A distance measure between labeled combinatorial maps

Tao Wang, Guojun Dai, Bingbing Ni, De Xu, Francois Siewe

Abstract
Combinatorial maps are widely used in image representation and processing, however map matching problems have not been extensively researched. This paper addresses the problem of inexact matching between labeled combinatorial maps. First, the concept of edit distance is extended to combinatorial maps, and then used to define mapping between combinatorial maps as a sequence of edit operations that transforms one map into another. Subsequently, an optimal approach based on A* algorithm and an approximate approach based on Greedy algorithm are proposed to compute the distance between combinatorial maps. Experimental results show that the proposed inexact map matching approach produces richer search results than the exact map matching technique by tolerating small difference between maps. The proposed approach performs better in practice than the previous approach based on maximum common submap which cannot be directly used for comparing labels on the maps.

1. Introduction

Graphs are widely used in the representation of complex structures, such as DNA sequences, documents, texts, and images. And many problems in these fields can be formulated as graph matching problem [1]. Traditionally, in many image processing applications [2–4], Region Adjacency Graphs (RAGs) use vertices to describe the maximal homogeneous sets of connected pixels and edges to describe the adjacency relationships. A main advantage of this method is its capability of modeling the structural relationships of image primitives, so that graph-based representations of images are close to the perception of the human visual system.

Although RAGs can model adjacency relationships of image primitives, there is still some important information absent in this model. In particular, the orientation of edges around vertices has not been explored in traditional RAGs. However the orientation of edges usually represents the spatial relationships between objects and is important for many applications, such as shape retrieval [5] and mobile robotics [6]. Also, RAGs cannot represent multi-adjacency. These issues result in ambiguities for RAG-based image representation [7]. To get round this default, the RAG model has to be extended.

Combinatorial maps define a general framework which allows for the encoding of any subdivision on n-dimensional topological spaces. Compared with the traditional graph model, combinatorial map is more precise for explicitly encoding the orientation of edges around vertices. Combinatorial maps have been utilized in 2D and 3D image representation and processing [8–14].

Matching combinatorial maps is therefore an important problem in the field of image understanding. There have been some previous works on the map matching problem. The early research can be traced back to Cori who discussed the computation of the automorphism group of a topological graph embedding in his report [15]. Liu defined sequence descriptions for combinatorial maps [16], which are subsequently used for map isomorphism and map automorphism [17]. Gosselin et al. proposed two map signatures which are used to efficiently search for a map in a database [18]. Damiand et al. proposed a polynomial algorithm for searching compact submap in planar maps [19], and then extended this work to n-dimensional open combinatorial maps [20]. Wang et al. proposed a quadratic algorithm for submap isomorphism based on sequence searching [21].

However, all these works only aim for the exact map matching problem, while the inexact map matching problem has not been researched extensively and systematically. In real applications, two objects having small structural differences are usually considered as matched. Also, real world objects are usually affected by noises so that map representations extracted from identical objects at different time are rarely exactly equal. Therefore, it is necessary to integrate some degree of error-tolerance into the map matching process. Neuhaus and Bunke proposed an error-tolerant
approximate edit distance algorithm for planar maps [39]. This algorithm is very efficient and suitable for large planar maps. Since the resulting edit path of this algorithm strongly depends on the seed substitution, the solution can be optimized by computing several times with different seed substitutions. Combier et al. defined a first error-tolerant distance measure for comparing generalized maps by means of the size of a largest common submap [22]. In most scenarios maps extracted from real world objects are always labeled. However, the previous approach cannot be directly used for comparing labels on the maps. It therefore demands research attention on how to measure distance between labeled maps in real applications.

