thus PM is planar isomorphic to a submap of PM'. □

The complexity of the planar subisomorphism algorithm is $\mathcal{O}(|D|^2 \cdot |D'|)$.

One loop of this algorithm is done by using Algorithm 2, which complexity (in 2D) is $\mathcal{O}(|D|)$. The complexity of planar submap computation is $\mathcal{O}(|D'|)$, thanks to the efficient computation of $infinite''$ and d''_{ext}. Finally the test of planar isomorphism is achieved by using Algorithm 5, the complexity of which is $\mathcal{O}(|Ext''| \cdot |D|)$. Since the number of darts in Ext'' is smaller than the number of darts in D'', we can bound the overall complexity of one loop by $\mathcal{O}(|D|^2)$ since $|D| = |D''|$.

	# maps	# darts			# vertices			# edges			# faces		
		avg	min	max	avg	min	max	avg	min	max	avg	min	max
2D targets	224	14,509	1,768	35,896	4,789	594	11,811	7,255	884	17,948	2,598	336	6,231
2D patterns	224	2,175	48	22,338	718	16	7,365	1,088	24	11,169	384	14	3,950
3D targets	800	179,540	672	1,337,652	10,683	42	98,256	24,502	92	167,219	14,962	56	111,471
3D patterns	62	23,413	456	175,584	1,704	40	14,848	5,033	114	44,119	3,326	76	29,264

Figure 9: Characteristics of the two databases: the first two lines describe the 2D database, the last two lines describe the 3D database. For each line, we first give the number of maps and then the number of darts, vertices, edges and faces (average, minimum and maximum values).



(a)

(b)

(c)

(d)

(e)

Figure 10: Examples of 2D segmented images. (a) arbogreens05. (b) cherries14. (c) football05. (d) greenlake06. (e) swiss-mountains22.

5. Experiments

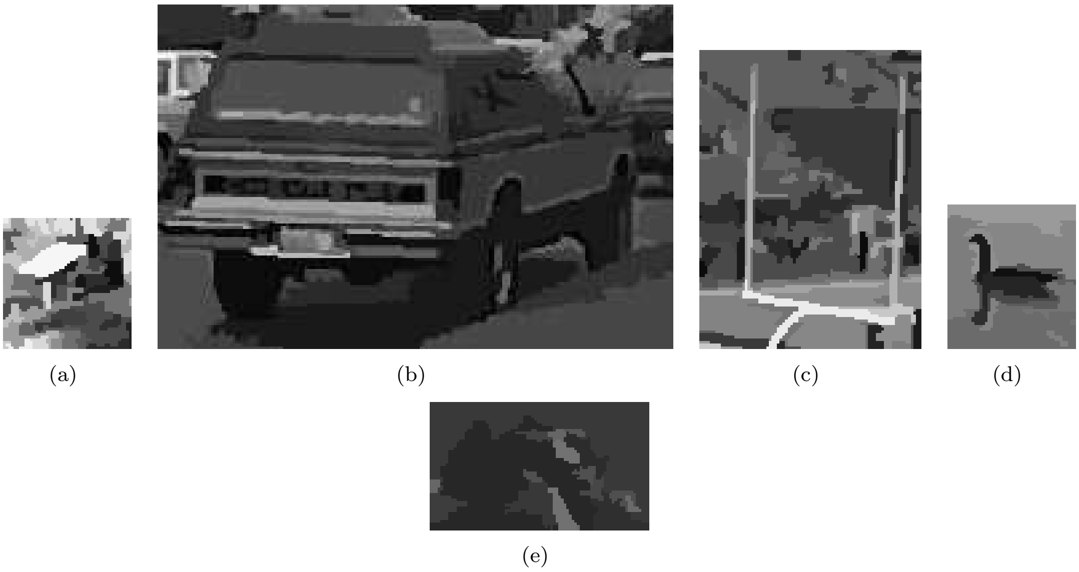
In this section, we carry out two experiments that show the effectiveness of our algorithm to detect patterns in images. The first challenge consists in searching for a sub-image from a database of 2D segmented images —a typical "Where's Wally" challenge. The second one aims at retrieving a sub-mesh from a database of 3D objects. These experiments illustrate the interest of generic algorithms for submap isomorphism. Indeed, the same procedure is used whatever the dimension of the maps used to model objects (2D, 3D or more).

