A Linear Algorithm for Exact Pattern Matching in Planar Subdivisions

Pedro Ribeiro de Andrade Neto
André Luiz Pires Guedes

Department of Computer Science
Federal University of Paraná
Centro Politécnico – Jardim das Américas. Curitiba-PR, Brazil 81531-990
{pedro,andre}@inf.ufpr.br

Abstract

Graph sub-isomorphism is a very common approach to solving pattern search problems, but this is a NP-complete problem. This way, it is necessary to invest in research of approximate solutions, or in special cases of the problem. Planar subdivisions can be considered as a special case of graphs, because, in addition to nodes and edges, there is a more rigid topology in relation to the order of the edges, arising to the concept of face. This work presents a linear algorithm for pattern search in planar subdivisions. The presented algorithm is based on a hybrid approach between the dual and the region adjacency graph (RAG) to represent the patterns, saving any additional storage cost. Thus, the patterns are looked over the search subdivision, using a region growing algorithm.

1. Introduction

Graphs with attributes in their vertices and edges are commonly used for representing complex visual structures, in applications of computer vision and pattern search. The use of graphs in these applications is related with the existence of efficient algorithms for solving problems like paths, trees and searches.

Comparison between graphs is a usual strategy for classifying images, for example, in CAD (Computer Aided Design) and robotics. Isomorphism is used to compare graphs, and it consists in a mapping that preserves the adjacency relations between the elements of two graphs.

One of the main problems concerning isomorphism is its time complexity. Exact algorithms require factorial time $O(n!)$ where n is the number of vertices, and graph sub-isomorphism is NP-complete [4]. Therefore, we need specific strategies for each case, with the objective of turning the problem tractable. Solutions using approximate algorithms, as neural networks and genetic algorithms, are very usual, but they can find a local maximum, and do not find the exact solution.

As application for isomorphism we can cite pattern matching in segmented images, for example in architecture drawings. Note that both patterns and segmented images have some special characteristics, that make them different from graphs with vertices and edges:

- There is an order in the associated topology, where edges of one vertex are sorted, following a certain direction;
- Established order in the topology, the concept of face emerges: a cycle without any edges in its internal space;
- Almost all patterns have, in each vertex, at least two edges. For example, in architecture drawings, patterns are doors, tables, etc., and they are structures in this format. Therefore, segmented image vertices that have only one edge can be removed.

Using these information, we can define a *planar subdivision*, a plane partition into regions named *faces*. Faces are limited by line segments, the edges, which ends are vertices. The patterns to be matched also have vertices, edges and faces, then they can be considered planar subdivisions, and its matching is a sub-isomorphism problem.

Region Adjacency Graph (RAG) is a very usual representation in digital image processing. It consists in grouping neighbor pixels with similar characteristics in a same region, that will be a vertex of the graph. One edge connecting two vertices determines that their respective regions have neighbor pixels.

Lládos et al. [3] proposed solving sub-isomorphism in segmented images using RAG's, with a region growing al-

gorithm. Their algorithm is string based, and calculates sub-isomorphism using RAG's. During the loop, the algorithm grows one region in both graphs, in an approximated way, going through all elements of the regions. The authors concluded that their algorithm has exponential time (due to sub-isomorphism NP-completeness), but tests showed a pseudo-polynomial time. The good results were justified by the number of vertices reduced in the RAG generation. They did not propose any data structure to store the graph nor the RAG.

The objective of this article is to investigate the use of RAG's in planar subdivisions sub-isomorphism. We propose one hybrid representation between the well known dual representation and RAG to be used in the sub-isomorphism, getting the advantages of each one of them. This work focuses in the problem's time complexity. Therefore, it is presented a linear algorithm to solve planar subdivisions topological sub-isomorphism. We present a RAG based algorithm similar to Lládos et al [3], but an exact one, and we show that this variation is linear.

This article is written in the following manner. In Section 2 some concepts of planar subdivision are presented. In Section 3 presents the algorithms for doing the isomorphism, for in Section 4 analyze the time complexity and show the implementation results. Finally, Section 5 shows the conclusions of this work.