In this paper, we address the problem of measuring the distance between two combinatorial maps, which is one of the most important and fundamental issues of the inexact map matching problem. In particular, we aim to solve the labeled map matching problem via relaxing the problem to inexact but efficient matching. We first extend the concept of edit distance to combinatorial maps (Section 3.1), and formally define the mapping of combinatorial maps (Section 3.2) to describe how a sequence of edit operations transforms one map into another. Then we propose an algorithm for computing the map edit distance based on a tree search approach (Section 3.3). This algorithm guarantees to find an optimal edit path, but it needs exponential computational time and space. To reduce the computational complexity, we subsequently propose an approximate algorithm for map edit distance based on a Greedy strategy (Section 3.4). The Greedy algorithm gains high computational efficiency at the cost of that it usually fails to produce optimal solution. Experiments in Section 4 show that the inexact map matching approach provides better search results than the exact map matching (Section 3.4). The Greedy algorithm gains high computational efficiency at the cost of that it usually fails to produce optimal solution. Experiments in Section 4 show that the inexact map matching approach provides better search results than the exact map matching (Section 3.4). The Greedy algorithm gains high computational efficiency at the cost of that it usually fails to produce optimal solution. Experiments in Section 4 show that the inexact map matching approach provides better search results than the exact map matching (Section 3.4). The Greedy algorithm gains high computational efficiency at the cost of that it usually fails to produce optimal solution. Experiments in Section 4 show that the inexact map matching approach provides better search results than the exact map matching (Section 3.4). The Greedy algorithm gains high computational efficiency at the cost of that it usually fails to produce optimal solution. Experiments in Section 4 show that the inexact map matching approach provides better search results than the exact map matching (Section 3.4). The Greedy algorithm gains high computational efficiency at the cost of that it usually fails to produce optimal solution. Experiments in Section 4 show that the inexact map matching approach provides better search results than the exact map matching (Section 3.4). The Greedy algorithm gains high computational efficiency at the cost of that it usually fails to produce optimal solution. Experiments in Section 4 show that the inexact map matching approach provides better search results than the exact map matching (Section 3.4).

\[ \psi(2\xi(x)) = \xi_2(\psi(x)) \]
\[ \psi(1\xi(x)) = 1\xi_2(\psi(x)) \]
\[ \xi G_1 = \xi G_2 \]
\[ G_1 = \xi G_2 \]
\[ G_1 = \xi G_2 \]
\[ G_1 = \xi G_2 \]
\[ G_1 = \xi G_2 \]
\[ G_1 = \xi G_2 \]
\[ G_1 = \xi G_2 \]
\[ G_1 = \xi G_2 \]

Definition 1 (Labeled combinatorial map). A labeled combinatorial map \( G = (D, \alpha, \sigma, \mu) \) where

- \( D \) is a finite set of darts,
- \( \alpha \) is the involution on \( D \),
- \( \sigma \) is the permutation on \( D \),
- \( \mu \) is a dart label function.

Fig. 1 demonstrates the derivation of a combinatorial map from a plane graph, where \( D = \{1, -1, 2, -2, 3, -3, 4, -4, 5, -5, 6, -6\} \), \( \alpha = \{(1, -1), (2, -2), (3, -3), (4, -4), (5, -5), (6, -6)\} \) and \( \sigma = \{(1, 2), (3, 4), (5, -4), (7, -2), (6, -3), (7, -5)\} \). Usually, \( \mu \) is a partial function mapping darts to a finite set of integers, characters or vectors. A labeled map \( G = (D, \alpha, \sigma, \mu) \) is connected if for any two darts \( x \) and \( y \) in \( D \), \( y \) can be reached from \( x \) by successive applications of the involution \( \alpha \) and the permutation \( \sigma \). For the sake of simplicity, maps in this paper are connected and vertices are unlabeled unless otherwise stated.

Compared with graph isomorphism problem, checking isomorphism of maps needs to integrate additional constraints on preserving topological relationships between edges. By considering this, the map isomorphism problem becomes simple [25].

Definition 2 (Map isomorphism). Given two labeled maps \( G_1 = (D_1, \alpha_1, \sigma_1, \mu_1) \) and \( G_2 = (D_2, \alpha_2, \sigma_2, \mu_2) \), if there is a one-to-one mapping \( \psi : D_1 \rightarrow D_2 \) such that for any \( x \in D_1 \), there are

\[ \psi(2\xi(x)) = \psi(1\xi(x)) \]
\[ \psi(1\xi(x)) = 1\xi_2(\psi(x)) \]
\[ \xi G_1 = \xi G_2 \]
\[ G_1 = \xi G_2 \]
\[ G_1 = \xi G_2 \]
\[ G_1 = \xi G_2 \]
\[ G_1 = \xi G_2 \]
\[ G_1 = \xi G_2 \]
\[ G_1 = \xi G_2 \]

then \( G_1 \) and \( G_2 \) are considered isomorphic.

Combinatorial maps explicitly encode the information of the orientation of darts around vertices. This information enables us to differentiate between configurations in the graph matching problem, and it is not encoded by region adjacency graphs. Moreover, combinatorial maps may be defined in any dimensions. Indeed, the 3D image representation and processing based on combinatorial maps is an active research field [11–13].