5.1. Description of the two databases

We have considered two different databases, the characteristics of which are summarized in table 9.

The 2D database has been generated from 224 segmented images[4]. For each of these 224 segmented images, we have computed a 2D-map, thus obtaining 224 target 2D-maps. We have used the algorithm of [DBF04] to build 2D-maps from segmented images. The basic idea of this algorithm is to build the map

[4] The database of free images for research purpose is available on Internet `http://www.cs.washington.edu/research/imagedatabase/`


(a) (b) (c) (d)

(e)

Figure 11: Some selected sub-images. (a) from arbogreens05, size 48 × 49 pixels. (b) from cherries14, size 193 × 129 pixels. (c) from football05, size 83 × 112 pixels. (d) from greenlake06, size 48 × 54 pixels. (e) from swissmountains22, size 82 × 48 pixels.

in an incremental way: each pixel is considered iteratively and, for each pixel, a square face is created and sewn to the combinatorial map. Then, this square face is merged with faces which are adjacent to it and which belong to the same region as it. Finally, consecutive edges separating the same two regions are merged so that two adjacent regions are separated by exactly one edge (see [DBF04] for more details). From each segmented image, we have randomly extracted a sub-image, thus obtaining 224 pattern maps. Figures 10 and 11 display five segmented images and the subimages which have been extracted from them.

The 3D database has been generated from 800 objects[5] represented in 3D, thus defining 800 target 3D-maps. From these 800 3D-maps, we have randomly extracted 62 pattern 3D-maps. Figures 12 and 13 display four 3D objects and the sub-meshes which have been extracted from them.

Since the patterns we are searching for must be connected by definition, we have only considered connected pattern submaps for the 2D and 3D databases.

5.2. Submap vs subgraph isomorphism

We compare our submap algorithm with algorithms for solving subgraph isomorphism problems. For each 2D or 3D map M, we have generated a graph $G = (V, E)$ such that V associates a vertex with every 0-cell of M and E associates an edge with every 1-cell of M, as illustrated in Fig. 14.

To search for patterns, we consider the partial subgraph isomorphism problem. Indeed, a pattern graph associated with a pattern map is not necessarily an induced subgraph of the target graph associated with the target map, as illustrated in Fig. 14. Hence, in the 2D (resp. 3D) database, 71 patterns (resp. 1) are not found in the corresponding target if we look for an induced subgraph instead of a partial subgraph.

Looking for partial subgraph isomorphisms allows us to find all submap isomorphisms. However, some of the subgraph isomorphisms may not correspond to submap isomorphisms as topological relationships are ignored, as illustrated in Fig. 14.

We have considered three different algorithms for solving subgraph isomorphism problems, *i.e.*, Vflib [CFSV04], ILF [ZDS10], and LAD [Sol10]. All of them perform a systematic exploration of the search space

[5] These objects are available in the Shape Retrieval Contest web page `http://www.aimatshape.net/event/SHREC/`

