Polynomial Algorithm for Submap Isomorphism: Application to searching patterns in images

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

A Polynomial Algorithm for Submap Isomorphism —Application to Searching Patterns in Images—

Guillaume Damiand[1,2], Colin de la Higuera[1,3], Jean-Christophe Janodet[1,3], Émilie Samuel[1,3], and Christine Solnon[1,2⋆ ⋆⋆]

[1] Université de Lyon
[2] Université Lyon 1, LIRIS, UMR5205 CNRS, F-69622, France
{guillaume.damiand,christine.solnon}@liris.cnrs.fr
[3] CNRS UMR5516, F-42023 Laboratoire Hubert Curien,
Université de Saint-Etienne - Jean Monnet
{cdlh,janodet,emilie.samuel}@univ-st-etienne.fr

Abstract. In this paper, we address the problem of searching for a pattern in a plane graph, *i.e.*, a planar drawing of a planar graph. To do that, we propose to model plane graphs with 2-dimensional combinatorial maps, which provide nice data structures for modelling the topology of a subdivision of a plane into nodes, edges and faces. We define submap isomorphism, we give a polynomial algorithm for this problem, and we show how this problem may be used to search for a pattern in a plane graph. First experimental results show the validity of this approach to efficiently search for patterns in images.

1 Introduction

In order to manage the huge image sets that are now available, and more particularly to classify them or search through them, one needs similarity measures. A key point that motivates our work lies in the choice of data structures for modelling images: These structures must be rich enough to describe images in a relevant way, while allowing an efficient exploitation. When images are modelled by vectors of numerical values, similarity is both mathematically well defined and easy to compute. However, images may be poorly modelled with such numerical vectors that cannot express notions such as adjacency or topology.

Graphs allow one to model images by means of, *e.g.*, region adjacency relationships or interest point triangulation. In either case, graph similarity measures

⋆ The authors acknowledge an Anr grant Blanc 07-1_184534: this work was done in the context of project Sattic. This work was partially supported by the IST Programme of the European Community, under the Pascal 2 Network of Excellence, Ist–2006-216886.

⋆⋆ Paper published in Proceedings of 7th Workshop on Graph-Based Representation in Pattern Recognition, LNCS 5534, pp. 102-112, 2009. Thanks to Springer Berlin Heidelberg. The original publication is available at http://dx.doi.org/10.1007/978-3-642-02124-4_11

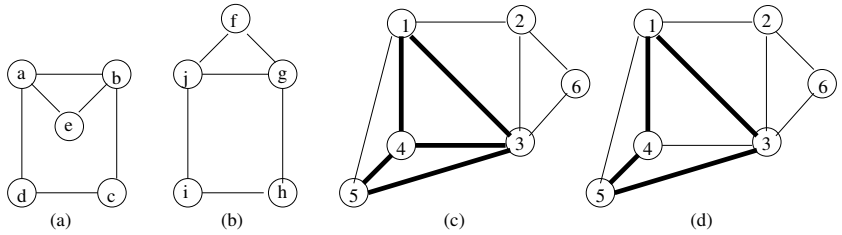


Fig. 1. (a) and (b) are not isomorphic plane graphs; bold edges define a compact plane subgraph in (c), but not in (d).

have been investigated [CFSV04]. These measures often rely on (sub)graph isomorphism —which checks for equivalence or inclusion— or graph edit distances and alignments —which evaluate the cost of transforming a graph into another graph. If there exist rather efficient heuristics for solving the graph isomorphism problem[4] [McK81,SS08], this is not the case for the other measures which are often computationally intractable (NP-hard), and therefore practically unsolvable for large scale graphs. In particular, the best performing approaches for subgraph isomorphism are limited to graphs up to a few thousands of nodes [CFSV01,ZDS+07].

