Methods for Common Subgraph Approximation

Simone Marini

Rapporto Tecnico IMATI 23/04

Methods for Common Subgraph Approximation

Simone Marini

Rapporto Tecnico IMATI 23/04

Abstract

The contribution presented in this report is aimed at defining a framework for expressing the optimal algorithm for the computation of the maximal common subgraph in a formalization which makes it straightforward usable for plugging heuristics in it, in order to achieving different approximations of the optimal solution according to the specific case.

Introduction

A large class of structural shape descriptors can be easily encoded as directed, a-cyclic and attributed graphs, thus the problem of comparing structural descriptors is approached as a graph matching problem. The techniques used for graph comparison have an exponential computational complexity and it is therefore necessary to define an algorithmic approximation of the optimal solution.

The methods for structural descriptors comparison, commonly used in the computer graphics community, consist of heuristic graph matching algorithms for specific application tasks, while it is lacking a general approach suitable for incorporating different heuristics applicable in different application tasks. The contribution presented in this report is aimed at defining a framework for expressing the optimal algorithm for the computation of the maximal common subgraph in a formalization which makes it straightforward usable for plugging heuristics in it, in order to achieving different approximations of the optimal solution according to the specific case.

Skeletons and Reeb graphs provide an efficient encoding of the structure of the object, where this encoding can be easily represented as a graph (see definition 1). This allow the use of graph matching methodologies in order to compare structural descriptors.

To be used for shape matching purposes, a structural descriptor should be independent of object position, rotation and scaling. Usually skeletons satisfy these requirements, while for the Reeb graph the choice depends on the mapping function as discussed in [BMMP03]. As proposed in [SSGD03, BMM+03], both skeletons and a Reeb graphs may be represented as a-cyclic, directed graphs. However, the skeletal structure requires a number of simplification steps and artifacts, which might alter the topology of the signature, while for the Reeb graph there is a strict relationship between the object topology and the graph structure. Therefore, in the rest of the report, the Reeb graph structural shape descriptor will be used for the experiments.

By the assumption that the Reeb graph representation is a directed a-cyclic graph that encodes the salient shape features and the most significant spatial relations between them, the approach proposed in this report is to find a mapping function between the structural parts of two Reeb graphs. This is achieved through the construction of a common subgraph 6 between two input graphs. This common subgraph should highlight how much the two shapes overlap.

1 Problem Statement

Beside the topological information stored in the graph structure, also attributes associated to edges and nodes, and the information deduced by the directions of the edges concurs to the construction of the mapping function between the structural parts of the two Reeb graph.

The following theorem assures that the Reeb graph is directed and a-cyclic.

Theorem 1 *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mu^{\mathcal{V}}, \mu^{\mathcal{E}})$ be a Reeb graph, it is directed, a-cyclic and attributed graph.*

Proof 1

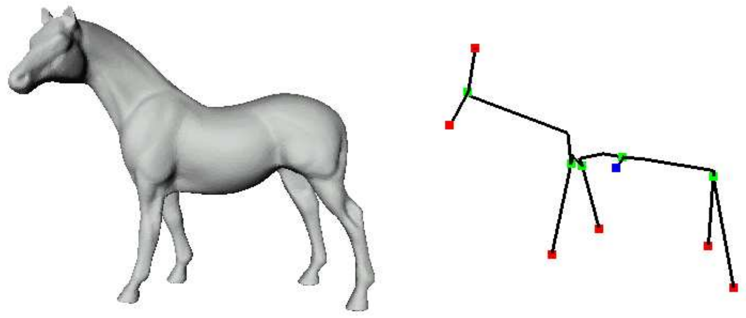
- Directed: *Let e be an edge of $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mu^{\mathcal{V}}, \mu^{\mathcal{E}})$, that is $e \in \mathcal{E}$. From the definition of Reeb Graph it follows that the function f cannot be constant along any edge e; therefore, denoting v_1 and v_2 the nodes of e, the relation $f(v_1) \neq f(v_2)$ holds. Then, the edge e may be oriented according to the monotonicity of f along that: the edge e $e = (v_1, v_2)$ (resp. $e = (v_2, v_1)$) is directed from v_1 to v_2 if $f(v_1) < f(v_2)$ (resp. $f(v_2) < f(v_1)$).*
- Acyclic: *The monotonicity of f along each edge $e \in \mathcal{E}$ implies that $\mathcal{G}$is acyclic. In fact, let P be a path (see definition 3) with starting and ending point v_1. Therefore, there would exist a*

sequence of nodes $v_1, v_2, \ldots, v_n, v_1$ *and edges* $e_1, e_2, \ldots, e_n$ *such that* $e_1 = (v_1, v_2), \ldots, e_n = (v_n, v_1)$. *This would imply the following sequence of inequalities:* $f(v_1) < f(v_2) < \ldots < f(v_1)$, *which is clearly impossible from the definition of Reeb Graph.*

- Attributed: *Attributes may be easily associated to* $\mathcal{G}$*by considering either the geometric properties of the part of the object described by nodes and edges, or shape attributes such as texture, color, etc. While their choice may be questionable, the existence of such attributes, both for nodes and edges, is always possible.*

In particular have to be observed that, since the Reeb graph is directed, each node identifies a subgraph S, where V_S contains the node v itself and all nodes for which the node is an ancestor.

The mapping function between the structural parts of two Reeb graph can be obtained by computing a bijective function between nodes and edges of the two graphs. As shown in appendix 5 there are different way to build such function: for example the graph isomorphism, subgraph isomorphism, common subgraph and error tolerant graph isomorphism. The existence of a graph isomorphism (see definition 4) implies that the graphs must be equivalent. This is a too strong condition that can not be satisfied for the evaluation of shape similarity, in fact, due to the capability of a descriptor to capture the salient features of an object, two similar object (similar and not identical) are described/represented with two similar and not identical shape descriptors. In figure 1 the models of a wolf and a horse are shown together with their Reeb graphs. The two models are similar but due to small morphological differences the two structural descriptors are slightly different: the two subgraphs related to the heads are different because the mouth of the wolf is open. The two front legs of the horse are connected to two different branching nodes, while the two front legs of the wolf are joined to the same branching node. The strong requirements of the graph isomorphism can be relaxed using a weaker similarity measure based on the subgraph isomorphism (definition 5) and the notion of common subgraph (CS) (6). From an intuitive point of view, comparing two structural descriptors means constructing the most suitable common sub-graph: the wider the common subgraph is, the more the two structural descriptors are similar. The word "suitable" has been intentionally used because more than one common subgraph can be defined. For example in figure 2 two common subgraphs are shown for the graphs representing the horse and the wolf of figure 1 respectively. In figure 2(a) is represented the maximum common subgraph (MCS) as defined in 7, while in figure 2(b) the common subgraph represented is smaller than the MCS but more


(a)

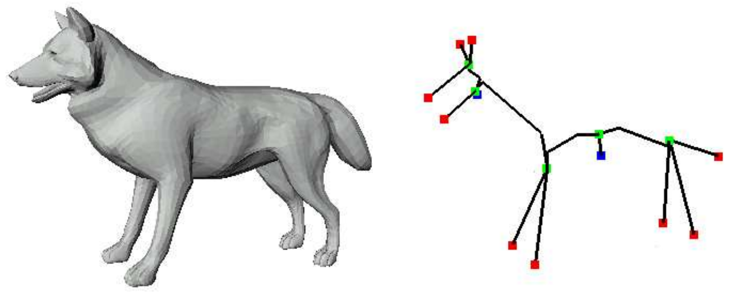

(b)

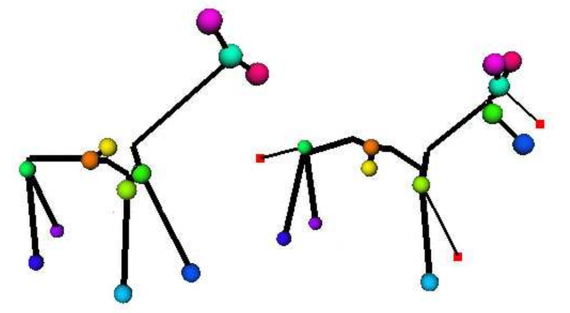
Figure 1: Horse (a) and wolf (b) models together with their Reeb graphs based on the distance from the barycentre function.

meaningful, because semantically equivalent sub-parts of the object are correctly recognized and mapped together.