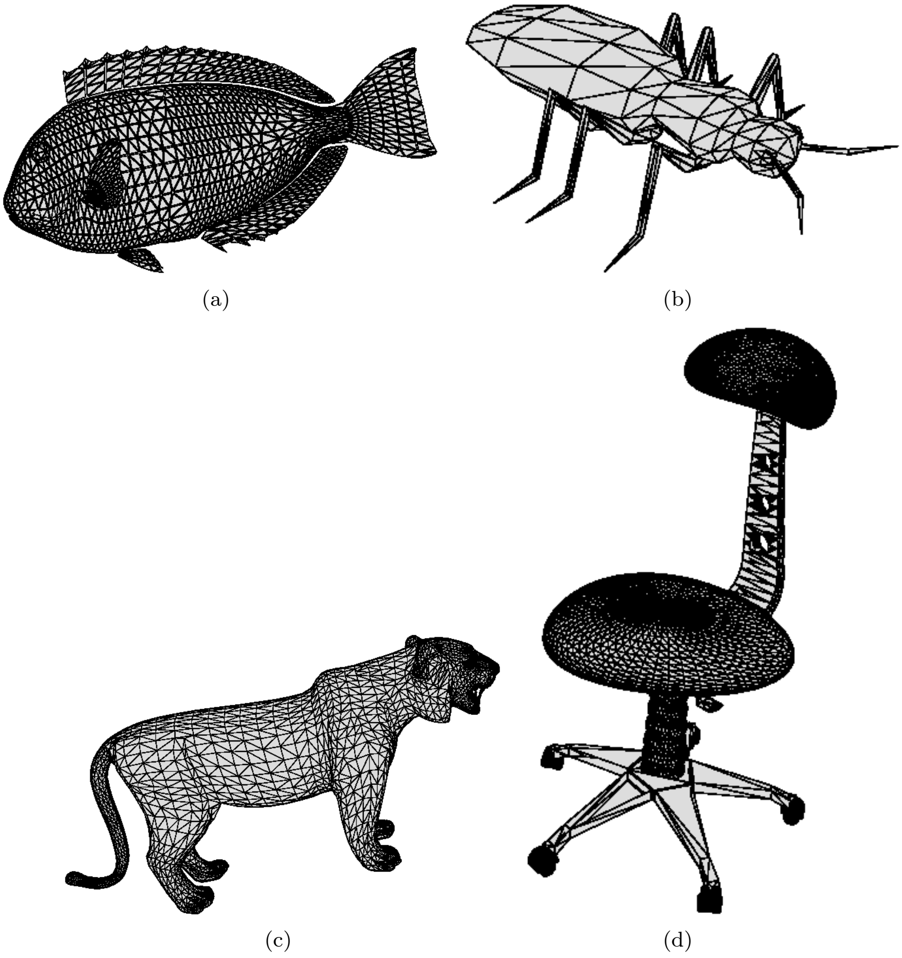


Figure 12: Examples of 3D objects.

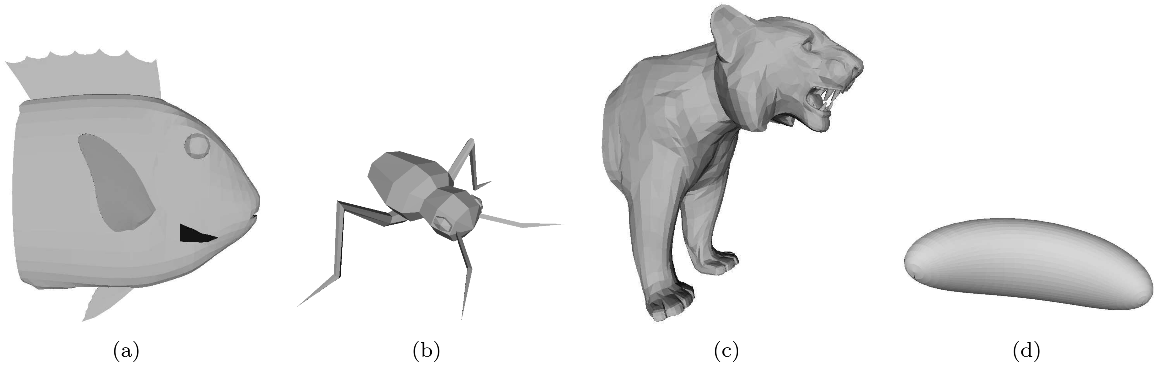


Figure 13: Some selected sub-meshes.

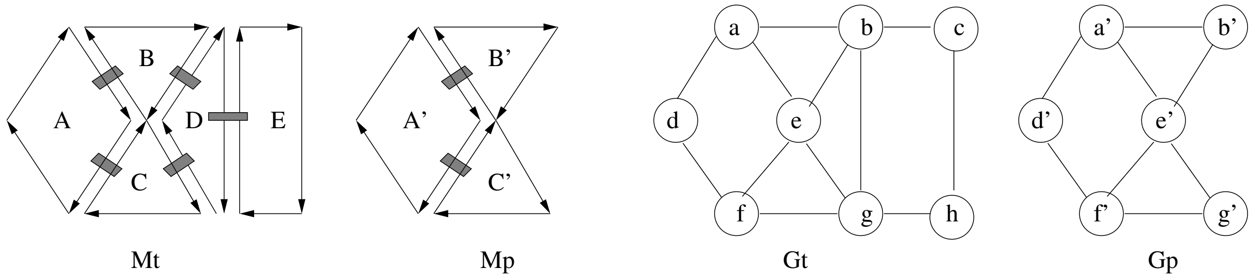


Figure 14: Example of graphs associated with 2D-maps: G_t (resp. G_p) is the graph generated from M_t (resp. M_p). M_p is a submap of M_t. There is only one submap isomorphism which matches faces A', B', and C' to faces A, B, and C respectively (we cannot exchange faces B' and C' without violating topological constraints). G_p is a partial subgraph of G_t; it is not an induced subgraph as edge (b', g') does not belong to G_p whereas edge (b, g) belongs to G_t. There are two partial subgraph isomorphisms: the first one matches vertices a', b', d', e', f', and g', to vertices a, b, d, e, f, and g respectively; the second one is obtained from the first one by exchanging a' and b' with f' and g'.