However, when measuring graph similarity, it is overwhelmingly forgotten that graphs actually model images and, therefore, have special features that could be exploited to obtain both more relevant measures and more efficient algorithms. Indeed, these graphs are planar, *i.e.*, they may be drawn in the plane, but even more specifically just one of the possible planar embeddings is relevant as it actually models the image topology, that is, the order in which faces are encountered when turning around a node.

In the case where just one embedding is considered, graphs are called *plane*. *Isomorphism of plane graphs* needs to be defined in order to integrate topological relationships. Let us consider for example the two plane graphs drawn in Fig. 1(a) and 1(b). The underlying graphs are isomorphic, *i.e.*, there exists a bijection between their nodes which preserves edges. However, these plane graphs are not isomorphic since there does not exist a bijection between their nodes which both preserves edges and topological relationships.

Now by considering this, the isomorphism problem becomes simple [Cor75], but the subgraph isomorphism problem is still too hard to be tackled in a systematic way. Yet we may argue that when looking for some pattern in a picture (for example a chimney in a house, or a wheel in a car) we may simplify the problem to that of searching for *compact plane subgraphs* (*i.e.*, subgraphs obtained from a graph by iteratively removing nodes and edges that are incident to the external face). Let us consider for example the plane graphs of Fig. 1.

[4] 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.

The bold edges in Fig. 1(c) constitute a compact plane subgraph. However, the bold edges in Fig. 1(d) do not constitute a compact plane subgraph because edge $(4, 3)$ separates a face of the subgraph into two faces in the original graph.

Contribution and outline of the paper. In this paper, we address the problem of searching for compact subgraphs in a plane graph. To do that, we propose to model plane graphs with 2-dimensional combinatorial maps, which provide nice data structures for modelling the topology of a subdivision of a plane into nodes, edges and faces. We define submap isomorphism, we give a polynomial algorithm for this problem, and we show how this problem may be used to search for a compact graph in a plane graph. Therefore we show that the problem can be solved in this case in polynomial time.

We introduce 2D combinatorial maps in Section 2. A polynomial algorithm for map isomorphism is given in Section 3 and submap isomorphism is studied in Section 4. We relate these results with the case of plane graphs in Section 5, and we give some experimental results that show the validity of this approach on image recognition tasks in Section 6.

2 Combinatorial Maps

A plane graph is a planar graph with a mapping from every node to a point in 2D space. However, in our context the exact coordinates of nodes matter less than their topological organisation, *i.e.*, the order nodes and edges are encountered when turning around faces. This topological organisation is nicely modelled by combinatorial maps [Edm60,Tut63,Cor75].

To model a plane graph with a combinatorial map, each edge of the graph is cut in two halves called *darts*, and two one-to-one mappings are defined onto these darts: the first to link darts belonging to two consecutive edges around a same face, the second to link darts belonging to a same edge.

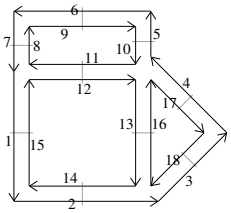
Definition 1. *(2D combinatorial map [Lie91]) A 2D combinatorial map, (or 2-map) is a triplet* $M = (D, \beta_1, \beta_2)$ *where* D *is a finite set of darts;* β_1 *is a* permutation *on* D*,* i.e.*, a one-to-one mapping from* D *to* D*; and* β_2 *is an* involution *on* D*,* i.e.*, a one-to-one mapping from* D *to* D *such that* $\beta_2 = \beta_2^{-1}$*.*

We note β_0 for β_1^{-1}. Two darts i and j such that $i = \beta_k(j)$ are said to be k-sewn. Fig. 2 gives an example of a combinatorial map.

In some cases, it may be useful to allow β_i to be partially defined, thus leading to open combinatorial maps. The intuitive idea is to add a new element ϵ to the set of darts, and to allow darts to be linked with ϵ for β_1 and/or β_2. By definition, $\forall 0 \leq i \leq 2$, $\beta_i(\epsilon) = \epsilon$. Fig. 3 gives an example of open map (see [PABL07] for precise definitions).