The following notations will be used in the rest of this paper. Given a map \( G \), let \( E(G) \) and \( V(G) \) denote the edge set and the vertex set of \( G \) respectively. Given a dart \( x \), let \( \text{tx} \) and \( \text{hx} \) denote the tail vertex and head vertex of \( x \) respectively, and let \( \text{ex} \) denote the corresponding edge of \( x \).

2.2. Graph edit distance

Edit distance is one of the most flexible methods for error-tolerant matching of structures. It was initially introduced for string-to-string comparison [29], and was later extended to compare trees [30–33] and graphs [34–36].

A standard set of edit operations on graphs is given by insertion, deletion and substitution of both vertices and edges. The substitution of two nodes \( u \) and \( v \) is denoted by \( (u \rightarrow v) \), the deletion of node \( u \) by \( (u \rightarrow \varnothing) \), and the insertion of vertex \( v \) by \( (\varnothing \rightarrow v) \). Similar notations are used for edges. Given two graphs, the source graph \( g_1 \) and the target graph \( g_2 \), a sequence of edit operations \( s_1, \ldots, s_k \) that transforms \( g_1 \) into \( g_2 \) is called an edit path between \( g_1 \) and \( g_2 \).

Definition 3 (Graph edit distance). Let \( g_1 \) be the source and \( g_2 \) be the target graph. The graph edit distance between \( g_1 \) and \( g_2 \) is defined by:

\[ d(g_1, g_2) = \min_{(c_1, \ldots, c_k) \in S(g_1, g_2)} \sum_{i=1}^{k} c(S_i), \]

where \( S(g_1, g_2) \) denotes the set of all edit paths transforming \( g_1 \) into \( g_2 \), and \( c \) denotes the cost function measuring the strength \( c(S_i) \) of edit operation \( S_i \).
The computation of the graph edit distance is usually carried out by means of a tree search approach based on the A* algorithm. The basic idea is to organize the underlying search space as an ordered tree, in which the root represents the starting point of the search procedure, inner vertices correspond to partial solutions, and leaf vertices represent complete but not necessarily optimal solutions. Such a search tree is constructed dynamically at runtime by iteratively creating successor vertices linked by edges to the currently considered vertices in the search tree. A heuristic function is usually used to integrate more knowledge about partial solutions and determine the most promising vertex in the current search tree. Formally, for a vertex \( p \) in the search tree, use \( g(p) \) to denote the cost of the partial edit path accumulated so far, and use \( h(p) \) to denote the estimated cost from \( p \) to a leaf vertex. In the simplest scenario this estimation \( h(p) \) is set to zero for each vertex \( p \), which is equivalent to using no heuristic information about the present situation at all.

The \( A^* \)-based algorithm for optimal graph edit distance computation is shown in Algorithm 1. This algorithm guarantees that the complete edit path is always optimal, i.e., has minimal costs among all possible edit paths. Let \( OPEN \) be the set of underlying edit paths. Note that edit operations on edges are implied by edit operations on their origin vertices. The cost of these implied edge operations are dynamically added to the corresponding paths in \( OPEN \).

**Algorithm 1. Graph edit distance algorithm**

| Input: Non-empty graphs \( g_1 = (V_1, E_1) \) and \( g_2 = (V_2, E_2) \), where \( V_1 = \{u_1, \ldots, u_{|V_1|}\} \) and \( V_2 = \{v_1, \ldots, v_{|V_2|}\} \) |
| Output: An edit path \( p_{\text{min}} \) from \( g_1 \) to \( g_2 \) with minimum cost. |

1. Initialize \( OPEN \) to the empty set \( \{ \} \).
2. For each node \( w \in V_2 \), insert the substitution \( \{u_1 \rightarrow w\} \) into \( OPEN \).
3. Insert the deletion \( \{u_1 \rightarrow \} \) into \( OPEN \).
4. \textbf{loop}
5. Remove \( p_{\text{min}} = \arg\min_{p \in OPEN}[g(p) + h(p)] \) from \( OPEN \).
6. if \( p_{\text{min}} \) is a complete edit path then
7. Return \( p_{\text{min}} \) as the solution.
8. else
9. Let \( p_{\text{min}} = \{u_1 \rightarrow v_{h_1}, \ldots, u_k \rightarrow v_{h_k}\} \).
10. if \( k < |V_1| \) then
11. For each \( w \in V_2 \{v_{h_1}, \ldots, v_{h_k}\} \), insert \( p_{\text{min}} \cup \{u_{h_k+1} \rightarrow w\} \) into \( OPEN \).
12. Insert \( p_{\text{min}} \cup \{u_{h_k+1} \rightarrow \} \) into \( OPEN \).
13. else
14. Insert \( p_{\text{min}} \cup \{u_{h_k+1} \rightarrow A\} \) into \( OPEN \).
15. end if
16. end if
17. \textbf{end loop}

End Algorithm.