	2D database: 224 patterns/224 targets				3D database: 62 patterns/800 targets			
	Map	ILF	LAD	Vflib	Map	ILF	LAD	Vflib
# fail	0	0	1	8	0	6	6	40
avg time	38.98	184.04	4,580.14	1,197.58	228.68	190.36	419.41	603.05
min time	19.11	29.40	99.05	64.50	16.75	32.34	9.38	175.87
max time	56.75	1,975.71	15,501.44	25,648.42	2,983.09	816.12	4,348.94	2,575.84

Table 1: Experimental results: for each database, the first column (Map) gives results obtained with our submap isomorphism algorithm; the last three columns give results obtained with the subgraph isomorphism algorithms ILF, LAD, and Vflib; the first line (# fail) gives the number of patterns for which the CPU time limit of 1000 seconds has been exceeded for at least one target of the database; the last three lines give the average, minimum and maximum number of seconds spent to find all occurrences of a pattern in all the targets (only for the patterns for which the CPU time limit has not been exceeded).

composed of all possible injective matchings from the set of pattern vertices to the set of target vertices. To reduce the search space, this exhaustive exploration is combined with filtering techniques that aim at removing candidate couples of non matched pattern-target vertices. Vflib, ILF, and LAD consider different filtering techniques, and we refer the reader to [Sol10] for a detailed comparison of them. For Vflib, we have considered the Vflib2 variant, which obtained the best results on our benchmarks. For ILF, we have set the k parameter to 1.

5.3. Experimental results

Experimental settings.. All algorithms have been implemented in C or C++. All our experiments were made on a PC with a 2.26 GHz Intel Xeon E5520 processor. For each database, and for each pattern of the database, we have searched for all occurrences of this pattern in all the targets of the database. This corresponds to $224 \times 224 + 62 \times 800 = 99,776$ runs of Algorithm 3. As the subgraph isomorphism problem is NP-complete, we have limited the CPU time spent to search for all occurrences of a pattern in one target graph to 1000 seconds. Note that when the search for a pattern in one target has exceeded this CPU time limit of 1000 seconds for one algorithm, we have discarded all results for this pattern and this algorithm in table 1.

Comparison of CPU times.. Table 1 gives the CPU time spent to find all occurrences of a pattern in all the targets of the considered database. Our submap algorithm is very effective: on average, it is able to find all occurrences of a pattern in the 224 (resp. 800) targets in 38.98 (resp. 228.68) seconds for the 2D (resp. 3D) database. Actually, the number of darts visited during the search process is usually far below the theoretical bound: in the 2D (resp. 3D) database, the average number of darts visited to find all occurrences of a pattern in one target is 45 (resp. 119) whereas a pattern has $2,175$ (resp. $23,413$) darts on average and a target has $14,509$ (resp. $179,540$) darts on average. This comes from the fact that we consider an optimized version of the algorithm which checks that the matching f is an injection during the traversal, and which stops the traversal as soon as two different pattern darts are matched to a same target dart.

Table 1 also gives results for subgraph isomorphism. For the 2D database, our submap algorithm is 4.7 times as fast as ILF, which is the fastest of the three considered subgraph isomorphism algorithms. Moreover, if ILF has never exceeded the CPU time limit of 1000 seconds for a pattern/target couple, LAD and Vflib have exceeded this limit for 1 and 8 patterns respectively.

For the 3D database, our submap algorithm has never exceeded the CPU time limit whereas it has been exceeded by ILF, LAD, and Vflib for 6, 6, and 40 patterns respectively (among the 62 considered patterns). On average, our submap algorithm is slightly slower than ILF, but this comes from the fact that the 6 unsolved instances are harder instances which increase the average CPU time of our submap algorithm. Indeed, if we do not consider these 6 instances, our submap algorithm solves the 56 remaining instances in 135.58 seconds on average (minimum 16.73 and maximum 770.23). If we only consider the 22 instances solved by Vflib, the average time is 84.98 seconds (minimum 16.73 and maximum 144.95). Actually, the harder instances for our submap algorithm are also the harder ones for the subgraph isomorphism. However, if our algorithm is still able to solve these hard instances rather quickly[6], subgraph isomorphism algorithms are not able to solve them within a reasonable amount of time.