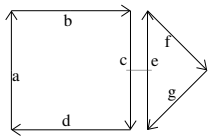
Finally, Def. 2 states that a map is connected if there is a path of sewn darts between every pair of darts.

Definition 2. *(connected map) A combinatorial map* $M = (D, \beta_1, \beta_2)$ *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* $\forall 1 \leq i < k, \exists j_i \in \{0, 1, 2\}, d_{i+1} = \beta_{j_i}(d_i)$*.*



	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

Fig. 2. Combinatorial map example. Darts are represented by numbered black segments. Two darts 1-sewn are drawn consecutively, and two darts 2-sewn are concurrently drawn and in reverse orientation, with little grey segment between the two darts.



	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

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

3 Map Isomorphism

Lienhardt has defined isomorphism between two combinatorial maps as follows.

Definition 3. *(map isomorphism [Lie94]) Two maps* $M = (D, \beta_1, \beta_2)$ *and* $M' = (D', \beta'_1, \beta'_2)$ *are isomorphic if there exists a one-to-one mapping* $f : D \to D'$*, called* isomorphism function*, such that* $\forall d \in D, \forall i \in \{1, 2\}, f(\beta_i(d)) = \beta'_i(f(d))$.

We extend this definition to open maps by adding that $f(\epsilon) = \epsilon$, thus enforcing that, when a dart is linked with ϵ for β_i, then the dart matched to it by f is also linked with ϵ for β'_i.

An algorithm may be derived from this definition in a rather straightforward way, as sketched in [Cor75]. Algorithm 1 describes the basic idea which will be extended in section 4 to submap isomorphism: We first fix a dart $d_0 \in D$; then, for every dart $d'_0 \in D'$, we call Algorithm 2 to build a candidate matching function f and check whether f is an isomorphism function. Algorithm 2 basically performs a traversal of M, starting from d_0 and using β_i to discover new darts from discovered darts. Initially, $f[d_0]$ is set to d'_0 whereas $f[d]$ is set to *nil* for all other darts. Each time a dart $d_i \in D$ is discovered, from another dart $d \in D$ through β_i so that $d_i = \beta_i(d)$, then $f[d_i]$ is set to the dart $d'_i \in D'$ which is linked with $f[d]$ by β'_i.

Complexity issues. Algorithm 2 is in $\mathcal{O}(|D|)$. Indeed, the while loop is iterated $|D|$ times as (i) exactly one dart d is removed from the stack S at each iteration; and (ii) each dart $d \in D$ enters S at most once (d enters S only if $f[d] = nil$, and before entering S, $f[d]$ is set to a dart of D'). In Algorithm 1, the test of line 4 may also be performed in $\mathcal{O}(|D|)$. Hence, the overall time complexity of

Algorithm 1: checkIsomorphism(M, M')

Input: two open connected maps $M = (D, \beta_1, \beta_2)$ and $M' = (D', \beta'_1, \beta'_2)$
Output: returns true iff M and M' are isomorphic

```
1 choose d_0 ∈ D
2 for d'_0 ∈ D' do
3     f ← traverseAndBuildMatching(M, M', d_0, d'_0)
4     if f is a bijection from D ∪ {ε} to D' ∪ {ε} and
      ∀d ∈ D, ∀i ∈ {1,2}, f[β_i(d)] = β'_i(f[d]) then
5         return true
6 return false
```

Algorithm 2: traverseAndBuildMatching(M, M', d_0, d'_0)

Input: two open connected maps $M = (D, \beta_1, \beta_2)$ and $M' = (D', \beta'_1, \beta'_2)$ and an initial couple of darts $(d_0, d'_0) \in D \times D'$
Output: returns an array $f : D \cup \{\epsilon\} \to D' \cup \{\epsilon\}$

```
1  for every dart d ∈ D do: f[d] ← nil
2  f[d_0] ← 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      for i ∈ {0,1,2} do
7          if β_i(d) ≠ ε and f[β_i(d)] = nil then
8              f[β_i(d)] ← β'_i(f[d])
9              push β_i(d) in S
10 f[ε] ← ε
11 return f
```

Algorithm 1 is $\mathcal{O}(|D|^2)$. Note that it may be optimised (without changing its complexity), *e.g.*, by detecting failure while building matchings.