Unfortunately, the construction of the MCS is a well known NP-complete problem, thus, its exact computation is time consuming, when the shape descriptors are composed by a large numbers of nodes and edges. Therefore, several strategies and heuristic assumptions have been adopted to simplify this problem [BMM$^+$03]. In the following sections of this report will be shown an algorithm for the computation of the maximum common subgraph (MCS) between two directed and a-cyclic graphs and several heuristic techniques Will be discussed.

2 Algorithm Description

A naive algorithm for the computation of the $\mathcal{MCS}_{\mathcal{G}_1,\mathcal{G}_2}$ between the two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ is shown in figure 3. This algorithm is optimal in the sense that it returns the correct result, but it has an exponential computational complexity. For this reason, heuristic techniques have to be considered in many applications in order to approximate the


(a)

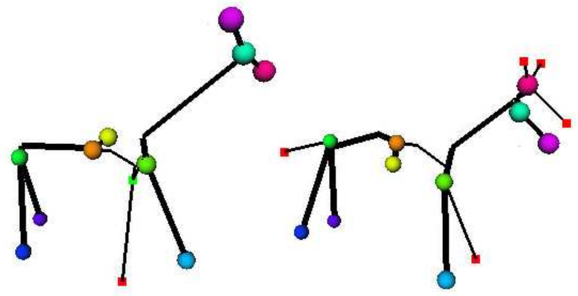

(b)

Figure 2: Two common subgraphs between the horse and wolf graphs. Nodes having the same colors are mapped together. Thick edges represent edge mapping

1. enumerate all the possible mappings m among $\mathcal{V}_1$ and $\mathcal{V}_2$;
2. search those mappings m that satisfy the definition of maximum common subgraph. Of course, m is not necessarily unique.

Figure 3: The naive algorithm

MCSs of two input graphs.

Even if the described algorithm is very simple, it is not easy to define heuristic techniques based on the attributes of edges and nodes, or on reasoning about the graph structure. Also, it is not easy to devise an approximation which makes the structural shape matching robust to structural noise in the graphs. With the aim to introduce such techniques the point 2 of the algorithm can be modified as in the figure 4.

The point 2 of the new algorithm expands the input mapping m as

1. enumerate all the possible mappings m among $\mathcal{V}_1$ and $\mathcal{V}_2$;
2. for each listed mapping m, compute the common subgraphs of $\mathcal{G}_1$ and $\mathcal{G}_2$ obtainable by expanding m.

Figure 4: The new algorithm obtained modifying the naive algorithm shown in figure 3

much as possible while respecting the definition of common subgraph.

Since the first point of both algorithms enumerates all the possible mappings, proving that the second formulation is correct reduces to proving that the expansion always produces a correct common subgraph and that it does not alter the structure of a maximum common subgraph, should this be given as input to the expansion process.

2.1 Pseudo code

In this section, the alternative algorithm is described by explaining the steps of its pseudo code.

The data structures involved in the algorithm are:

- `G_1` $= \mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1, \mu^{\mathcal{V}_1}, \mu^{\mathcal{E}_1})$ and `G_2` $= \mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2, \mu^{\mathcal{V}_2}, \mu^{\mathcal{E}_2})$, are the two input graphs of the algorithm;
- `M`: is a set of node pairs (v_1, v_2), where $v_1 \in \mathcal{V}_1$ and $v_2 \in \mathcal{V}_2$;
- `CS`: is a common subgraph of `G_1` and `G_2`. Each element of `CS` is a four-tuple (v_1, v_2, e_1, e_2), where $v_1 \in \mathcal{V}_1$, $v_2 \in \mathcal{V}_2$ and $e_1 \in \mathcal{E}_1$, $e_2 \in \mathcal{E}_2$. The node pair (v_1, v_2) and the edge pair (e_1, e_2) of each four-tuple, are the node-mapping and the edge mapping of the common subgraph.
- `MCS`: is the set of all the common subgraphs computed by the algorithm;
- `CANDIDATES`: is the set of four-tuples (v_1, v_2, e_1, e_2)candidate to became an element of `CS`.
- `CS_SET`: is a set of pairs (`CS`, `CANDIDATES`), where each pair represents a common subgraph `CS` and the set `CANDIDATES` is the set of possible node and edge mappings candidate to expand `CS`.

The main procedure of the algorithm is `MCS()`, shown in the listing 1. It enumerates the set of initial mappings $\mathcal{M}_{\mathcal{G}_1,\mathcal{G}_2}$ among the nodes of $\mathcal{G}_1$ and $\mathcal{G}_2$. For each $m \in \mathcal{M}_{\mathcal{G}_1,\mathcal{G}_2}$ it generates the set `CS` of the common subgraphs obtained expanding m. `CS` is computed by the function `CS_from_Mapping(`m`)` applied on the set of initial mappings

Listing 1: The main procedure

```
MCS(G_1, G_2)
{
    M   = empty_set
    CS  = empty_set
    MCS = empty_set

    M = Mappings_Set(G_1, G_2)

    for each m in M{
        CS = CS_from_Mapping(m)
        Add(CS, MCS)
    }
    return Max(MCS)
}
```

m. Obviously, each m may generate more than one common subgraph and the expansion procedure `CS_from_Mapping(`m`)` produces all of them. Finally, the MCSs are obtained selecting the common subgraphs with the largest number of nodes.

The procedure `CS_from_Mapping()` (listing 2) expands the set of node pairs in `m` in order to generate a set of common subgraphs of the two input graphs. First `Init_Candidates()` (listing 3) transforms the set of node pairs in m in a set of candidates, where each pair become a candidate for growing the common subgraph. At the line 8 of the procedure `Init_Candidates()`, the two edges e_1 and e_2 are set to `NULL`, because it make no sense to account for the edge mapping for the initial candidates generated from `m`. After the initialization of `CANDIDATES`, the set `CS_SET` (line 10 of the listing 2) contains only one pair (`CS`, `CANDIDATES`) with `CS` $= \phi$, since no node and edge mappings has been yet produced.

The main loop of the procedure `CS_from_Mapping()` (starting at line 12 of the listing 2) aims at adding new elements to `CS` by checking if the nodes of the candidate element can be mapped correctly. This loop iterates through all the elements (`CS`, `CANDIDATES`) of `CS_SET`.

Given the pair (`CS`, `CANDIDATES`), the secondary loop starting at line 13, iterates through the candidate elements `(v_1, v_2,` $\star$, $\star$`)` until `CANDIDATES`become empty. The symbols $\star$ replace the edges of the candidate element because it is not necessary at this moment. The candidate element is extracted through `Pop()`, that remove it from

Listing 2: Expansion of the initial mapping

```
1  CS_from_Mapping(m)
2  {
     CS          = empty_set
4    CANDIDATES  = empty_set
     CS_SET      = empty_set
6    v_1         = empty_node
     v_2         = empty_node
8
     Init_Candidates(CANDIDATES, m)
10   Add((CS, CANDIDATES), CS_SET)

12   for each (CS, CANDIDATES) in CS_SET{
       while not Empty(CANDIDATES){
14       (v_1,v_2,*,*) = Pop(CANDIDATES)
         if(Mapped(v_1) or Mapped(v_2))
16         Resolv_Conflict((v_1,v_2,*,*),
                           (CS,CANDIDATES), CS_SET)
18       else{
           Add((v_1,v_2,*,*), CS)
20         Update((v_1,v_2,*,*), CANDIDATES)
         }
22     }
     }
24   return Max(CS_SET)
   }
```

CANDIDATES. For each node pair (v_1, v_2) line 15 checks if the nodes of the candidate extracted are already mapped or not: if not, that is if they do not belong to any four-tuple in CS, they are added to CS, and new candidates are generated by Update() (listing 4); otherwise, the Resolve_Conflict() (listing 5) is called to handle the situation.