### 3. The proposed approach to inexact map matching

In this section, we extend the concept of edit distance to combinatorial maps, and then propose an optimal approach based on \( A^* \) algorithm and an approximate approach based on Greedy algorithm for the computation of map edit distance.

#### 3.1. Map edit distance

Similar to graph edit distance, the key idea of map edit distance is to define the dissimilarity, or distance between maps by the minimum number of edition operations that is needed to transform one map into another.

A standard set of edit operations includes insertion, deletion and substitution of both darts and vertices. We denote the substitution of two darts \( x \) and \( y \) by \( (x \rightarrow y) \), the deletion of dart \( x \) by \( (x \rightarrow \) \), and the insertion of dart \( y \) by \( (\) \( y \rightarrow ) \). Other operations, such as merging and splitting, can be useful in certain applications but are not considered in this paper. Note that an edit operation of a dart \( x \) also refers to the same edit operation performed on the edge \( e_x \), so the edit operation of \( z(x) \) is implied simultaneously. Edit operations on vertices are also implied by edit operations on their darts, i.e. whether a vertex is substituted, deleted, or inserted, depends on the edit operations of its darts. Formally, let \( x \in D_1 \cup \{A\} \) and \( y \in D_2 \cup \{A\} \), and assume an edit operation \( (x \rightarrow y) \) has been executed. We distinguish three cases:

1. \( x \neq A \) and \( y \neq A \), that is \( (x \rightarrow y) \) is a substitution. Then the vertex substitutions \( (h_x \rightarrow h_y) \) and \( (t_x \rightarrow t_y) \) are implied by the dart substitution. So, two dart substitutions, say \( (x \rightarrow y) \) and \( (x' \rightarrow y') \), should fulfill the consistency of the mapping of their head vertices and tail vertices, i.e. \( h_x = h_{x'} \), \( t_x = t_{x'} \). Also, this case implies constraints on the other darts of both \( x \) and \( y \), i.e. the darts \( \sigma(x), \ldots, \sigma^{t(x)−1}(x) (k > 1) \) must be deleted if there is also a dart substitution \( (\sigma(x) \rightarrow \sigma(y)) \), and the darts \( \sigma(y), \ldots, \sigma^{t(y)}(y) \) should be inserted into the source map if it exists a dart substitution \( (\sigma(x) \rightarrow \sigma(y)) \).

2. \( x = A \) and \( y = A \), that is \( (x \rightarrow y) \) is a deletion. Then the vertex deletion \( (h_x \rightarrow \) \) (respectively \( (t_x \rightarrow ) \)) is implied if the vertex \( h_x \) (respectively \( t_x \)) is an isolated vertex after the dart deletion.

3. \( x = A \) and \( y \neq A \), that is \( (x \rightarrow y) \) is an insertion. Then the vertex insertion \( (\) \( A \rightarrow h_y) \) (respectively \( (A \rightarrow t_y) \)) is implied if the vertex \( h_y \) (respectively \( t_y \)) has not been edited before.

Let \( \gamma \) be a cost function that assigns a nonnegative real number \( \gamma(x \rightarrow y) \) to each edit operation \( (x \rightarrow y) \), where \( x \) and \( y \) may be darts or vertices. We constrain \( \gamma \) to be a distance metric as follows:

1. \( \gamma(x \rightarrow y) \) \( \geq 0 \), \( \gamma(x \rightarrow k) = 0 \);
2. \( \gamma(x \rightarrow y) = \gamma(y \rightarrow x) \);
3. \( \gamma(x \rightarrow z) \leq \gamma(x \rightarrow y) + \gamma(y \rightarrow z) \).