2. Planar Subdivisions Concepts

A planar subdivision S is a triple (V, E, F), where the three elements are sets, V represents vertices, E edges and F faces. For generalizing, capital letters represent sets, and small letters represent a set's element. Given a set X, its length is represented by $|X|$. One *element* of a planar subdivision can be a vertex, an edge or a face, and its *type* is the set it belongs. Each element of X has an unique identification, one integer from 1 to $|X|$.

In this work, we consider only finite partitions of the plane. It is assumed that no edge passes by any vertex besides their limits, and each vertex belongs to, at least, two edges. It's also considered that each edge separate two, and only two, faces. In this text, frequently we will use the word *subdivision* instead of *planar subdivision*, for simplify. Now we will define some functions to access subdivisions.

$p(x, y)$**:** This function represents the *perimeter* access of an element, defining its topological and geometric limits. Given $x \in X$ and $y \in Y$, $X \neq Y$, and y in x perimeter, this operator returns a value $z \in Y$, the next element from y clockwise around x. Figure 1 shows the perimeter of the three elements.

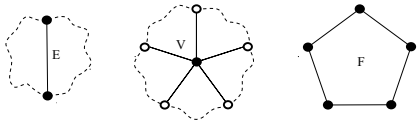


Figure 1. Perimeter of the elements

$n(x, y)$**:** The *neighborhood* access involves elements belonging to a same set, and it is based in sharing perimeters. Two edges are neighbors if they share a vertex and a face. Therefore, edges have exactly four neighbors. Two vertices (faces) are neighbors if they share one edge and, then, two faces (vertices). Note that vertices and faces differ from edges, because they do not have a fixed number of neighbors. Given $x, y \in X$, and y adjacent to x, this primitive returns the next neighbor of x clockwise from y. The neighborhood of the three elements is shown in Figure 2.

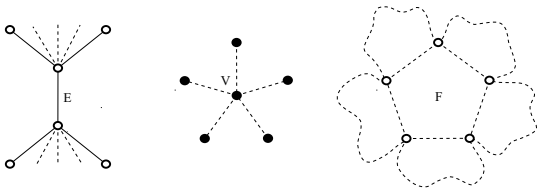


Figure 2. Neighborhood of the elements

$D(S)$**:** Given a planar subdivision $S = (V, E, F)$, its dual representation can be defined. The dual subdivision $D(S) = (F, E', V)$ contains vertices corresponding to the the faces of S, and faces corresponding to its vertices. Two vertices are connected by one edge in $D(S)$ if the corresponding faces in S contains one edge separating them. This way, for each edge joining two vertices in S, there is one splitting the respective faces in the dual, and then $|E| = |E'|$. There is an example of dual in Figure 3.

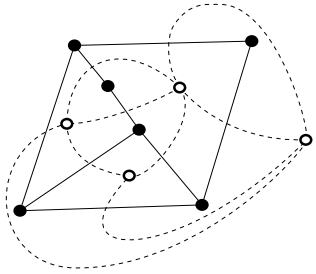


Figure 3. Dual representation

$RAG(S)$**:** given $S = (V, E, F)$, this function returns a subdivision $S' = (V', E', F')$. Each element of V' corresponds to a face of S, except the external face, then $|V'| = |F| - 1$. Two vertices are connected in S' if their respective faces are adjacent in S. For example, if we generate the RAG from Figure 3 we will have a triangle: the external face and its four edges are removed, and there will be only one edge connecting each pair of vertices.

3. Pattern Search Algorithm

Planar subdivisions have specific characteristics if compared to graphs with only vertices and edges. The proposed algorithm is a variation of Lládos et al.'s algorithm [3], noting the differences between an attributed graph, representing an image segments, and a planar subdivision. To facilitate the algorithm's comprehension, it was split in two parts. The first one consists in the definition of a function for region growing, as proposed by Lládos et. al [3], and the second does the matching using region growing.

3.1. RAG versus dual

The algorithm proposed by Lládos et al. [3] gets as input segments of an image, and then the RAG is built, executing one algorithm similar to establish order in subdivisions. The storage of a RAG in the same data structure of the original subdivision is not straightforward. When two or more edges share the same faces perimeter, they have to be merged in only one in the RAG, and the adjacency relations must be conserved. This problem would be simple if all edges that share faces did a path: given an edge that belongs to the path in the subdivision, it would be represented by the halves of the two outer edges of the path. The fact that these edges are not necessarily adjacent, as shown in Figure 4, demonstrates the difficulty of this problem. In this Figure, the two thickest edges are not adjacent, but they will merge in one in the RAG. This way, storing RAG in the same data structure implies in losing performance, because we need algorithmic control over the data structure. Then, the loss of performance is proportional to the number of removed edges when generating the RAG.