Of course, our submap algorithm does not solve the same problem: It exploits the topology to search for patterns in polynomial time. When ignoring this topology, the problem becomes NP-complete, and it is worth noting here the rather good performances of the three subgraph isomorphism algorithms with regard to the fact that they have exponential time complexities in the worst case together with the fact that graphs have thousands of vertices. In particular, ILF exhibits very good performances on some instances. This comes from the fact that it exploits structural properties to prune the search space. In particular, it checks that, for every pattern node u, there exists at least one target node v such that for every neighbor of u there exists a different neighbor of v which has a greater or equal degree (thus ensuring that every neighbor of u can be matched to a different neighbor of v). If this is not the case, then ILF can conclude that there is no subgraph isomorphism without any enumeration. Actually, this kind of structural heuristic could be very easily integrated in our submap isomorphism algorithm.

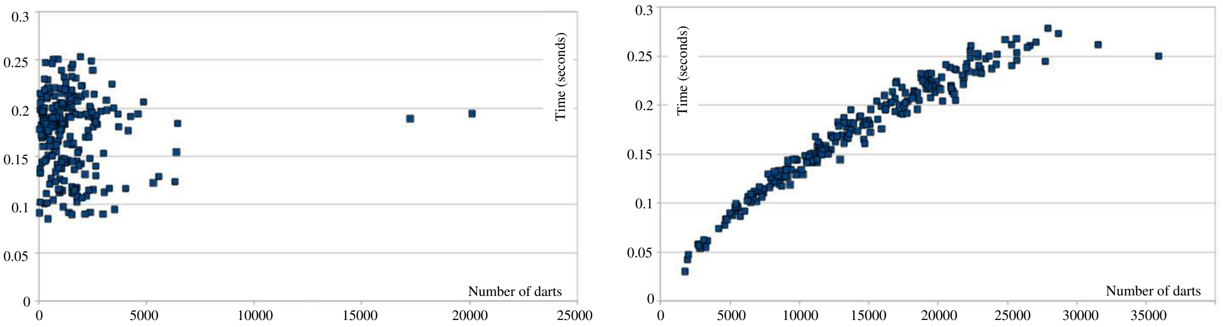
Comparison of the solutions found.. For all 2D patterns and for all 3D patterns but 5, our submap algorithm has found only one solution, corresponding to the target from which it has been extracted. However, 5 3D-patterns have been found in more than one image: These patterns are classical patterns (such as the sphere displayed in Fig. 13(d)) which are building components of many meshes.

Subgraph isomorphism has always found at least as many solutions as submap isomorphism. However, as pointed out in Section 5.2, subgraph isomorphism may find more solutions than submap isomorphism as it ignores the topology. Hence, in the 2D (resp. 3D) database, 148 (resp. 10) patterns have been found more than once in the corresponding targets due to automorphisms. Like for submap isomorphism, 5 3D patterns have been found in more than one target. However, if submap isomorphism never found a same 2D pattern in more than one target, subgraph isomorphism has found 2 2D patterns in more than one target.

Scale up property of the submap algorithm.. Figures 15 and 16 allow us to study the scale up property of the submap algorithm by showing the relation between the average time spent to search for a pattern map in the whole database (both in 2D and 3D) and the number of darts of the pattern (Figures 15(a) and 16(a)), or the target (Figures 15(b) and 16(b)).