Usually edit distance measures are metrics only if the underlying edit operations satisfy the conditions listed above [37]. Given a source map \( G_1 \) and a target map \( G_2 \), a sequence of edit operations \( S = s_1, \ldots, s_k \) that transforms \( G_1 \) completely into \( G_2 \) is called an edit path between \( G_1 \) and \( G_2 \). Note that \( \gamma \) is defined on the whole set of operations and is not specific to two given maps. We extend \( \gamma \) to the edit path \( S \) by:

\[
\gamma(S) = \sum_{i=1}^{k} \gamma(S_i).
\]

Then, the edit distance between two maps \( G_1 \) and \( G_2 \) is defined as follows:

\[
d(G_1, G_2) = n(\gamma(S)|S \text{ is an edit path between } G_1 \text{ and } G_2).\]

To describe the map similarity explicitly, we extend \( \gamma \) to an arbitrary map \( G = (D, x, \sigma, \mu) \) by:

\[
\gamma(G) = \sum_{x \in D} \gamma(x \rightarrow A).
\]

Similar to the definition of graph similarity measure [38], we define a map similarity measure based on map edit distance as:
\[ d(G_1, G_2) = 1 - \frac{d(G_1, G_2)}{\gamma(G_1) + \gamma(G_2)} \]

3.2. Mapping

Mappings were introduced in [27] to describe how a sequence of edit operations transforms one tree into another, ignoring the order in which edit operations are applied. Here, we extend mappings to combinatorial maps.

Given two maps \( G_1 = (D_1, x_1, \sigma_1, \mu_1) \) and \( G_2 = (D_2, x_2, \sigma_2, \mu_2) \), and a triple \((M, G_1, G_2)\) where \( M \) is a set of pairs of darts \((x, y) \in D_1 \times D_2\), we say that \( x \) and \( y \) are touched by a line in \( M \) if \((x, y) \in M\), and that \( x \) follows \( y \) \( x_1 \in D_1 \) and \( x_2 \in D_1 \) in \( M \) if \( x_1 = \sigma^{-1}_1(x_2) \) \((k > 0)\) and none of the darts \( \sigma_1(x_3), \sigma_1^2(x_3), \ldots, \sigma_1^{k-1}(x_3) \) is touched by any line in \( M \).

Definition 4 (Mapping of combinatorial maps). A triple \((M, G_1, G_2)\) is a mapping from map \( G_1 = (D_1, x_1, \sigma_1, \mu_1) \) to map \( G_2 = (D_2, x_2, \sigma_2, \mu_2) \), if \( M \) is any set of pairs of darts \((x, y) \in D_1 \times D_2\), satisfying the following conditions, for all \((x_1, y_1), (x_2, y_2) \in M:\n
1. \( x_1 = x_2 \) iff \( y_1 = y_2 \); (one-to-one),
2. \( x_1 = x_1(x_2) \) iff \( y_1 = x_2(y_2) \); (x involution preserved),
3. \( t_{x_1} = t_{y_2} \) iff \( t_{x_2} = t_{y_2} \), and \( h_{x_1} = h_{y_2} \) iff \( h_{y_1} = h_{y_2} \); (adjacency relationships preserved),
4. \( x_1 \) follows \( x_2 \) iff \( y_1 \) follows \( y_2 \). (orientation order preserved).

We do not distinguish between the triple \((M, G_1, G_2)\) and the set \( M \) if there is no confusion. As a matter of fact, a mapping \( M \) from the source map \( G_1 \) to the target map \( G_2 \) describes the edit operations that allow to transform \( G_1 \) into \( G_2 \). A pair \((x, y)\) indicates the substitution of dart \( x \) by dart \( y \), and a dart \( x \) with no pair \((x, y) \in M \) is deleted from \( G_1 \). For a dart \( y \) in \( G_2 \) not belonging to any pair, we have two cases:

1. It exists at least one pair \((a, b)\) with \( t_b = t_\gamma \). Let \((x_1, y_1)\) and \((x_2, y_2)\) denote two pairs such that:
   - \( t_{x_1} = t_{y_2} \),
   - \( x_2 \) follows \( x_1 \) by a sequence \( \sigma_1(x_1), \ldots, \sigma_1^{k-1}(x_1) \),
   - \( y_2 \) follows \( y_1 \) by a sequence \( \sigma_2(y_1), \ldots, \sigma_2^{k-1}(y_1) \).
   - it exists \( p \leq k-2 \) such that \( \sigma_2^{p}(y_1) = y_2 \).

   \[ \text{Note that we may have } (x_1, y_1) = (x_2, y_2) \text{ if there is only a matched dart in } t_{x_1} \text{ and } t_{y_2}. \]

2. It does not exist such a mapping. Then the hole vertex \( t_y \) should be inserted into \( G_1 \) preserving the orientation.