Analyzing the quantity of removed edges when generating the RAG, we can see that it is not always significant, because this number depends on the subdivision. For example, in a triangular mesh, this procedure will remove only the edges of the the external face, therefore, it will not reduce considerably the time of the subsequent algorithm. Then, the real advantage of using RAG's for reducing the size of the subdivision is the removing of the external face.

In the case of planar subdivisions, we can consider that the algorithm to establish order in the subdivision was already executed, and then the faces were generated. But they are not necessarily the same of RAG's faces, because in this scenario all faces are preserved.

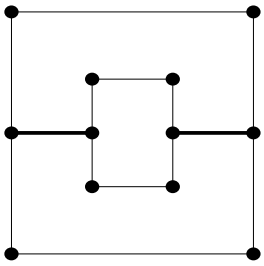


Figure 4. Two non adjacent edges can share the same faces perimeter

As to avoid the storage of the subdivision and its RAG in different structures, and because this representation does not have a real time advantage, we propose a hybrid structure between dual and RAG to represent the subdivision, taking their advantages. This representation has the same characteristics of dual, with two differences: (i) the external face must be marked with a special value, called NULL, and (ii) one edge belongs to the dual iff the two faces of its perimeter do not have NULL value. Therefore, it contains only the internal regions of a planar subdivision. Given a subdivision S, this representation is named $D_R(S)$.

3.2. DCEL (Doubly-Connected-Edge-List)

The algorithm proposed in the next Subsection has as input subdivisions represented using DCEL [5]. DCEL is one of the simplest data structures to represent planar subdivisions, and its name is due to the fact that each edge is connected to only two other edges in the data structure, instead of four, as in *Winged Edge*[1]. A comparative study between these data structures can be found in [2].

This structure is composed by a table with six columns, where four of them contain information ($V1$, $V2$, $F1$ and $F2$) and the other two contain pointers ($P1$ and $P2$). Each line of the table represents one edge, with its two vertices, begin ($V1$) and end ($V2$), and its two faces, left ($F1$) and right ($F2$). As edges need orientation and direction, there are two pointers $P1$ and $P2$. $P1$ stores the first edge counterclockwise from $V1$, and it also is the first edge clockwise from $F1$. Similarly, $P2$ is used as reference for $V2$ and $F2$.

3.3. Region growing

The first part of the algorithm consists in a region growing, as proposed by Lládos et al [3]. This algorithm repre-

Input: $\mathrm{T} = \{\mathrm{T}_E, \mathrm{T}_V, \mathrm{T}_F\}$, e_p and v_p.
Output: If the region growing was successfully done (T is or not valid).

1. $e_s \leftarrow \mathrm{T}_E[e_p]$, $v_s \leftarrow \mathrm{T}_V[v_p]$

   $f_p \leftarrow$ right face from v_p in e_p, $f_s \leftarrow$ left face from v_s in e_s

   $eloop_p \leftarrow e_p$, $eloop_s \leftarrow e_s$

   $of_p \leftarrow p(e_p, f_p)$, $of_s \leftarrow p(e_s, f_s)$

2. if $f_p =$ NULL return true, else if $f_s =$ NULL return false

3. if $\mathrm{T}_F[f_p] \neq$ NULL return $\mathrm{T}_F[f_p] = f_s$, else $\mathrm{T}_F[f_p] \leftarrow f_s$

4. do

   (a) if $\mathrm{T}_V[v_p] \notin \{\text{NULL}, v_s\}$ return false else $\mathrm{T}_V[v_p] \leftarrow v_s$

   (b) if $\mathrm{T}_E[eloop_p] \notin \{\text{NULL}, eloop_s\}$ return false else $\mathrm{T}_E[eloop_p] \leftarrow eloop_s$

   (c) if $of_p \neq$ NULL and $\mathrm{T}_F[of_p] \notin \{\text{NULL}, of_s\}$ return false