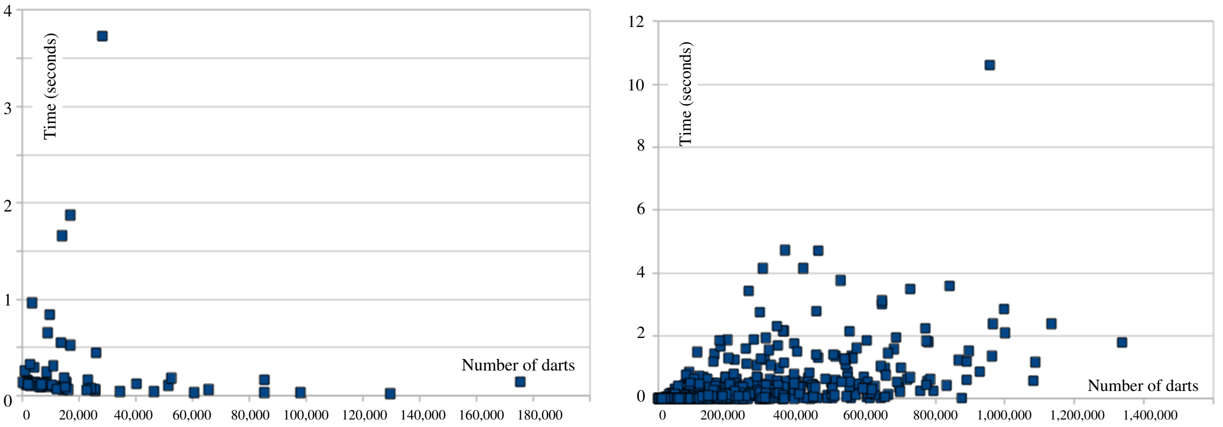
These figures show us that, in practice, the time much more depends on the size of the target than on the size of the pattern. Actually, each traversal is stopped as soon as a pattern dart is matched to more than one target dart so that the number of darts visited during one traversal is usually far below the number of darts in the pattern map (this number is 45 in 2D and 119 in 3D, and it does not really depend on the

[6] The maximum time for searching for all occurrences of a pattern in all the targets of the 3D database is $3,000$ or so seconds, but this database contains 800 target maps so that the CPU time spent for each target is far below the time limit of 1000 seconds.



(a) (b)

Figure 15: CPU time with respect to size for the 2D database. (a) each points corresponds to a different pattern map M; the x-axis gives the number of darts of M and the y-axis gives the average time for finding all occurrences of M in all target maps. (a) each points corresponds to a different target map M'; the x-axis gives the number of darts of M' and the y-axis gives the average time for finding all occurrences of every pattern map in M'.



(a) (b)

Figure 16: CPU time with respect to size for the 3D database. (a) each points corresponds to a different pattern map M; the x-axis gives the number of darts of M and the y-axis gives the average time for finding all occurrences of M in all target maps. (a) each points corresponds to a different target map M'; the x-axis gives the number of darts of M' and the y-axis gives the average time for finding all occurrences of every pattern map in M'.

number of darts of the pattern.). However, we must perform a new traversal for every dart of the target map so that the observed CPU time increases linearly with respect to the size of the target map.

6. Discussion

This paper is a first contribution to the problem of comparing combinatorial maps. We define the map and submap isomorphism problems, which may be used to decide if two maps are equivalent, or if a copy of a map is included in another map, and we describe polynomial algorithms for solving these problems. The proposed definitions and algorithms are generic and hold for any open nD combinatorial maps, which may be used to model nD objects with boundaries. We also introduce planar combinatorial maps, which may be used to model 2D objects that are embedded on a plane such as, for example, images, and we show how to extend our algorithms to decide if two planar maps are isomorphic. Experiments show that this work may be used to search for patterns in 2D images or 3D meshes with very challenging CPU times.

The problem of finding a pattern in an image is an important issue that has often been tackled by modeling images with graphs such as, for example, RAGs. If there exist rather efficient heuristics for solving the graph isomorphism problem[7] [McK81, SS08], this is not the case for the subgraph isomorphism problem which is computationally intractable in the general case (NP-complete), and therefore practically unsolvable for some instances. Hence, if subgraph isomorphism algorithms have been able to solve many instances with reasonable CPU times, they are usually longer than submap isomorphism and, for some instances, they have not been able to find solutions within a reasonable amount of time.

Interestingly, using combinatorial maps to model images allows us to have more relevant results —as combinatorial maps model information that cannot be modelled with classical RAGs such as multi-adjacency or the order of neighbor regions around a region— and these results are computed much more quickly —as the subisomorphism problem becomes polynomial.

Some polynomial algorithms have been proposed to solve particular cases of the subgraph isomorphism problem. In particular, Jiang and Bunke have proposed polynomial algorithms solving the isomorphism [JB99] and subsiomorphism [JB98] decision problems of ordered graphs, *i.e.*, graphs such that, for each vertex, the edges incident to this vertex have a unique order. These algorithms are based on graph traversals and our algorithms for (sub)map isomorphism may be viewed as a generalization of this work to nD open combinatorial maps.