   \[ \text{Note that an intermediate structure after some edit operations does not guarantee to preserve the combinatorial map (e.g. it may become non-planar after an insertion, or become unconnected after a deletion). Let } \mathcal{I} \text{ and } \mathcal{J} \text{ be the sets of darts in } G_1 \text{ and } G_2, \text{ respectively, not touched by any line in } M. \text{ Then we can define the cost of } M:} \]

\[ \gamma(M) = \sum_{(x, y) \in M} \gamma(x \rightarrow y) + \sum_{x \in \mathcal{I}} \gamma(x \rightarrow A) + \sum_{y \in \mathcal{J}} \gamma(A \rightarrow y). \]

Lemma 1. Let \( M_1 \) be a mapping from \( G_1 \) to \( G_2 \) and let \( M_2 \) be a mapping from \( G_2 \) to \( G_3 \). Then:

1. \( M_1 \cdot M_2 = \{(x, y) \mid \exists z \text{ s.t. } (x, z) \in M_1 \text{ and } (z, y) \in M_2\} \) is a mapping from \( G_1 \) to \( G_3 \);
2. \( \gamma(M_1 \cdot M_2) \leq \gamma(M_1) + \gamma(M_2) \).

Proof.

(1) Let \((x_1, y_1)\) and \((x_2, y_2)\) be two pairs in \( M_1 \cdot M_2 \). Then there exists \( z_1 \) and \( z_2 \) such that \((x_1, z_1), (z_2, y_2) \in M_1 \) and \((z_1, y_1), (z_2, y_2) \in M_2 \). By the definition of a mapping:

- \( x_1 = x_2 \) iff \( z_1 = z_2 \) and \( z_1 = z_2 \) iff \( y_1 = y_2 \);
- \( x_1 = x_2 \) iff \( z_1 = z_2 \) and \( z_1 = z_2 \) iff \( y_1 = y_2 \);
- \( t_{x_1} = t_{x_2} \) iff \( t_{z_1} = t_{z_2} \), and \( t_{z_1} = t_{z_2} \) iff \( t_{y_1} = t_{y_2} \); \( h_{x_1} = h_{x_2} \) iff \( h_{z_1} = h_{z_2} \) and \( h_{z_1} = h_{z_2} \) iff \( h_{y_1} = h_{y_2} \);
- \( x_1 \) follows \( x_2 \) iff \( z_1 \) follows \( z_2 \), and \( z_1 \) follows \( z_2 \) follows \( y_2 \).

Therefore, \( M_1 \cdot M_2 \) is a mapping from \( G_1 \) to \( G_3 \).

(2) Let \( I \) and \( J \) be the corresponding deletion and insertion sets transforming \( G_1 \) into \( G_3 \). There are three general situations occurring for all darts \( x \) in \( G_1 \) and \( y \) in \( G_1 \): \((x, y) \in M_1 \cdot M_2 \), \( x \in I \) or \( y \in J \). In all such cases, the triangle inequality on the distance metric \( \gamma \) ensures that \( \gamma(x \rightarrow y) \leq \gamma(x \rightarrow z) + \gamma(z \rightarrow y) \).

\[ \square \]

Lemma 2. Given \( S \), a sequence \( s_1, \ldots, s_k \), of edit operations from \( G_1 \) to \( G_2 \), there exists a mapping \( M \) from \( G_1 \) to \( G_2 \) such that \( \gamma(M) = \gamma(S) \). Conversely, for any mapping \( M \), there exist a sequence of edit operations, say \( S \), such that \( \gamma(S) = \gamma(M) \).

Proof.

(1) The first part can be proved by induction on \( k \). The base case is \( k = 1 \). This case holds because any single edit operation preserves the adjacency and orientation relationships in the mapping. In the general case, let \( S \) be the sequence \( s_1, \ldots, s_k \), of edit operations. There exist a mapping \( M \) such that \( \gamma(M) = \gamma(S) \). Let \( M_2 \) be the mapping for \( s_k \). From Lemma 1, we have that:

\[ \gamma(M_1 \cdot M_2) \leq \gamma(M_1) + \gamma(M_2) \leq \gamma(S) + \gamma(s_k) = \gamma(S). \]

(2) For the second part of this lemma, let \( M = \{(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)\} \). The sequence \( S \) of edit operations can be constructed by simply performing all the substitutions indicated by the mapping \((x_1 \rightarrow y_1), (x_2 \rightarrow y_2), \ldots, (x_k \rightarrow y_k)\), then all deletions \((x_{k+1} \rightarrow A), \ldots, (x_{2k} \rightarrow A)\), then all insertions \((A \rightarrow y_{2k+1}), \ldots, (A \rightarrow y_{2k})\). Obviously, \( \gamma(S) = \gamma(M) \).