   (d) $v_p \leftarrow p(eloop_p, v_p)$, $v_s \leftarrow p(eloop_s, v_s)$

   (e) $eloop_p \leftarrow n(eloop_p, v_p)$, $eloop_s \leftarrow n(eloop_s, v_s)$

   (f) $of_p \leftarrow p(eloop_p, f_p)$, $of_s \leftarrow p(eloop_s, f_s)$

   while $eloop_p \neq e_p$ and $eloop_s \neq e_s$

5. return $eloop_p = e_p$ and $eloop_s = e_s$

Figure 5. Region growing algorithm

sents a function, and it will be used by the matching algorithm. Figure 5 shows its steps.

The region growing algorithm has three arguments. The first one is T, one triple (T_E, T_V, T_F), where the reference of each pattern's element is stored. If $x \in X$ does not have a reference in the search subdivision, $\mathrm{T}_X[x]$ will have NULL value. In this algorithm, T is an argument taken by reference, then, if its values are changed, the variable used as argument will be changed too.

The other two arguments are an edge e_p, indicating from where the face will start, and a vertex v_p, establishing the search direction, that will be from v_p to $p(e_p, v_p)$, circulating the face in the right side of e_p. The index p indicates that the variable refers to the pattern, and the algorithm considers that e_p and v_p have already a reference in T before it starts.

Given T, e_p, and v_p, the algorithm has two objectives. First, to verify if it is possible to realize the isomorphism for all e_p right face adjacent elements, based in a partial isomorphism stored in T. The second objective is to update T with the new references found while circulating the face. The algorithm returns a boolean value, indicating if it was possible to do the region growing, i.e., the required T modifications were done successfully.

First, the algorithm initiates some variables. By convention, f_p represents the right side face of e_p going from v_p to $p(e_p, v_p)$. The variable of_p stores the other face of e_p related to f_p, i.e., $p(e_p, f_p)$. $eloop_p$ represents an edge that initially has the value of a_p, and will be used for circulating the face edges. For each of these variables, and also for e_p and v_p, there is a variable storing the reference of its element in the search subdivision, and they will be used to verifying and updating T. These variables have a index s to indicate that they refer to the search subdivision.

Step 2 verifies some characteristics about the faces to be circulated. If pattern's face is the external face (NULL), then the algorithm ends returning true value. The external face can't be compared with other faces, because it usually corresponds to more then one face in the search subdivision. The same occurs with the search subdivision external face. As it cannot be circulated (there is not a pattern face with its length), if the algorithm find this situation, it stops returning false. The external face is important only when we search for isomorphism. If the search subdivision does not have an external face, the condition that verifies its external face might be removed.

Step 3 checks if T_F already stores some reference to f_p, meaning that this face was circulated before. If this situation occurs, the algorithm returns if the face reference is equal to the value in T_F. If the face was not circulated yet, the

algorithm attributes f_s to $T_F[f_p]$, and then starts to circulate f_p and f_s.

The repetition of this algorithm is represented by step 4. It has six sub-steps, the first half (a,b,c) verify the isomorphism and update T, and the second half (d,e,f) update the variables, going through f_p and f_s. The first three steps verify the T entries, checking if the element already has some reference, and if this value is different from what was found. If the algorithm finds a new value to an element already referenced, then the isomorphism fails. If not, the new value will be the reference to the element, and its slot is changed in T. Note that the algorithm does not change T_F in sub-step c, because face references are updated once for each execution of this algorithm, only for the face that is being circulated, and it is represented by step 3, outside the loop.

If the algorithm does not fail in the verify-and-update-T steps, the next three sub-steps change the variables. Values of $eloop_p$, v_p e of_p (as their references) are updated with the next value clockwise around f_p. The new values of v_p and of_p are calculated using the edge's perimeter, and $eloop_p$ is updated with v_p's neighbor, because going counterclockwise v is the same as to go f_p clockwise. Therefore, this algorithm always go through one face only in one direction, obeying the DCEL requirements.

The loop circulates f_p and f_s, until at least one of them come back to its initial value. If the algorithm finishes the repetition, there is a matching with the two faces' elements, if they were completely gone around, and then they have the same number of adjacent elements. At the end of the algorithm, T is up-to-dated.