Correction of Algorithm 1. Let us first note that if checkIsomorphism(M, M') returns true, then M and M' are isomorphic as true is returned only if the isomorphism test of line 4 suceeds.

Let us now show that if M and M' are isomorphic then checkIsomorphism(M, M') returns true. If M and M' are isomorphic then there exists an isomorphism function $\varphi : D \to D'$. 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 will be 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 f such that $\forall d \in D, f[d] = \varphi(d)$ so that true will be returned line 5. Claim 1: When pushing a dart d in S, $f[d] = \varphi(d)$. This is true for the push of line 3 as $f[d_0]$ is set to $d'_0 = \varphi(d_0)$ at line 2. This is true for the push of line 9 as $f[\beta_i(d)]$ is set to $\beta'_i(f[d])$ line 8 and $f[d] = \varphi(d)$ (induction hypothesis) and $\varphi(d) = d' \Rightarrow \varphi(\beta_i(d)) = \beta'_i(d')$ (by def-

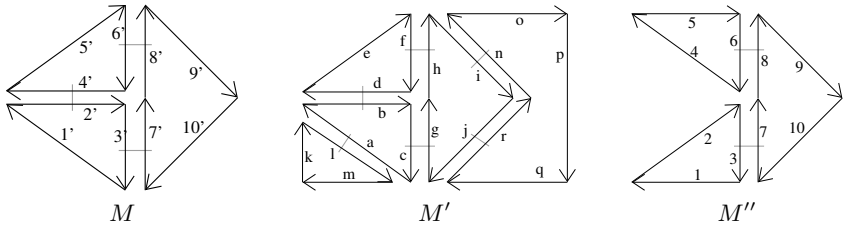


Fig. 4. Submap example. M is a submap of M' as it is obtained from M' by deleting darts k to r. M'' is not a submap of M' as the injection $f : D'' \to D$ wich matches darts 1 to 10 with darts a to j does not verify Def. 4: $\beta_2(2) = \epsilon$ and $f(2) = b$ but $\beta'_2(b) \neq \epsilon$ and $f^{-1}(\beta'_2(b))$ is not empty because $f^{-1}(d) = 4$.

inition of an isomorphism function). Claim 2: Every dart $d \in D$ is pushed once in S. Indeed, M is connected. Hence, there exists at least one path $(d_0, \ldots, d_n)$ such that $d_n = d$ and $\forall k \in [1; n], \exists j_k \in \{0, 1, 2\}, d_k = \beta_{j_k}(d_{k-1})$. 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).

4 Submap Isomorphism

Intuitively, a map M is a submap of M' if M can be obtained from M' by removing some darts. When a dart d is removed, we set $\beta_i(d')$ to ϵ for every dart d' such that $\beta_i(d') = d$.

Definition 4. *(submap) An open combinatorial map $M = (D, \beta_1, \beta_2)$ is isomorphic to a submap of an open map $M' = (D', \beta'_1, \beta'_2)$ if there exists an injection $f : D \cup \{\epsilon\} \to D' \cup \{\epsilon\}$, called a subisomorphism function, such that $f(\epsilon) = \epsilon$ and $\forall d \in D, \forall i \in \{1, 2\}$, if $\beta_i(d) \neq \epsilon$ then $\beta'_i(f(d)) = f(\beta_i(d))$ else either $\beta'_i(f(d)) = \epsilon$ or $f^{-1}(\beta'_i(f(d)))$ is empty.*

This definition derives from the definition of isomorphism. The only modification concerns the case where d is i-sewn with ϵ. In this case, the definition ensures that $f(d)$ is i-sewn either with ϵ, or with a dart d' which is not matched with a dart of M, *i.e.*, such that $f^{-1}(d')$ is empty (see example in Fig. 4).