According to Lemma 2, the edit distance:

\[ d(G_1, G_2) = \min \gamma(M) \text{ is a mapping from } G_1 \text{ to } G_2. \]

Hence the search for a minimal cost sequence of edit operations has been reduced to a search for a minimal cost mapping. Mappings ignore the order in which edit operations are applied, so it is easier to construct a mapping \( M \) than a sequence of edit operations \( S \).

3.3. The optimal approach based on \( A^* \) algorithm

Similarly to graph edit distance, we propose in this section a tree search approach to compute map edit distance using the \( A^* \) algorithm. Inner nodes of the search tree correspond to partial...
solutions, and leaf nodes represent complete but not necessarily optimal solutions from the source map to the target map. In order to integrate more knowledge about partial solutions and reduce the search space, a heuristic function is usually used.

Unlike the graph edit distance problem, the mappings of combinatorial maps have strong constraints. So it is necessary to check whether the partial mapping satisfies the conditions in Definition 4 when expanding the search tree. We do not need to check all pairs in the mapping every time we expand the search tree. Checking only the newly inserted pairs is adequate.

Algorithm 2. Check condition function

Input: A partial mapping $M$, and a pair $(x, y)$.  Output: whether the mapping $M \cup [(x, y)]$ satisfies the conditions in Definition 4.

1: for all pair $(x_0, y_0) \in M$ such that $t_x = t_{x0}$ or $t_y = t_{y0}$ do
2: if $(x = x_0)$ or $(y = y_0)$ then return false;
3: end if
4: if $(t_x = t_{x0}$ and $t_y < t_{y0}$) or $(t_x < t_{x0}$ and $t_y = t_{y0}$) then
5: return false;
6: end if
7: if $(h_x = h_{x0}$ and $h_y < h_{y0}$) or $(h_x < h_{x0}$ and $h_y = h_{y0}$) then
8: return false;
9: end if
10: if $(x_0$ follows $x$ and $y_0$ does not follow $y$) or $(x_0$ does not follow $x$ and $y_0$ follows $y$) then
12: return false;
13: end if
14: if $(x$ follows $x_0$ and $y$ does not follow $y_0$) or $(x$ does not follow $x_0$ and $y$ follows $y_0$) then
15: return false;
16: end if
17: end for
18: return true;
End Algorithm.

The Boolean function that checks whether a pair $(x, y)$ can be added to a map $M$ is given in Algorithm 2. From Definition 4, all conditions are only concerned with darts in the same cycle as $x$ or $y$, so we add a constraint in line 1 to filter unconcerned pairs. Lines 2–4 are to check the condition 1, lines 5–10 are to check the condition 3, and lines 11–16 are to check the condition 4. It is not necessary to check the condition 2 here because the edit operations $(x \to y)$ and $(x(x) \to x(y))$ imply each other and are inserted simultaneously into the mapping when expanding the search tree.

Algorithm 3. $A'$ algorithm for map edit distance.

Input: Non-empty combinatorial maps $G_1 = (D_1, x_1, \sigma_1, \mu_1)$ and $G_2 = (D_2, x_2, \sigma_2, \mu_2)$, where $D_1 = \{x_1(1), x_1(2), \ldots, x_{E(G_1)}(1), x_{E(G_1)}(2)\}$ and $D_2 = \{y_1(1), y_1(2), \ldots, y_{E(G_2)}(1), y_{E(G_2)}(2)\}$.  Output: A Minimum cost edit path from $G_1$ to $G_2$.

1: Initialize OPEN to the empty set {}.
2: for all $y \in D_2$ do
3: Insert the substitutions $\{x_1 \to y, x_1(x_1) \to x_2(y)\}$ into OPEN.
4: end for
5: Insert the deletions $\{x_1 \to A, x_1(x_1) \to A\}$ into OPEN.
6: loop
7: Remove $p_{min} = arg\min_{p \in OPEN}(g(p) + h(p))$ from OPEN.
8: if $p_{min}$ denotes a complete edit path then
9: return $p_{min}$ as the solution.
10: else
11: Let $p_{min} = \{x_1 \to y_1, x_1(x_1) \to x_2(y_1), \ldots, x_k \to y_k, x_1(x_k) \to x_2(y_k)\}$.
12: if $k < |E(G_1)|$ then
13: for all $y \in D_2, \{x_1 \to x_2(y), \ldots, x_2(y), x_1(x_2(y))\}$ do
14: if Check_Conditions($p_{min}, x_1(x_k) \to y_2(y)$) = true then
15: insert $p_{min} \cup \{x_1(x_k) \to y_2(y)\}$ into OPEN
16: end if
17: end for
18: Insert $p_{min} \cup x_1(x_k) \to A, x_1(x_1) \to A$ into OPEN
19: else
20: Insert $p_{min} \cup \{y_1(x_1) \to x_2(y_1), \ldots, y_2(y_1), y_1(x_2(y_1))\} \{A \to y\}$ into OPEN
21: end if
22: end if
23: end loop
End Algorithm.