One example of region growing is shown in Figure 6. The images demonstrate the variable updating, in its respective steps (hatched lines). If the algorithm continues from the last drawing, there will be two more repetitions, until $eloop_p$ becomes equal to e_p again. Then the algorithm stops and return false, because the two faces have different lengths ($eloop_s \neq e_s$). Note that the algorithm can stop before, if it finds an invalid slot in T.

3.4. Sub-isomorphism algorithm

Once the region growing algorithm is defined, now we will describe the main part of the algorithm, that executes the sub-isomorphism. Given two subdivisions S and P, where P is the pattern to be matched, and S represents the search subdivision, the algorithm uses region growing for calculating the sub-isomorphism. The steps executed by the algorithm are shown in Figure 7.

The first step of this algorithm generates $D_R(P)$. After it, a list $Result$, that will store the sub-isomorphisms found during the search, is initiated with void. Then, there are two possibilities for the edge set of $D_R(P)$, to be checked in steps 2 and 3:

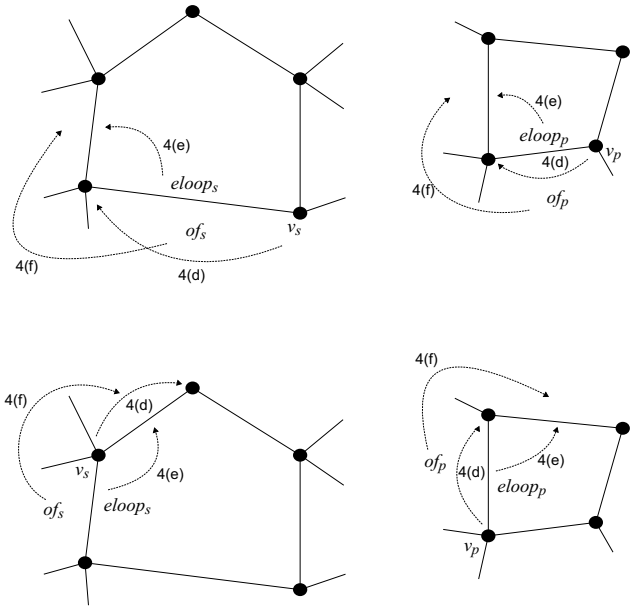


Figure 6. Example of region growing

$|\mathbf{E_{D_R(P)}}| \neq \mathbf{0}$. The pattern has at least two internal faces, and their vertices will be united by one edge in $D_R(P)$. Then, one of $D_R(P)$ edges must be chosen to start the search. This way, the algorithm will execute a breadth-first search, looking for edges in $D_R(P)$, in step 3. The order of edges to have its regions circulated must be the same order found in the search, because, once one given face of an edge has been circulated, the edge neighbors have already reference in T, and can have its faces circulated (note that, when the region growing algorithm is executed for a neighbor of one edge, one of its two faces have already been gone through). Therefore, there is no difference between depth-first or breadth-first search, because both guarantee a valid region growing order.

$|\mathbf{E_{D_R(P)}}| = \mathbf{0}$. P has only two faces, one external and other internal. Then, the growing of only one region is sufficient for solving the problem, and therefore any edge can be chosen.

The first three steps prepare the variables to be used in step 4, that does the pattern search. Each search subdivision edge can be matched with the first edge of L, and these edges can be matched only in the two possible combinations of their vertices. Steps (a) and (b) initiate the last variables before checking the isomorphism, and they are (a) the cleaning of all T positions with NULL and (b) the setting of true value to *isomorphic*, i.e., there is a matching unless it is proven no. Step (c) selects the first edge of L, and in (d) changes the first values in T: the equivalence between e_p

Input: One pattern $P = (V_P, E_P, F_P)$ and one search subdivision $S = (V_S, E_S, F_S)$
Output: One list $Result$ with the sub-isomorphisms found