Note that if M is isomorphic to a submap of M', then M is isomorphic to the map M'' obtained from M' by restricting the set of darts D' to the set of darts $D'' = \{d \in D' | \exists a \in D, f(a) = d\}$.

Algorithm 3 determines if there is a submap isomorphism between two open connected maps. 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. The time complexity of this algorithm is in $\mathcal{O}(|D| \cdot |D'|)$ as traverseAndBuilMatching is called at most $|D'|$ times and its complexity is in $\mathcal{O}(|D|)$. Note that the subisomorphism test may be done in linear time.

Algorithm 3: checkSubIsomorphism(M, M')

Input: two open connected maps $M = (D, \beta_1, \beta_2)$ and $M' = (D', \beta'_1, \beta'_2)$
Output: returns true iff M is isomorphic to a submap of M'

choose $d_0 \in D$
for $d'_0 \in D'$ **do**
  $f \leftarrow$ traverseAndBuildMatching(M, M', d_0, d'_0)
  if *f is an injection from $D \cup \{\epsilon\}$ to $D' \cup \{\epsilon'\}$ and* $\forall d \in D, \forall i \in \{1,2\}, \beta_i(d) \neq \epsilon \Rightarrow f(\beta_i(d)) = \beta'_i(f(d))$ *and* $\forall d \in D, \forall i \in \{1,2\}, \beta_i(d) = \epsilon \Rightarrow \nexists e \in D, f(e) = \beta'_i(f(d))$ **then**
    return true
return false

Concerning correctness, note that 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.

5 From Plane Graphs to Maps

In this section, we show how to transform the problem of finding a compact plane subgraph inside a plane graph into the problem of finding a submap in a map, thus allowing to use our polynomial algorithm.

Let us first precise what we exactly mean by (compact) plane (sub)graph isomorphism. Let us consider two graphs $G_1 = (N_1, E_1)$ and $G_2 = (N_2, E_2)$ that are embedded in planes and let us note $o(i, j)$ the edge which follows edge (i, j) when turning around node i in the clockwise order. We shall say that

- G_1 and G_2 are *plane isomorphic* if there exists a bijection $f : N_1 \to N_2$ which preserves (i) edges, i.e., $\forall (i, j) \in N_1 \times N_1, (i, j) \in E_1 \Leftrightarrow (f(i), f(j)) \in E_2$ and (ii) topology, i.e., $\forall (i, j) \in E_1, o(i, j) = (k, l) \Leftrightarrow o(f(i), f(j)) = (f(k), f(l))$;
- G_1 is a *compact plane subgraph* of G_2 if G_1 is plane isomorphic to a compact subgraph of G_2 which is obtained from G_2 by iteratively removing nodes and edges that are incident to the external face.

Note that the pattern may be a partial subgraph of the target. Let us consider for example Fig. 1c. Edge $(1, 5)$ needs not to belong to the searched pattern, even-though nodes 1 and 5 are matched to nodes of the searched pattern. However, edge $(4, 3)$ must belong to the searched pattern; otherwise it is not compact.

To use submap isomorphism to solve compact plane subgraph isomorphism, we have to transform plane graphs into 2-maps. This is done by associating a face in the map with every face of the graph except the external face. Indeed, a 2-map models a drawing of a graph on a sphere instead of a plane. Hence, none of the faces of a map has a particular status whereas a plane graph has an external (or unbounded) face. Let us consider for example the two graphs in Fig. 1a and Fig. 1b: When embedded in a sphere, they are topologically isomorphic because

one can translate edge (d, c) by turning around the sphere, while this is not possible when these graphs are embedded in a plane. In order to forbid one to turn around the sphere through the external face, graphs are modelled by open 2-maps such that external faces are removed: Only β_2 is opened, and only external faces are missing. Such open 2-maps correspond to topological disks.