The procedure Update() generates new candidate elements from the four-tuple (v_1, v_2, *, *). A new four-tuple is obtained for each out coming edge e_1 from v_1 and e_2 from v_2, where the new candidate element (u, v, e_1, e_2) is generated exploiting the direction of the edge e_1 = (v_1, u) and e_2 = (v_2, v).

The conflicts are solved by the procedure Resolve_Conflict() (listing 5) by forking the expansion of the current CS into two com-

Listing 3: Transforms the in ital set of node mappings into the initial candidates

```
Init_Candidates(CANDIDATES, m)
 {
   CANDIDATES = empty_set
    v_1       = empty_node
    v_2       = empty_node

    for each (v_1, v_2) in m
       Add((v_1, v_2, NULL, NULL), CANDIDATES)
 }
```

Listing 4: `Update` adds new elements to `CANDIDATES`by processing the four-tuple `p`.

```
Update((v_1, v_2, *, *), CANDIDATES)
{
  u   = empty_node
  v   = empty_node
  e_1 = empty_edge
  e_2 = empty_edge

  for each edge e_1 out coming from v_1 {
     for each edge e_2 out coming from v_2 {
        u = Opposite(v_1, e_1)
        v = Opposite(v_2, e_2)
        Add((u,v, e_1, e_2), CANDIDATES)
     }
  }
}
```

mon subgraphs where the new one is obtained by eliminating the pairs of nodes responsible of the conflict and the subsequent part of the common subgraph through the procedure `Reset()` (listing 6). The lines 10, 12 and 14 controls which nodes raises the conflict, and as consequence `Reset()` is called. After `Reset()` finished, the two sets `NEW_CS` and `NEW_CANDIDATES` are produced, each one representing a new common subgraph and a new set of candidates, respectively. Therefore the four-tuple `(v_1,v_2, *, *)` is added to `NEW_CS`, the new set of candidates `NEW_CANDIDATES` is updated and finally the new pair `(NEW_CS, NEW_CANDIDATES)` is added to `CS_SET`.

Listing 5: The procedure `Resolve_Conflict()` is called to solve a conflict

```
Resolv_Conflict((v_1,v_2, *, *),
                     (CS, CANDIDATES), CS_SET)
{
 NEW_CS             = empty_set
 NEW_CANDIDATES = empty_set
 CS_SET             = empty_set

 NEW_CS             = CS
 NEW_CANDIDATES = CANDIDATES


  if(Mapped(v_1) and Mapped(v_2))
    Reset(v_1, v_2, NEW_CS, NEW_CANDIDATES)
  else if(Mapped(v_1) and not Mapped(v_2))
          Reset(v_1, NULL, NEW_CS, NEW_CANDIDATES)
        else if(not Mapped(v_1) and Mapped(v_2))
                Reset(NULL, v_2, NEW_CS, NEW_CANDIDATES)

 Add((v_1,v_2, *, *), NEW_CS)
 Update((v_1,v_2, *, *), NEW_CANDIDATES)
 Add((NEW_CS, NEW_CANDIDATES), CS_SET)
}
}
```

The procedure `Reset()` (listing 6) updates both the sets `CS` and `CANDIDATES` by deleting all the nodes involved in the conflict. The statement `Delete(v_1,CS)` eliminates all the four-tuples containing the node `v_1` from the common subgraph `CS`, and the loop at line 7 iterates among all the nodes `v` of `CS`, connected through a directed path in `CS`to `v_1`. All the four-tuples of `CS` containing such `v` are deleted at line 8. The same considerations hold for the loops starting at the lines 9, 14 and 16.

To better understand how the `Resolve_Conflict()` works, the example of figure 5 has been provided, where only bold edges are the mapped ones. The node and edge mapping of the common subgraph shown in figure 5(a) is represented by `CS_SET` = {(`CS`, `CANDIDATES`)}

Listing 6: The procedure `Reset()` starts a new common subgraph as consequence of the conflict.

```
Reset(v_1, v_2, CS, CANDIDATES)
{
  v = empty_node

  if(v_1 not NULL){
    Delete(v_1,CS)
    for each v such that Path(v,v_1,CS)
      Delete(v, CS)
    for each v such that Path(v,v_1,CANDIDATES)
      Delete(v,CANDIDATES)
  }
  if(v_2 not NULL){
    Delete(v_2,CS)
    for each v such that Path(v,v_2,CS)
       Delete(v, CS)
    for each v such that Path(v,v_2,CANDIDATES)
       Delete(v,CANDIDATES)
  }
}
```

where:

$$\texttt{CS} = \{(\texttt{a,a1},\texttt{NULL},\texttt{NULL}),(\texttt{c,f1},\beta,\beta 1),(\texttt{d,c1},\chi,\chi 1),(\texttt{b,b1},\alpha,\alpha 1)\}$$

and

$$\texttt{CANDIDATES} = \{(\texttt{c,c1},\delta,\delta 1),(\texttt{e,c1},\varepsilon,\chi 1)\}.$$

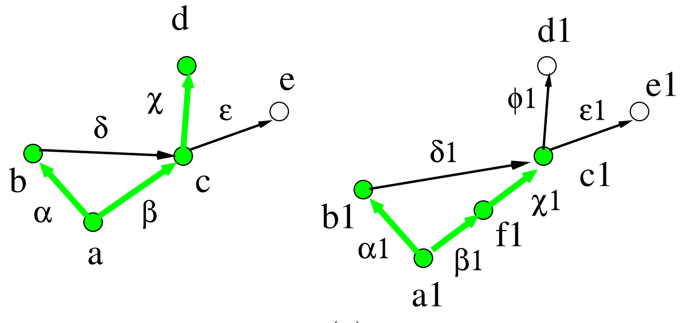
If `Pop()` selects the candidate ($\texttt{c,c1},\delta,\delta 1$), then both nodes `c` and `c1` are already mapped and the statement `Resolve_Conflict()` has to be executed. The new pair generated by `Reset()` (listing 6) is ($\texttt{CS}'$, $\texttt{CANDIDATES}'$), represented in figure 5(b), where:

$$\texttt{CS}' = \{(\texttt{a,a1},\texttt{NULL},\texttt{NULL}),(\texttt{b,b1},\alpha,\alpha 1),(\texttt{c,c1},\delta,\delta 1)\}$$

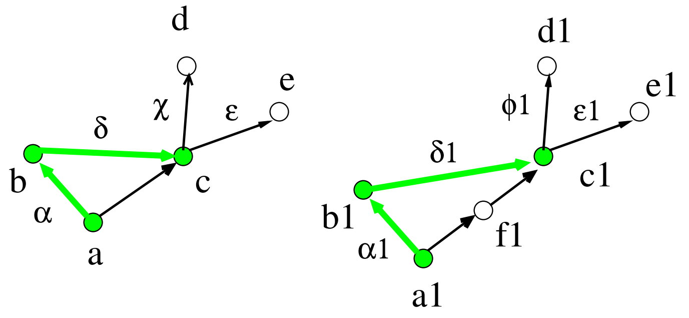
and

$$\texttt{CANDIDATES}' = \{(\texttt{d,d1},\chi,\phi 1),(\texttt{d,e1},\chi,\varepsilon 1),(\texttt{e,d1},\varepsilon,\phi 1),(\texttt{e,e1},\varepsilon,\varepsilon 1)\}$$

$\texttt{CS}'$ has been obtained from `CS` deleting the two elements involving `c` and `c1`, and the elements whose nodes are connected to `c` or `c1` with a path contained in `CS`. Analogous considerations are used to obtain $\texttt{CANDIDATES}'$ from `CANDIDATES`.



(a)



(b)

Figure 5: The conflict between the candidates (`c,c1`,δ, $\delta 1$) of the two graphs shown in a) is solved and a new common subgraph is produced b).

2.2 Correctness

The aim of this section is to provide an outline of the proof that the algorithm of figure 4 produces the MCSs of two input graphs. The first step of the naive algorithm enumerates all mappings among the nodes of the two graphs. If a node mapping m produced by the step 1 of the algorithm shown in figure 4 represents the MCS, the proposed algorithm is correct if the statement `CS_from_Mapping(`m`)` (listing 2) produces as output a common subgraph $\mathcal{CS}_m$ where the corresponding node mapping is identical to m.