1. generate $D_R(P)$, $Result \leftarrow$ void
2. if $|E_{D_R(P)}| = 0$, initiate a list L with an edge $e \in E_P$, else choose $e \in E_{D_R(P)}$ to initiate the list
3. if $|E_{D_R(P)}| \neq 0$, execute a breadth-first search algorithm, from e in $D_R(P)$, pushing all found edges in the back of L
4. for each edge $e_s \in E_S$ and for $v \in \{V1, V2\}$ do
   (a) initiate each position of T with NULL
   (b) isomorphic $\leftarrow$ true
   (c) $e_p \leftarrow$ top of L
   (d) $\mathrm{T}_E[e_p] \leftarrow e_s$, $\mathrm{T}_V[e_p[v]] \leftarrow e_s[V1]$, $\mathrm{T}_V[p(e_p, v)] \leftarrow e_s[V2]$
   (e) while not reach the back of L and isomorphic repeat
      i. if not grow region(T, $e_p, e_p[V1]$) isomorphic $\leftarrow$ false
      ii. if not grow region(T, $e_p, e_p[V2]$) isomorphic $\leftarrow$ false
      iii. $a_p \leftarrow$ next of L
   (f) if isomorphic insert T in $Result$
5. return $Result$

Figure 7. Algorithm for planar subdivisions sub-isomorphism

and e_s, and one vertex combination. Therefore, step 4 must be repeated $2|E_S|$ times.

In step (e), e_p goes through L, and the region growing function is called for both e_p faces. If any region growing fails, there is not an isomorphism using the initial configuration (e_s and one vertices combination). But, if all regions are successfully gone through, the isomorphism exists, and T is pushed into $Result$. Finally, at step 5, $Result$ is returned with all sub-isomorphisms found.

Figure 8 shows an example of executing step 4(e) of the second algorithm. In this case, there are five ($|E_{D_R(P)}|$) repetitions, and let us suppose that L has the sequence $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$. The region growing algorithm is called ten times, but only in five the regions are really circulated, and they are represented by the thick lines. Hatched lines indicate that the region growing algorithm was done before in this face. Therefore, once the first edge (a, in this case) is matched in the search subdivision, there is not any combinatorial factor, because as the structure is closed, there are only verifications to be done, and it depends on the pattern's size, but not on the search subdivision size.

Note that, in this Figure, there is another matching using the same initial edge, and it occurs if its vertexs are traded. This justifies the need of two repetitions for each edge of the search subdivision.

4. Algorithm Analysis

To validate the pattern search algorithm, we analyzed its time complexity and implementation. They are described in this Section.

4.1. Time Complexity

For analyzing the algorithm complexity, first we will check the second part, and then the first. In the second part of the algorithm, the loop at step 4 requires more processing time, because any pattern alone processing would take less time then $O(|E_S|)$. In this step, there are two code stretches that requires more time. The first one is step (a), that initializes all T positions with NULL, and consumes time equals to its size, that is $|V_P| + |F_P| + |E_P| = 2|E_P|$. The second stretch is the step (e), that calls region growing for the faces in the perimeter of each edges belonging to L. So the time complexity of this step depends on the region growing time.

Each region growing algorithm calling circulates the edges of two faces, one of the pattern and other of the search subdivision, simultaneously, until one of them reach its initial value again. As DCEL need constant time for all operations used in this algorithm, the region growing function has time equals to the minor perimeter. Then, we can consider