Finally, a strong precondition for using our algorithms is that maps must be connected. This implies that the original graphs must also be connected. However, this is not a sufficient condition. One can show that an open 2-map M modelling a plane graph G without its external face is connected if G is connected and if the external face of G is delimited by an elementary cycle.

Hence, submap isomorphism may be used to decide in polynomial time if G_1 is a compact plane subgraph of G_2 provided that (i) G_1 and G_2 are modelled by open 2-maps such that external faces are removed, and (ii) external faces of G_1 and G_2 are delimited by elementary cycles.

This result may be related to [JB98,JB99] which describe polynomial-time algorithms for solving (sub)graph isomorphism of ordered graphs, i.e., graphs in which the edges incident to a vertex are uniquely ordered.

6 Experiments

This section gives some preliminary experimental results that show the validity of our approach. We first show that it allows to find patterns in images, and then we study scale-up properties of our algorithm on plane graphs of growing sizes. Experiments were run on an Intel Core2 Duo CPU at 2.20GHz processor.

6.1 Finding patterns in images

We have considered the MOVI dataset provided by Hancock [LWH03]. This dataset consists of images representing a house surrounded by several objects. We consider two different kinds of plane graphs modelling these images. First, we have segmented them into regions and compute the 2D combinatorial map of the segmented image using the approach described in [DBF04]. Second, we have used the plane graphs provided by Hancock. They correspond to a set of corner points extracted from the images and connected by Delaunay triangulation. These graphs were then converted into 2D combinatorial maps. In both cases, we have extracted patterns from original images, and used our approach to find these patterns.

The left part of Fig. 5 shows an image example, together with its plane graph obtained after segmentation. This graph consists of 2435 nodes, 4057 edges and 1700 faces. The pattern extracted from this image corresponds to the car, composed of 181 nodes, 311 edges and 132 faces. This pattern has been found by our algorithm in the original image, even when submitted to rotation, in 60ms.

The Delaunay graph corresponding to the corner points is shown on the right part of Fig. 5. It has 140 nodes, 404 edges and 266 faces. The graph corresponding

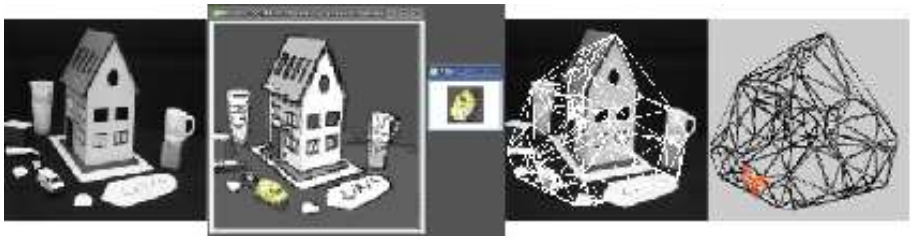


Fig. 5. Finding a car in an image: The original image is on the left. The plane graph obtained after segmentation is on the middle; the car has been extracted and rotated on the right and it has been found in the original image. The graph obtained by Delaunay triangulation and the corresponding combinatorial map are on the right; the car has been extracted and it has been found in the original image.

to the car has 16 nodes, 38 edges and 23 faces. This pattern has been found by our algorithm in the original image in 10ms.

Experiments show that our approach always allows one to find these patterns in the image they have been extracted from.

6.2 Scale up properties

To compare scale-up properties of our approach with those of subgraph isomorphism algorithms, we have performed a second series of experiments: We have randomly generated 3 plane graphs, g_{500}, g_{1000} and g_{5000} which have 500, 1000 and 5000 nodes respectively. These plane graphs are generated by randomly picking n 2D points in the plane, then by computing Delaunay graph of these points. For each plane graph g_i, we have generated 5 sub-graphs, called $sg_{i,k\%}$, which have $k\%$ of the number of nodes of the original graph g_i, where k belongs to $\{5, 10, 20, 33, 50\}$.