In order to prove the previous assertions the following results have to be shown:

- the nodes involved in $\mathcal{CS}_m$ are an injective function among the nodes of the two graphs;
- $\mathcal{CS}_m$ is a common subgraph of the two input graphs, for each $m \in \mathcal{M}_{\mathcal{G}_1,\mathcal{G}_2}$;
- if m is a MCS, than the node mapping related to $\mathcal{CS}_m$ corresponds to the same MCS.

In the following, these notations are used: let $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1, \mu^{\mathcal{V}_1}, \mu^{\mathcal{E}_1})$ and $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2, \mu^{\mathcal{V}_2}, \mu^{\mathcal{E}_2})$ be two graphs,

$$\mathcal{C}S_m \subseteq (\mathcal{V}_1 \times \mathcal{V}_2 \times \mathcal{E}_1 \times \mathcal{E}_2)$$

is the set of four-tuples built from the input mapping $m \in \mathcal{M}_{\mathcal{G}_1,\mathcal{G}_2}$ by `CS_from_Mapping(`m`)`, and

$$K = \{(v_1, v_2) \mid (v_1, v_2, \cdot, \cdot) \in \mathcal{C}S_m\}$$

is the set of node mappings of $\mathcal{C}S_m$.

Lemma 1 *The set of node pairs K is a injective function between $\mathcal{V}_1$ and $\mathcal{V}_2$.*

Proof 2 *When the execution of `CS_from_Mapping(m)` ends, `CS_SET` is a set of pair (CS_i, C$ANDIDATES_i$) where C$ANDIDATES_i = \phi$ for each i, and the sets K_i are injective functions between nodes. This assertion can be proved by induction on $|K_i|$:*

base: *initially `CS_SET` $= \{(\phi,$ C$ANDIDATES_0)\}$, thus $K_0 = \phi$ and $|K_0| = 0$. In this case $K_0 = \phi$ satisfies the definition of injective function.*

induction step: *K_0 is a injective function where $|K_0| = n$. Let $(v_1, v_2, \cdot, \cdot)$ be the candidate extracted by `Pop()` from `CANDIDATES`. If the control at line 15 of `CS_from_Mapping()` (listing 2) is not satisfied, then there is no exists a pair $(v'_1, v'_2) \in K_0$ such that $v_1 = v'_1$ and $v_2 = v'_2$. Thus the four-tuple $(v_1, v_2, \cdot, \cdot)$ is added to CS_0 and $K_0 = K_0 \cup \{(v_1, v_2)\}$ is an injective function where $|K_0| = n + 1$. If the `if` condition is satisfied, `Resolve_Conflict()` has to be executed and a new pair (CS_1,C$ANDIDATES_1$) is added to `CS_SET`. CS_1 is obtained from CS_0 deleting the four-tuple $(v'_1, v'_2, \cdot, \cdot)$ that makes the `if` condition false and deleting also the four-tuples whose nodes are linked to v_1 and v_2 through a path in CS_0, then $(v_1, v_2, \cdot, \cdot)$ is added to CS_1. Thus K_1 is again an injective function.*

This result asserts that, during the construction of the common subgraph between $\mathcal{G}_1$ and $\mathcal{G}_2$, each node belonging to $\mathcal{G}_1$, is associated to one and only one node of $\mathcal{G}_2$. The following theorem uses this lemma to prove that each element (CS, C$ANDIDATES$) $\in$ `CS_SET` is a common subgraph of $\mathcal{G}_1$ and $\mathcal{G}_2$.

Theorem 2 *$\mathcal{C}S_m$ represents a common subgraph of the two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$.*

Proof 3 *The results to prove are:*

1. *the two graphs* $\mathcal{G}'_1 = (\mathcal{V}'_1, \mathcal{E}'_1, \mu^{\mathcal{V}'_1}, \mu^{\mathcal{E}'_1})$ *and* $\mathcal{G}'_2 = (\mathcal{V}'_2, \mathcal{E}'_2, \mu^{\mathcal{V}'_2}, \mu^{\mathcal{E}'_2})$ *are subgraphs of* $\mathcal{G}_1$ *and* $\mathcal{G}_2$ *respectively, where*

$$\begin{array}{rcl} \mathcal{V}'_1 & = & \{v \mid (v, \cdot, \cdot, \cdot) \in \mathcal{C}S_m\} \\ \mathcal{V}'_2 & = & \{v \mid (\cdot, v, \cdot, \cdot) \in \mathcal{C}S_m\} \\ \mathcal{E}'_1 & = & \{e \mid (\cdot, \cdot, e, \cdot) \in \mathcal{C}S_m\} \\ \mathcal{E}'_2 & = & \{e \mid (\cdot, \cdot, \cdot, e) \in \mathcal{C}S_m\} \end{array}$$

2. $\mathcal{G}'_1$ *and* $\mathcal{G}'_2$ *are isomorphic.*

The proof of the previous two points are given below:

1. $\mathcal{G}'_1$ *is a subgraph of* $\mathcal{G}_1$ *because* $\mathcal{V}'_1 \subseteq \mathcal{V}_1$ *and* $\mathcal{E}'_1 \subseteq \mathcal{E}_1$ *where each edge* $e \in \mathcal{E}'_1$ *has extreme nodes belonging to* $\mathcal{V}'_1$*. If an edge* $e = (v_1, v'_1) \in \mathcal{E}'_1$ *exists such that* $v_1 \notin \mathcal{V}'_1$ *or* $v'_1 \notin \mathcal{V}'_1$*, the statement* ***Update(***(v_1, v_2), *CANDIDATES) would find an edge not having* v_1 *as source node as out coming edge from* v_1*, or an edge having as opposite node of* v_1 *a node* v'_1 *not adjacent to itself. Analogous considerations hold for* $\mathcal{G}'_2$*.*

2. K *corresponds to a bijective function* $f : \mathcal{V}'_1 \rightarrow \mathcal{V}'_2$ *(*$\mathcal{V}'_1 \subseteq \mathcal{V}_1$ *and* $\mathcal{V}'_2 \subseteq \mathcal{V}_2$*) because, as shown by lemma 1, it is injective and* $|\mathcal{V}'_1| = |\mathcal{V}'_2|$*. The result to prove is that for each edge* $e_1 = (v_1, v'_1) \in \mathcal{E}'_1$ *there exists an edge* $e_2 = (f(v_1), f(v'_1)) \in \mathcal{E}'_2$ *and vice versa. If an edge* $e = (v_1, v'_1) \in \mathcal{E}'_1$ *exists such that* $f(v_1)$ *and/or* $f(v'_1)$ *are not extreme nodes of an edge belonging to* $\mathcal{E}'_2$ *, then* ***Update(***$(v_1, f(v_1))$, *CANDIDATES) would generates a candidate* $(v'_1, f(v'_1), e_1, e_2)$ *where* v'_1 *would be adjacent to* v_1 *and* $f(v'_1)$ *would not be adjacent to* $f(v_1)$*, which is impossible. Analogously for the vice versa.*

The next theorem assures that the algorithm produce the MCS of $\mathcal{G}_1$ and $\mathcal{G}_2$.