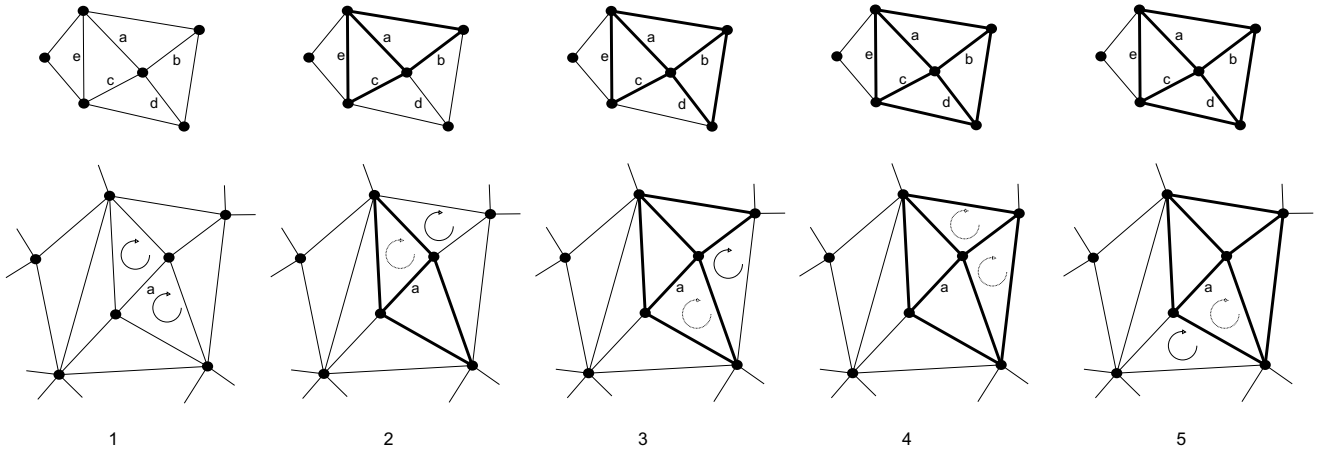


Figure 8. Example of the sub-isomorphism algorithm

that this time is limited by the pattern's face perimeter.

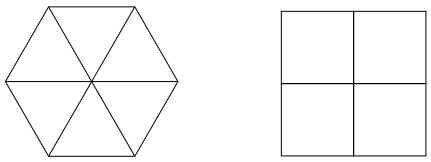
Note that each pattern face is circulated only one time, because the region growing algorithm verifies if the face was circulated before, trying to avoid the loop. As each face is circulated just once, then the algorithm goes through each pattern edge at maximum two times. This way, step 4(e) of the second algorithm needs time limited by $2|E_P|$. So, for each repetition of step 4 of the sub-isomorphism algorithm it is consumed time $4|E_P| = O(|E_P|)$.

As the sub-isomorphism algorithm does two repetitions for each edge of the search subdivision, the total time is $2|E_P| \times O(|E_S|) = O(|E_P||E_S|)$. The pattern can be considered very small, and therefore we can ensure that the algorithm has linear time. In the isomorphism case, as $(|E_P| = |E_S|)$, the algorithm needs time $O(|E_S|^2)$.

4.2. Implementation and tests

The algorithm was implemented using C++, and, to generate test subdivisions, we used the program *triangle* [6]. It can build triangular meshes given an evolving polygon and a resolution. From these meshes, we randomly removed some edge, but only if both vertices have at least three edges, keeping the data structure closed. The probability of removing one edge was 15%. This way, the generated subdivisions are always evolved in a rectangle, but it is not a requirement of the algorithm, because it works with subdivisions in surfaces too.

For test, we used a Debian/Linux Pentium IV 2.4GHz with 512Mb of RAM. Figure 9 shows two examples of patterns used in the tests, and in Figure 10 we have the results of matching these patterns in a random subdivision. Figure 10(a) has ten patterns, and two of them shares five edges. Figure 10(b) has only two matches, because the generated subdivision was from a triangle mesh, and then different patterns are more difficult to be found.



(a) (b)