A number of perspectives are being looked into. In particular, (sub)map isomorphism is an exact decision problem which allows us to search for a pattern in a map. We plan to develop error-tolerant methods, which are able to quantify the similarity of two maps by means of the size of their largest common submap, or by means of the cost to transform a map into another by using the edit operations defined in [DL03, BADSM08]. We also plan to use our (sub)isomorphism algorithms to search for patterns that occur frequently in a database of images modelled by combinatorial maps, thus characterizing classes of images. We have proposed in [GDS09] a signature of combinatorial maps which is based on a map traversal similar to the one used in our isomorphism algorithm. This map signature may be used to decide in linear time if a new combinatorial map already belongs to a database of map signatures. We now plan to use this map signature and our submap algorithm in order to find frequent patterns in a database of maps.

Acknowledgement

The authors acknowledge authors of [PABL07], specially Pascal Lienhardt, for all the fruitful discussions about open combinatorial maps. This work was done in the context of project Sattic (Anr grant Blanc07-1_184534). Colin de la Higuera, Jean-Christophe Janodet and Émilie Samuel also acknowledge that this work was supported in part by the IST Programme of the European Community, under the Pascal 2 Network of Excellence, Ist–2006-216886.

[AK89] D. Arques and P. Koch. Modélisation de solides par les pavages. In *Proc. Pixim 89*, pages 47–61, Paris, 1989.

[BADSM08] M. Baba-Ali, G. Damiand, X. Skapin, and D. Marcheix. Insertion and expansion operations for n -dimensional generalized maps. In *Proc. Discrete Geometry for Computer Imagery*, volume 4992 of *LNCS*, pages 141–152, Lyon, France, April 2008. Springer-Verlag.

[BDB97] L. Brun, J.-P. Domenger, and J.-P. Braquelaire. Discrete maps : a framework for region segmentation algorithms. In *Proc. Workshop on Graph-based Representations in Pattern Recognition*, Lyon, april 1997. IAPR-TC15. published in Advances in Computing (Springer).

[BDD01] J.-P. Braquelaire, P. Desbarats, and J.-P. Domenger. 3d split and merge with 3-maps. In *Proc. Workshop on Graph-based Representations in Pattern Recognition*, pages 32–43, Ischia, Italy, may 2001. IAPR-TC15.

[BDDW99] J.P. Braquelaire, P. Desbarats, J.P. Domenger, and C.A. Wüthrich. A topological structuring for aggregates of 3d discrete objects. In *Proc. Workshop on Graph-based Representations in Pattern Recognition*, pages 193–202, Austria, may 1999. IAPR-TC15.

[Bru96] L. Brun. *Segmentation d'images couleur à base Topologique.* Thèse de doctorat, Université Bordeaux I, décembre 1996.

[CFSV04] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1367–1372, 2004.

[7] The theoretical complexity of graph isomorphism is an open question: If it clearly belongs to NP, it has not been proven to be NP-complete.

[CFSV07] D Conte, P. Foggia, C. Sansone, and M. Vento. *Applied Graph Theory in Computer Vision and Pattern Recognition*, chapter How and Why Pattern Recognition and Computer Vision Applications Use Graphs, pages 85–135. Studies in Computational Intelligence. Springer Berlin / Heidelberg, 2007.

[Cor75] R. Cori. Un code pour les graphes planaires et ses applications. In *Astérisque*, volume 27. Soc. Math. de France, Paris, France, 1975.

[Dam08] G. Damiand. Topological model for 3d image representation: Definition and incremental extraction algorithm. *Computer Vision and Image Understanding*, 109(3):260–289, March 2008.

[DBF04] 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.

[DD08] A. Dupas and G. Damiand. First results for 3d image segmentation with topological map. In *Proc. Discrete Geometry for Computer Imagery*, volume 4992 of *LNCS*, pages 507–518, Lyon, France, April 2008. Springer-Verlag.

[DDLHJ$^+$09] 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 *Proc. Workshop on Graph-Based Representation in Pattern Recognition*, volume 5534 of *LNCS*, pages 102–112, Venice, Italy, May 2009. Springer-Verlag.

[DL03] G. Damiand and P. Lienhardt. Removal and contraction for n-dimensional generalized maps. In *Proc. Discrete Geometry for Computer Imagery*, volume 2886 of *LNCS*, pages 408–419, Naples, Italy, November 2003. Springer-Verlag.