Theorem 3 *Let* m *be a node mapping representing an MCS between two input graphs, than the node mapping* K *related to* $\mathcal{C}S_m$ *corresponds to the same maximum common subgraph as* m*.*

Proof 4 *Let* $\mathcal{C}S_m$ *a common subgraph not representing the maximum common subgraph* m*, then the following cases have to be discussed:*

1. $K \neq m$, $|K| < |m|$: *The output of CS_from_Mapping(*m*) is* $\mathcal{C}S_m = \max\ \{\mathrm{CS}_i \mid (\mathrm{CS}_i,\ \mathrm{C}ANDIDATES_i) \in$ ***CS_SET***$\}$*, let (*CS*,*$\mathrm{C}ANDIDATES$*) be such output. Let us suppose, without lack of generality, that the difference between* K *and* m *is the node pair* (v_1, v_2)*. While executing CS_from_Mapping() the four-tuple* $(v_1, v_2, NULL, NULL)$ *belongs to the set* $\mathrm{C}ANDIDATES$

but it is never added to CS. *This means that it is never selected, but this is impossible because when* `CS_from_Mapping()` *ends all the* $\mathrm{CANDIDATES}_i$ *are empty. On the other hand, if the four-tuple were selected, it would cause the execution of* `Resolve_Conflict()`*. Also this case is absurd because neither* v_1 *nor* v_2 *can be already mapped because from lemma 1* K *is a bijective mapping.*

2. $K \neq m$ *and* $|K| > |m|$*: from theorem 2,* $\mathcal{CS}_m$ *corresponds to a common subgraph between the two input graphs, and since* m *corresponds to the MCS of th same two graphs,* $|K| > |m|$ *is absurd.*
3. $K \neq m$ *and* $|K| = |m|$*: the proof of this case is analogous to the proof of the point 1.*

3 Heuristics

The computation of the maximum common subgraph of two graphs is a common approach for comparing graphs, but its computational costs make the problem not tractable in many application domains. Most importantly, it is often necessary to insert heuristics in the matching process to be able to adapt the process to the characteristics of the shapes under examination. The algorithm described in section 2 is structured in a way that heuristic techniques can be easily plugged in it. Also, the expansion mechanism allows to gain in efficiency and speed as it will be discussed in this section.

The maximum common subgraph is obtained by providing as input to the `CS_from_Mapping()` all the mappings among the nodes of the two input graphs and by selecting the common subgraph with the largest number of nodes. A sensible improvement of the matching process can achieved by relaxing the problem setting and allowing a common subgraph to be accepted also an approximated solution.

3.1 Node Relevance and Initial Mapping

with respect to the example in Figure 6, the optimal solution, that is the MCS, is obtained running the expansion process simply on the pair $m = \{(\mathtt{a},\mathtt{a1})\}$. In this case the common subgraph obtained as output from `CS_from_Mapping(`m`)` corresponds to the MCS of the two graphs, and the process is run on a highly reduced input set of mapping. In general, running the algorithm on a subset of the initial mapping yields to approximations of the maximum common subgraph, and heuristics or semantic knowledge can be used to select the best candidate initial

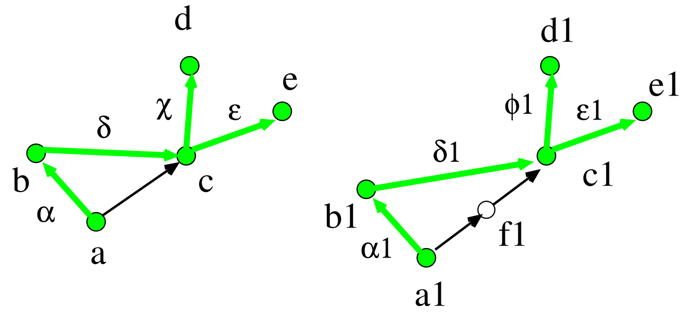


Figure 6: Nodes and edges belonging to the maximum common subgraph are: {(`a, a1`, NULL, NULL), (`b, b1`, $\alpha, \alpha 1$), (`c, c1`, $\delta, \delta 1$), (`d, d1`, $\chi, \phi 1$), (`e, e1`, $\varepsilon, \varepsilon 1$)}

mappings. It is clear, indeed, that some nodes are more relevant than others, depending on the attributes and on the topology of the graph.

A sensible improvement of the computational cost of the process can be obtained by reducing the number of input mappings, at the cost of accepting solutions that are not optimal, that is common subgraphs which might be not maximum. This task can be achieved taking into account the direction of the edges. Since the considered input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mu^{\mathcal{V}}, \mu^{\mathcal{E}})$ is directed, each node $v \in \mathcal{V}$ identifies a subgraph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}', \mu^{\mathcal{V}'}, \mu^{\mathcal{E}'})$ induced by $\mathcal{V}'$, where $\mathcal{V}'$ is the set of nodes with v as ancestor included v itself. For example, in figure 6, the nodes of the subgraph associated to the node **c** are: **d**, **e** and **c** itself.

The notion of node relevance can be captured by the subgraph associated to the node: for example, the larger the subgraph associated to the node is, the more the node is relevant. With reference to figure 6, the node **c** is more relevant than **d** and the node **a** is more relevant than **c**. This concept of relevance can be used to drive the selection of the best initial candidates for the expansion process. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mu^{\mathcal{V}}, \mu^{\mathcal{E}})$, for each node $v \in \mathcal{V}$, has been computed the related subgraph $\mathcal{G}_v = (\mathcal{V}_v, \mathcal{E}_v, \mu^{\mathcal{V}_v}, \mu^{\mathcal{E}_v})$, than the average node relevance $anr_{\mathcal{G}}$ has been computed as:

$$anr_{\mathcal{G}} = \frac{\sum_{v \in \mathcal{V}} |\mathcal{V}_v|}{|\mathcal{V}|} \tag{1}$$

The nodes that are relevant has been computed selecting all the nodes whose subgraph has a size bigger than the average node relevance defined in equation 1:

$$R_{\mathcal{G}} = \{v \in \mathcal{V} \mid |\mathcal{V}_v| \geq anr_{\mathcal{G}}\} \tag{2}$$

Figure 8(a) shows the two graphs of the horse and the wolf of figure 2 where the nodes are endowed with labels. In this example, the relevant node of the graph representing the horse and computed through the equation 2 are: $anr_{Horse} = \{\mathrm{n, m, l, i}\}$; while for the wolf are: $anr_{Wolf} = \{\mathrm{r1, q1, p1, n1, i1}\}$.

A more general definition of the equations 1 and 2 takes in account also attributes of nodes and edges of the graph. The relevance $Rel_{\mathcal{G}}(v)$ of the node v can be defined as:

$$Rel_{\mathcal{G}}(v) = |\mathcal{V}_v| \; w_{\mathcal{G}}(v) \tag{3}$$

where $w_{\mathcal{G}}(v)$ is a weight varying on the $[0, 1]$ range and depending on the attributes of nodes and edges of the subgraph $\mathcal{G}_v$ associated to the node v. The weight captures the meaning of the object sub-parts represented as nodes and edges of the graph. For example a complex subgraph, with many nodes and edges but representing small object sub-parts, may be less relevant of a simple subgraph, with few nodes and edges but representing a big object sub-part. Starting from the equation 3, the equation 1 can be rewritten as:

$$\overline{anr_{\mathcal{G}}} = \frac{\sum_{v \in \mathcal{V}} Rel_{\mathcal{G}}(v)}{|\mathcal{V}|} \tag{4}$$

and thus also the set of relevant nodes can be rewritten as:

$$\overline{R_{\mathcal{G}}} = \{v \in \mathcal{V} \mid Rel_{\mathcal{G}}(v) \geq \overline{anr_{\mathcal{G}}}\} \tag{5}$$

For example, assuming that the attribute of the edge represents the volume of the object sub-part described by the edge itself, the weight determining the relevance of a node v, can be defined as:

$$w_{\mathcal{G}}(v) = \frac{\sum_{e \in \mathcal{E}_v} \mu^{\mathcal{E}_v}(e)}{\sum_{e \in \mathcal{E}} \mu^{\mathcal{E}}(e)} \tag{6}$$

In this case nodes whose subgraphs describe sub-parts with a small volume (with respect to the volume of the whole object) are less relevant than nodes describing sub-parts with a large volume. In figure 7(a) the model of a human body is shown together with its structural descriptor, figure 7(b). The nodes belonging to the subgraph that correspond to the left hand, figure 7(c), are considered relevant with respect to the equations 1 and 2. On the contrary, due to the small volume of the fingers and the palm of the hand, the nodes of the Left hand are not relevant with respect to the equations 3, 6 and 5.