Table 1 compares CPU times of our approach with those of Vflib2 [CFSV01], a state-of-the-art approach for solving subgraph isomorphism (we present only results of Vflib2 which is, in our experiments, always faster than Vflib and Ullmann). It shows the interest of using submap isomorphism to solve compact plane subgraph isomorphism. Indeed, if both approaches spend comparable time for the smaller instances, larger instances are much quickly solved by our approach. In particular, instances $(g_{5000}, sg_{5000,k\%})$ are solved in less than one second by our approach whereas it is not solved after one hour of computation by Vflib2 when $k \geq 20$.

It is worth mentionning here that the two approaches actually solve different problems: Our approach searches for compact plane subgraphs whereas Vflib2 searches for induced subgraphs and do not exploit the fact that the graphs are planar. Hence, the number of solutions found may be different: Vflib2 may found subgraphs that are topologically different from the searched pattern; also our approach may found compact plane subgraphs that are partial (see Fig. 1c) whereas Vflib2 only searches for induced subgraphs. For each instance considered

	$sg_{i,5\%}$		$sg_{i,10\%}$		$sg_{i,20\%}$		$sg_{i,33\%}$		$sg_{i,50\%}$	
g_i	vf2	map	vf2	map	vf2	map	vf2	map	vf2	map
g_{500}	0.08	0.07	0.04	0.10	0.47	0.03	0.7	0.02	10.4	0.10
g_{1000}	4.7	0.21	2.54	0.07	0.55	0.05	7.31	0.06	12.7	0.06
g_{5000}	12.3	0.28	156.5	0.31	¿3600.	0.31	¿3600.	0.31	¿3600.	0.31

Table 1. Comparison of scale-up properties of submap and subgraph isomorphism algorithms. Each cell gives the CPU time (in seconds) spent by Vflib2 and our submap algorithm to find all solutions. > 3600 means that Vflib2 had not finished after one hour of computation.

in Table 1, both methods find only one matching, except for $sg_{5000,10\%}$ which is found twice in g_{5000} by vflib2 and once by our approach.

7 Discussion

We have defined submap isomorphism, and we have proposed an associated polynomial algorithm. This algorithm may be used to find compact subgraphs in plane graphs. First experiments on images have shown us that this may be used to efficiently find patterns in images.

These first results open very promising further work. In particular, our approach could be used to solve the subgraph isomorphism problem in polynomial time for classes of planar graphs which admit polynomial numbers of planar embeddings. Also, generalisation to 3 and higher dimensional combinatorial maps is immediate. Hence, our approach could be used to find subsets of compact volumes in 3D images.

Submap isomorphism leads to exact measures, which may be used to check if a pattern belongs to an image. We plan to extend this work to error-tolerant measures such as the largest common submap, which could be used to find the largest subset of edge-connected faces, and map edit distances, which could be used to measure the similarity of maps by means of edit costs.

Finally, more relevant results in the image field could be obtained by integrating geometric information: combinatorial maps may be labelled by features extracted from the modelled image such as, *e.g.*, the shape or the area of a face, the angle between two segments, or the length of a segment. These labels may be used to measure map similarity by quantifying the similarity of labels associated with matched cells.

References

[CFSV01] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. An improved algorithm for matching large graphs. In *3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, pages 149–159, Ischia, Italy, 2001.

[CFSV04] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):265–298, 2004.

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

[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.

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

[JB98] X. Jiang and H. Bunke. Marked subgraph isomorphism of ordered graphs. In *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.

[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.

[LWH03] B. Luo, R. C. Wilson, and E. R. Hancock. Spectral embedding of graphs. *Pattern Recognition*, 36(10):2213–2230, 2003.

[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.

[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.

[ZDS+07] S. Zampelli, Y. Deville, C. Solnon, S. Sorlin, and P. Dupont. Filtering for subgraph isomorphism. In *Proc. 13th Conf. of Principles and Practice of Constraint Programming*, volume 4741 of *Lecture Notes in Computer Science*, pages 728–742. Springer, 2007.