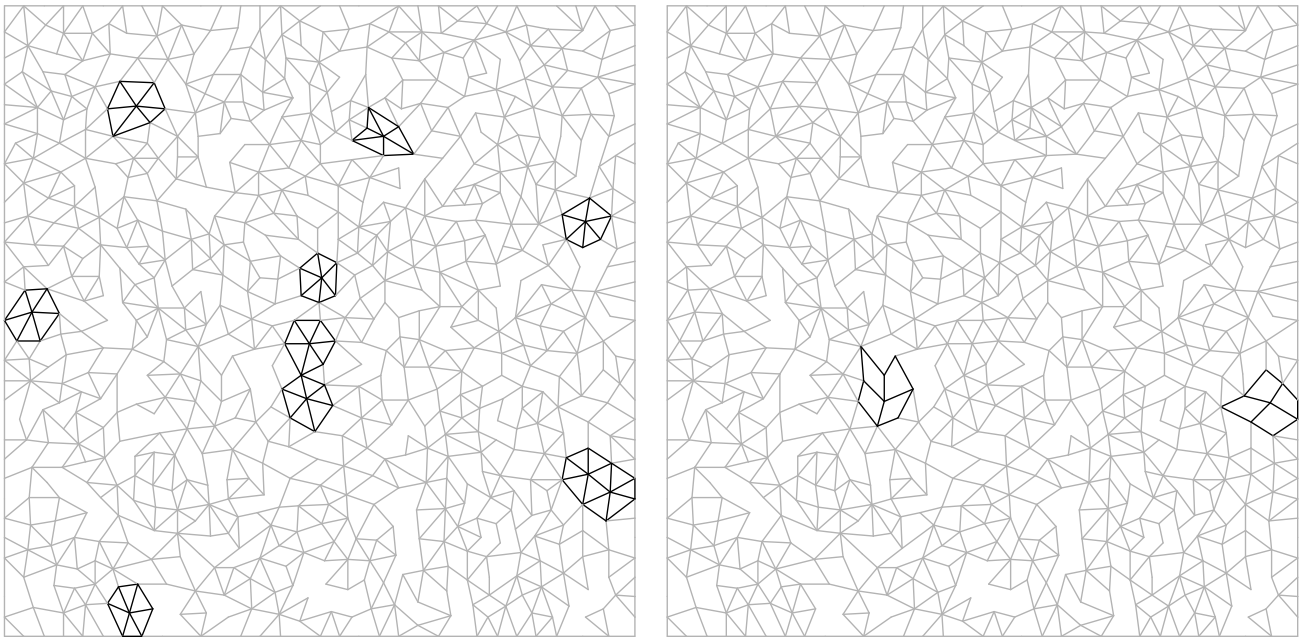
Figure 9. Examples of patterns

The algorithm was tested using subdivisions of size from 1.500 to 1.500.000 edges, and the times to search the pattern in Figure 9(a) are shown in Figure 11. As we can see, the algorithm has linear growing, proving the analysis.

5. Conclusions

Using RAG's, as proposed by Lládos et al. [3], not always reduce significantly the size of the subdivision, therefore, this representation do not reduce the time complexity of the associated algorithm. In this paper, we proposed a hybrid representation between RAG and dual, in the way that it can be stored in the same data structure of the subdivision.

The presented algorithm does planar subdivisions sub-isomorphism based in edges matching, vertices and faces are used only to verify. There are $2|A|$ possibilities for matching edges, and, as the faces keep the data structure united, the algorithm needs only to check if the other edges matches the pattern conditions. This way, the time to verify



(a) Result of searching the pattern of Figure 9(a)

(b) Result of searching the pattern of Figure 9(b)

Figure 10. Searching results

the sub-isomorphism, given a pair of edges to be matched, is equal to the pattern size. As the pattern has very small size, linear time is sufficient to solve the problem. The algorithm was implemented, and the results prove the time complexity.

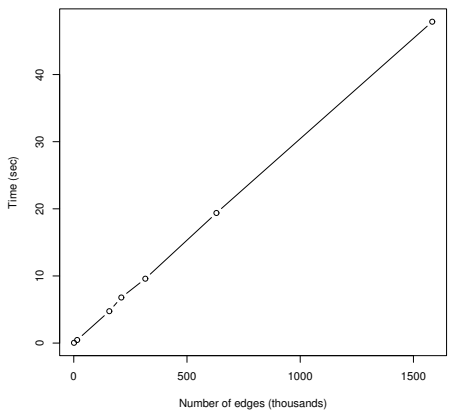


Figure 11. Pattern search times

One deeper study of Lládos et al. work [3] can prove that their algorithm has polynomial time, as shown in their tests, not by the reduction of the edges number, as justified by them, but indeed because the data structure is united by the existence of faces. Two improvements can be done in their algorithm: the first is to use the hybrid representation here presented, saving space, and the second is to execute a search algorithm in the pattern, before the matching, to find the order of the region growing callings.

References

[1] B. G. Baumgart. A polyhedron representation for computer vision. *AFIPS National Computer Conference*, pages 589–596, 1975.

[2] P. R. de Andrade Neto. Busca de padrões em subdivisões planares. Master's thesis, PPGINF – UFPR, Curitiba – PR, 2004.

[3] J. Lladós, E. Martí, 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, Oct. 2001.

[4] K. Mehlhorn. *Graph Algorithms and NP-Completeness*. Springer-Verlag, 1984.

[5] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[6] J. R. Shewchuk. Triangle: Engineering a 2D quality mesh generator and delaunay triangulator. *First Workshop on Applied Computational Geometry*, pages 124–133, May 1996.