After the relevant nodes has been computed, they can be used as input for the expansion process defined by `CS_from_Mapping(`m`)`. The relevant nodes can be combined to produce the initial node mapping

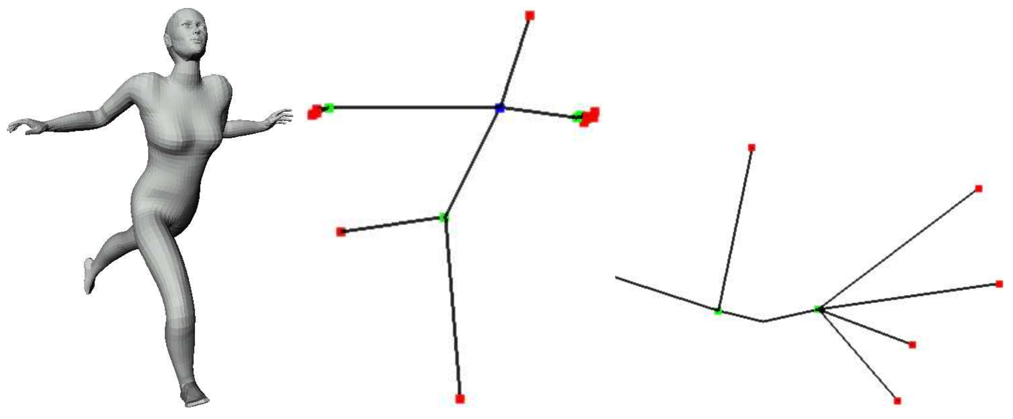


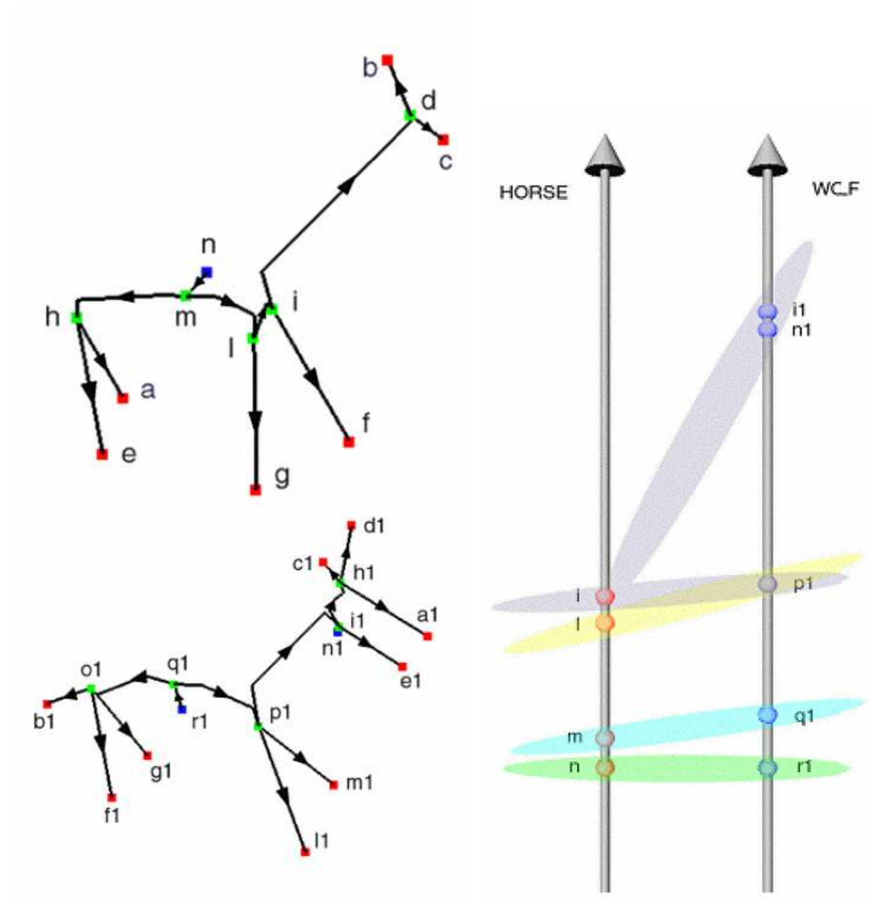
Figure 7: The model of a human body (a), its Reeb graph with respect to the integral geodesic distance (b) and the subgraph representing the left hand of the model (c).

m. The simplest way to compute the initial mapping between two input graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, is to define it as Cartesian product between the set of relevant nodes:

$$m = R_{\mathcal{G}_1} \times R_{\mathcal{G}_2} \tag{7}$$

An example of common subgraph obtained considering the set of relevant nodes defined by the equation 2 and the initial mapping defined by the equation 7 is shown in figure 2(a). In this case, although heuristic technique has been used, the comparison algorithm produces as output the maximum common subgraph.

Another method to combine the relevant node is shown in figure 8(b), where they are combined with respect to their attributes, in this case, the distance value from the barycentre of the object. Relevant nodes has been grouped in order to combine nodes close to the barycentre with node of the other graph close to the barycentre, and node far from the barycentre with nodes of the other graph far from the barycentre. Nodes close to the barycentre are associated to large subgraphs, thus, are associated to internal large sub-parts of the object, while nodes that move away from the barycentre are associated to progressively small subgraphs (possibly empty subgraphs), thus, are associated to small external sub-parts of the object. The node relevance detection together with the relevant node grouping, allow a coarse to fine matching process that first produces a correspondence between sub-parts of the two object and then it refines the coarse correspondences producing finer sub-part correspondence.S



(a) (b)

Figure 8: The graphs of a horse and a wolf (a), and an example of node clustering among relevant nodes (b).

3.2 Distance Function Between Nodes

Another useful heuristic technique can be constructed by combining the notion of subgraph relevance to the idea of expansion process, in particular associating to the pair of nodes (v_1, v_2) the information about how much the common subgraph would expand with the addition of that pair to the subgraph itself. A distance function d between two nodes v_1 and v_2 could be defined in order to capture this information.

The distance $d(v_1, v_2)$ is defined involving node and edge attributes and an approximation of the structure of the subgraph related to v_1 and v_2. Two examples of distances that can be plugged in the algorithm come from [SD01] and [HSKK01]. In [SD01] a topological signature vector $\chi(v)$ describing the structure of the subgraph related to the node v is defined for each node of the graph. The distance $d(v_1, v_2)$ corresponds to the euclidean distance between $\chi(v_1)$ and $\chi(v_2)$. In [HSKK01] the distance value depends mainly by the node attributes.

The distance $d(v_1, v_2)$, proposed in this report, involves node and edge attributes and the approximation of the structure of the subgraphs related to v_1 and v_2. It is defined as:

$$d(v_1, v_2) = \frac{w_1 G\_S + w_2 St\_S + w_3 Sz\_S}{w_1 + w_2 + w_3} \tag{8}$$

Where G_S, St_S and Sz_S are real numbers belonging to the range $[0, 1]$. They represent geometrical similarity between the node attributes, structural similarity between the node subgraphs and Sz_S evaluate the similarity between the size of the sub-parts associated to nodes, finally, the three weight w_1, w_2 and w_3 belong to the range $[0, 1]$ and combine the three components of d.

G_S compares the geometric shape descriptors associated to v_1 and v_2. The geometric descriptors capture the parts of the object corresponding to the subgraph associated both to v_1 and v_2. St_S compares the structure between the subgraphs associated to the nodes v_1 and v_2. The structures of two subgraphs can be compared by analyzing the spectrum of the graph, while another coarse but efficient technique can be defined as follow:

$$St\_S = \frac{\overline{in} + \overline{out} + \overline{sub\_n} + \overline{sub\_in} + \overline{sub\_out}}{5} \tag{9}$$

where

$$\overline{X} = \frac{|X(v_1) - X(v_2)|}{\max(X(v_1), X(v_2))}$$

and where *in* and *out* represent the in degree and the out degree of the two nodes, *sub_n* the number of the subgraph nodes, *sub_in* and *sub_out* the in degree and out degree sum of the subgraph nodes. Finally, two nodes may have sub-parts similar both in structure and in geometry but not in size, in this case they have to result dissimilar. This kind of dissimilarity is captured by Sz_S defined as:

$$\begin{aligned} Sz\_S &= \overline{sub\_s} \\ sub\_s &= w_{\mathcal{G}} \end{aligned} \tag{10}$$

where $w_{\mathcal{G}}$ is the sum of the edge attributes of the subgraph and can be defined as in the equation 6.