[Dom92] J.P. Domenger. *Conception et implémentation du noyau graphique d'un environnement 2D1/2 d'édition d'images discrètes*. Thèse de doctorat, Université Bordeaux I, avril 1992.

[DR02] G. Damiand and P. Resch. Topological map based algorithms for 3d image segmentation. In *Proc. Discrete Geometry for Computer Imagery*, number 2301 in LNCS, pages 220–231, Bordeaux, France, april 2002.

[Edm60] J. Edmonds. A combinatorial representation for polyhedral surfaces. *Notices of the American Mathematical Society*, 7, 1960.

[Fio96] C. Fiorio. A topologically consistent representation for image analysis: the frontiers topological graph. In *Proc. Discrete Geometry for Computer Imagery*, number 1176 in LNCS, pages 151–162, Lyon, France, november 1996.

[GDS09] S. Gosselin, G. Damiand, and C. Solnon. Signatures of combinatorial maps. In *Proc. International Workshop on Combinatorial Image Analysis*, volume 5852 of *LNCS*, pages 370–382, Cancun, Mexico, November 2009. Springer Berlin/Heidelberg.

[GMBM95] T. Geraud, J.F. Mangin, I. Bloch, and H. Maitre. Segmenting internal structures in 3d mr images of the brain by markovian relaxation on a watershed based adjacency graph. In *Proc. IEEE International Conference on Image Processing*, volume 3, pages 548–551, oct 1995.

[JB93] X. Y. Jiang and H. Bunke. An optimal algorithm for extracting the regions of a plane graph. *Pattern Recognition Letters*, 14(7):553–558, 1993.

[JB98] X. Jiang and H. Bunke. Marked subgraph isomorphism of ordered graphs. In *Proc. Advances in Pattern Recognition*, volume 1451 of *LNCS*, pages 122–131, 1998.

[JB99] X. Jiang and H. Bunke. Optimal quadratic-time isomorphism of ordered graphs. *Pattern Recognition*, 32(7):1273–1283, 1999.

[KM95] W.G. Kropatsch and H. Macho. Finding the structure of connected components using dual irregular pyramids. In *Proc. Discrete Geometry for Computer Imagery*, pages 147–158, september 1995.

[Kov89] V.A. Kovalevsky. Finite topology as applied to image analysis. *Computer Vision, Graphics, and Image Processing*, 46:141–161, 1989.

[Lie91] P. Lienhardt. Topological models for boundary representation: a comparison with n-dimensional generalized maps. *Computer-Aided Design*, 23(1):59–82, 1991.

[Lie94] P. Lienhardt. N-dimensional generalized combinatorial maps and cellular quasi-manifolds. *International Journal of Computational Geometry and Applications*, 4(3):275–324, 1994.

[LMV01] J. Llados, E. Marti, and J.J. Villanueva. Symbol recognition by error-tolerant subgraph matching between region adjacency graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):1137–1143, October 2001.

[McK81] B. D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

[PABL07] 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.

[Ros74] A. Rosenfeld. Adjacency in digital pictures. *Information and Control*, 26(1):24–33, 1974.

[Saa94] K. Saarinen. Color image segmentation by a watershed algorithm and region adjacency graph processing. In *Proc. IEEE International Conference on Image Processing*, volume 3, pages 1021–1025, nov 1994.

[SC84] M. Suk and T.H. Cho. An object-detection algorithm based on the region-adjacency graph. *Proceedings of the IEEE*, 72:985–986, 1984.

[Sol10] C. Solnon. Alldifferent-based filtering for subgraph isomorphism. *Artificial Intelligence*, 174(12-13):850–864, 2010.

[Spe91] J.C. Spehner. Merging in maps and in pavings. *Theoretical Computer Science*, 86(2):205–232, September 1991.

[SS08] S. Sorlin and C. Solnon. A parametric filtering algorithm for the graph isomorphism problem. *Constraints*, 13(4):518–537, 2008.

[Tut63] W.T. Tutte. A census of planar maps. *Canad. J. Math.*, 15:249–271, 1963.

[ZDS10] S. Zampelli, Y. Deville, and C. Solnon. Solving subgraph isomorphism problems with constraint programming. *Constraints*, 15(3):327–353, 2010.