The distance d can be used to reduce the number of elements of `CS_SET`. It acts on the selection of the $CANDIDATES$ elements and the `Resolve_Conflict()` statement. The simplest way to use d is to extract the best element $(v_1, v_2, \cdot, \cdot)$ (minimum distance between v_1 and v_2) from $CANDIDATES$ and add it to CS if and only if neither v_1 nor v_2 are already mapped. If v_1 or v_2 are mapped, the candidate is discarded and a new one is extracted until $CANDIDATES$ becomes empty. In this case the `Resolve_Conflict()` statement is never recalled. Another example of use of d is to add the best candidate (v_1, v_2) to CS even if v_1 and/or v_2 are already mapped if and only if the new mapping has a minor distance than the previous ones. In both the previous cases the `CS_SET` set corresponds to a single pair $(CS, CANDIDATES)$.

4 Computational complexity

The computational cost of the algorithm described in figure 4 and detailed in section 2.1 is exponential, because it depends on the first step of the algorithm, that enumerates all the mappings among the nodes of the two input graphs. Actually, given two graphs $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1, \mu^{\mathcal{V}_1}, \mu^{\mathcal{E}_1})$ and $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2, \mu^{\mathcal{V}_2}, \mu^{\mathcal{E}_2})$ where $\|\mathcal{V}_1\| = n$ and $\|\mathcal{V}_2\| = m$, the set of node mappings from $\mathcal{G}_1$ to $\mathcal{G}_2$ has m^n elements.

The heuristic techniques discussed in section 3 reduce drastically the computational costs of the matching algorithm. The processes that extract the information describing the structure of the subgraph related to a node, and the computation of the relevant nodes of the graphs can be considered as pre-processing steps that are off-line with respect to the graph matching algorithm. However the information involved in the equations 3 and 9 correspond to the number of nodes belonging to the subgraph associated to a specific node, the sum of the in degree and out degree of the subgraph nodes and the information encoded into the nodes and edges attributes describing the size of the sub-part of the object related to the subgraph. The extraction of all these information have $O(n^2)$ as computational cost, where n is the number of nodes of the graph. This quadratic cost is obtained by analyzing the descendant nodes of each node of the graph. The choice of the set of relevant nodes, defined in the equation 5 has a linear computational cost, and the generation of the initial mapping is quadratic with respect to the sum of the relevant nodes of both the graphs, while the evaluation of the similarity between two nodes defined in the equation 8 depend only by the computational cost of the comparison process used for the geometric attributes, because all the involved information are computed in the pre-processing step.

Given the initial node mapping and the heuristics defined in section 3.2 also the expansion process has quadratic cost with respect to the maximum number of nodes of both the two graphs. When a pair of nodes is selected from the set of candidates (`Pop()` procedure mentioned in the listing 2) is added to the common subgraph or it is discarded because, at least, one of the two nodes is already mapped (heuristic rule discussed in section 3.2 that influences the `Resolve_Conflict()` procedure shown in the listings 2 and 5). For each pair of nodes added to the common subgraph, at most, $O(n^2)$ new candidates are added to C$ANDIDATES$, and such pair is added to the common subgraph exactly once. The extraction of the best candidate from C$ANDIDATES$ is constant if the candidates are arranged with respect to the distance function between nodes (equation 8), while the insertion of a candidate into the candidates set is $\log m$, where m is the number of candidates.

Finally the algorithm described in section 2 endowed with the heuristic techniques discussed in section 3, approximates the maximum common subgraph with a computational cost of $O(n^3)$, where n is the maximum between the number of nodes of the two input graphs.

5 Reeb Graph Simplification

Due to the capability of the structural shape descriptors to capture salient structural and topological features, the object may be represented by a graph with a some redundant nodes and edges, as shown in figure 9(a). In this case, such redundant information can be considered as structural noise of the shape descriptor, and it can be removed from the graph through a simplification algorithm able to highlight the significative shape of the object, maintaining the topological information captured by the descriptor, as shown in figure 9(b).

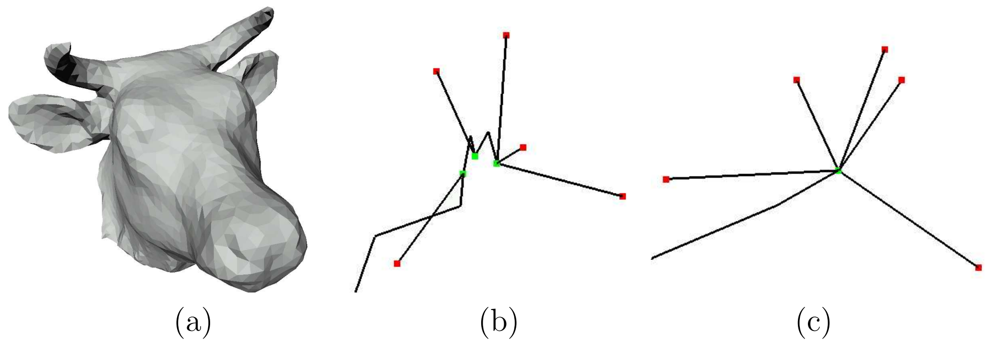


Figure 9: A 3D model (a), its structural shape descriptor (b) and a simplified version of the descriptor (c). In (c) the redundant nodes (green nodes) of the graph has been detected and collapsed into only one node.

The simplification algorithm proposed in this section is not a simple pruning but a more complex process able to transform the input graph into a new one, which describes the global structure of the object and discards the graph elements responsible for the noise.

Aim of the simplification algorithm is to eliminate the non-relevant object features represented in the graph by guaranteeing the topological consistency of the underlying model. In order to conserve the capability of the Reeb graph to represent the topology of the object, when an edge is removed from the graph also a node has to be deleted. Moreover, in order to preserve the topology and the acyclicity of the graph, a node n and an edge $e = (v, n)$ are deleted from $\mathcal{G}$ only if there are not other edges e_1 from the same node v to n.

Minima and maxima are regarded as feature nodes, while saddle ones describe how the features and the "body" parts of the object are connected. Three threshold values, Th_m, Th_M and Th_s, are set to check the relevance of both minima, maxima and saddles nodes respectively. Such thresholds, which vary from the smallest to the biggest edge attribute, may be either automatically computed, for instance as the average of the edge attributes, or user defined. The main idea is to simplify nodes that have relevance less or equal to the given thresholds. The simplification is the more incisive, the higher threshold is.

Simple minimum and maximum simplification: In the Reeb graph, simple minima and maxima are represented by nodes, which may be linked to both saddle and maximum (minimum) nodes. However, when a simple maximum M is connected to a simple minimum m, the graph is composed only by the two nodes m and M and the edge (m, M) because a surface S having only two critical points is homeomorphic to a sphere [Mil63].

Only minimum and maximum nodes that are adjacent to saddle nodes and less relevant than Th_m and Th_M are simplified. When all minima and maxima adjacent to a saddle node are removed, the saddle might be transformed into a minimum or a maximum node (eventually complex), according to the number and the direction of the edges incident into it, see figure 10.

Complex node simplification: Complex maxima and minima can be adjacent both to nodes and macro-nodes, has to be recalled that macro-nodes correspond to complex maxima and minima. A complex minimum mm is not relevant if there exists an edge $e = (mm, s)$ where s is a saddle node and the attribute of e is smaller than Th_m. If there exists more than one edge with such characteristics, the simplification

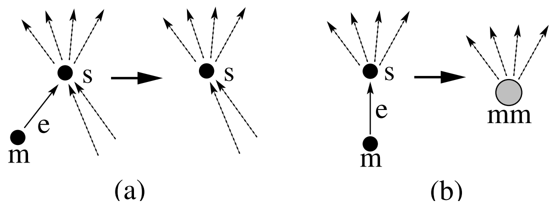


Figure 10: The simplification of a minimum node. The labels of the node indicate its type.

algorithm chooses the one with the smallest relevance. The node mm is simplified removing the edge e and connecting all other outgoing nodes to the saddle node s. The node mm is also removed and the saddle s re-classified according to its behavior in the new situation (see figure 11). Complex maxima are handled in a symmetric manner.

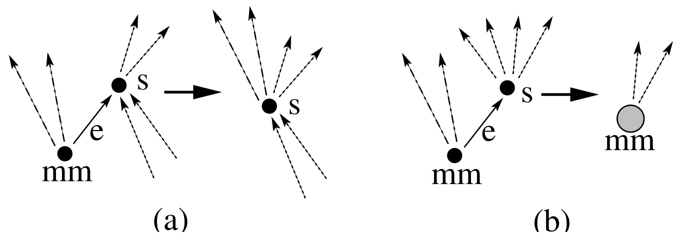


Figure 11: The simplification of a minimum macro-node. The labels of the node indicate its type.

Saddle simplification: Saddles are graph nodes with both ingoing and outgoing edges. A saddle node v_1 can be simplified if it is connected to another saddle v_2 and the edge $e = (v_1, v_2)$ connecting the two saddles has attribute smaller than the threshold Th_s. In this case all nodes adjacent to v_1 are connected to v_2 and both the edge e and the node v_1 are removed, see figure 12.

The simplification process may be arbitrarily repeated until a very simple structure is reached and no more nodes can be simplified according to the proposed criteria.

In particular, has to be highlighted that the constraints inserted in the simplification process of macro-nodes produce a final graph representation which is topologically equivalent to the original one but may not be minimal, in the number of nodes and edges. In addition the order in which the simplification operations are performed may influence the final results.

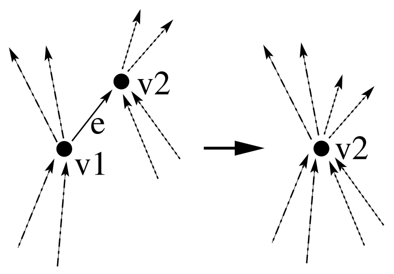


Figure 12: The simplification of a saddle v_1.

A Basic Definitions on Graphs

Definition 1 (Attributed graph) *An attributed graph $\mathcal{G}$ is given by a quadruple $G = (V, E, \mu_V, \mu_E)$, where V is a set of nodes, E is the set of the graph edges, $\mu_V : V \to A_V$ and $\mu_E : E \to A_E$ are the node and the edge attribute functions, with A_V, A_E sets of node and edge attributes of G. The set of attributed graphs is denoted by M_{Gset}.*

Definition 2 (Attributed sub-graph) *A subgraph S of G is a quadruple $(V_S, E_S, \mu_{V_S}, \mu_{E_S})$, where $V_S \subseteq V$, $E_S \subseteq E$, μ_{V_S} and μ_{E_S} are induced by μ_V and μ_E, respectively.*

Definition 3 (path) *a path between two vertices n_1, $n_2 \in V$ is a non-empty sequence of k different vertices $v_0, v_1, \dots, v_k$ where $v_0 = n_1$ and $v_k = n_2$ and $(v_i, v_{i+1}) \in E$, $i = 0, \dots, k-1$. Finally, a graph G is said to be* acyclic *when there are no cycles between its edges, independently of whether the graph G is directed or not.*

Definition 4 (graph isomorphism) *is a bijective function $f : \mathcal{V}_1 \to \mathcal{V}_2$ such that*

1. *$\mu^{\mathcal{V}_1}(v) = \mu^{\mathcal{V}_2}(f(v))$, $v \in \mathcal{V}_1$.*
2. *for all the edges $e_1 = (v_1, v_1') \in \mathcal{E}_1$, there exists an edge $e_2 = (f(v_1), f(v_1')) \in \mathcal{E}_2$ such that $\mu^{\mathcal{E}1}(e_1) = \mu^{\mathcal{E}_2}(e_2)$. Moreover, for all the edges $e_2 = (v_2, v_2') \in \mathcal{E}_2$, there exists an edge $e_1 = (f^{-1}(v_2), f^{-1}(v_2')) \in \mathcal{E}_1$ such that $\mu^{\mathcal{E}_1}(e_1) = \mu^{\mathcal{E}_2}(e_2)$.*

If a graph is not attributed, the condition 1 and the equality between the edge attributes in the condition 2, are not necessary.

Definition 5 (subgraph isomorphism) *If $f : \mathcal{V}_1 \to \mathcal{V}_1'$ is a graph isomorphism between $\mathcal{G}_1$ and $\mathcal{G}'$, and $\mathcal{G}'$ is a subgraph of $\mathcal{G}_2$, then f is called a subgraph isomorphism from $\mathcal{G}_1$ to $\mathcal{G}'$.*

Definition 6 (common subgraph) *A common subgraph of $\mathcal{G}_1$ and $\mathcal{G}_2$ is a graph $\mathcal{G}$ such that there exists a subgraph isomorphism from $\mathcal{G}$ to $\mathcal{G}_1$ and from $\mathcal{G}$ to $\mathcal{G}_2$.*

Definition 7 (maximum common subgraph) *A maximum common subgraph of $\mathcal{G}_1$ and $\mathcal{G}_2$, denoted as $\mathcal{MCS}_{\mathcal{G}_1,\mathcal{G}_2}$ is a common subgraph $\mathcal{G}$ such that there exists no other common subgraph having more nodes than $\mathcal{G}$. The $\mathcal{MCS}_{\mathcal{G}_1,\mathcal{G}_2}$ in not necessarily unique.*

Definition 8 (error tolerant graph isomorphism) *Let $\mathcal{G}$ and $\mathcal{G}'$ be two attributed graphs as proposed in definition 1 and $\Delta = (\delta_1, \ldots, \delta_n)$ a sequence of graph editing operations, where a graph edit operation, δ_i, is an addition, a deletion or an attribute modification of nodes and edges, then:*

- *the* edited graph $\Delta(\mathcal{G})$ *is* $\Delta(\mathcal{G}) = \delta_n(\delta_{n-1}(\ldots (\delta_1(\mathcal{G})) \ldots))$;
- *an* error tolerant graph isomorphism *is a couple* (Δ, ψ), *where Δ is a sequence of editing operations such that there exists an graph isomorphism ψ between $\Delta(\mathcal{G})$ and $\mathcal{G}'$.*

References

[BMM+03] S. Biasotti, S. Marini, M. Mortara, G. Patané, M. Spagnuolo, and B. Falcidieno. 3D shape matching through topological structures. In I. Nyströn, G. Sanniti di Baja, and S. Svennson, editors, *Proceedings of the* 11th *Discrete Geometry for Computer Imagery Conference*, volume 2886 of *Lecture Notes in Computer Science*, pages 194–203, Naples, 2003. Springer Verlag.

[BMMP03] S. Biasotti, S. Marini, M. Mortara, and G. Patané. An overview on properties and efficacy of topological skeletons in shape modelling. In *Proceedings of Shape Modelling and Applications*, pages 245–254, Seoul, South Korea, May 2003. IEEE Press.

[HSKK01] M. Hilaga, Y. Shinagawa, T. Komura, and T. L. Kunii. Topology matching for fully automatic similarity estimation of 3D shapes. In *ACM Computer Graphics, (Proc. of SIGGRAPH 2001)*, pages 203–212, Los Angeles, 2001. ACM Press.

[Mil63] J. Milnor. *Morse Theory.* Princeton University Press, New Jersey, 1963.

[SD01] A. Shokoufandeh and S. Dickinson. A unified framework for indexing and matching hierarchical shape structures. In *Proceedings of 4th International Workshop on Visual Form*, Capri, Italy, May 28–30 2001.

[SSGD03] H. Sundar, D. Silver, N. Gagvani, and S. Dickinson. Skeleton based shape matching and retrieval. In *Proceedings of Shape Modelling and Applications*, pages 130–139, Seoul, South Korea, June 2003. IEEE Press.