The $A'$ algorithm for map edit distance is described in Algorithm 3. Similar to Algorithm 1 for graph edit distance described in Section 2.2, this algorithm is based on tree searching and guarantees that the complete edit path is always optimal. The Check_Conditions function called in line 14 is described in Algorithm 2. Note that edit operations on vertices are implied by edit operations on their darts. Obviously, implied vertex operations can be derived from every partial or complete edit path during the search procedure. The costs of these implied vertex operations are dynamically added to the corresponding paths in OPEN.

3.4. The approximate approach based on Greedy algorithm

Greedy algorithm for map edit distance described in Section 3.3 guarantees to find an optimal edit path between two maps. However, the computational complexity of this algorithm is exponential regardless a heuristic function is used or not to govern the tree traversal process. The computational and space complexity may be high even for reasonably small maps. Thus, it is difficult to apply this algorithm in real applications. To reduce the computational complexity, finding approximate optimization methods at the cost of acceptable suboptimal solutions is an alternative approach.

A Greedy algorithm can achieve local optimum at each stage in the hope of finding the global optimum. This strategy does not always produce an optimal solution, but as we show in experiments, global optimum frequently occurs in small maps in practice. A Greedy algorithm for the map edit distance is described in Algorithm 4.

Algorithm 4. Greedy algorithm for map edit distance

Input: Non-empty combinatorial maps $G_1 = (D_1, x_1, \sigma_1, \mu_1)$ and $G_2 = (D_2, x_2, \sigma_2, \mu_2)$, where $D_1 = \{x_1(1), x_1(2), \ldots, x_{E(G_1)}(1), x_{E(G_1)}(2)\}$ and $D_2 = \{y_1(1), y_1(2), \ldots, y_{E(G_2)}(1), y_{E(G_2)}(2)\}$.  Output: An approximate solution $S$ from $G_1$ to $G_2$.

// $C$ is a candidate set of edit operations at each stage.
1: Initialize $C$ to the empty set {}.
2: for all $y \in D_2$ do
3: Insert the substitutions $\{x_1 \rightarrow y, \sigma_1(x_1) \rightarrow \sigma_2(y)\}$ into $C$.
4: end for
5: Insert the deletions $\{x_1 \rightarrow A, \sigma_1(x_1) \rightarrow A\}$ into $C$.
6: loop
7: Select the edit operation, say $x_0 \rightarrow y_0$, from $C$ such that
8: $\gamma(x_0 \rightarrow y_0) = \min(\gamma(s) \mid s \in C)$. 
9: Add the edit operations $\{x_0 \rightarrow y_0, \sigma_1(x_0) \rightarrow \sigma_2(y_0)\}$ into $S$.
10: if $S$ denotes a complete edit path then return $S$ as the solution.
11: else
12: Let $S = \{x_1 \rightarrow y_1, \sigma_1(x_1) \rightarrow \sigma_2(y_1), \ldots, x_k \rightarrow y_k, \sigma_1(x_k) \rightarrow \sigma_2(y_k)\}$.
13: if $k < |E(G_1)|$ then
14: Clear $C$ to the empty set {}.
15: for all $y \in D_2(y_1, \sigma_2(y_1), \ldots, y_k, \sigma_2(y_k))$ do
16: if Check_Conditions($S, x_k+1 \rightarrow y$) = true and Check_Conditions($S, x_k+1 \rightarrow \sigma_2(x_k) = y$) = true then
17: Insert $\{x_k+1 \rightarrow y, \sigma_1(x_k+1) \rightarrow \sigma_2(y)\}$ into $C$.
18: end if // corresponds to the if condition in line 16
19: end for
20: Insert $\{x_k+1 \rightarrow A, \sigma_1(x_k+1) \rightarrow A\}$ into $C$.
21: else
22: Set $S = S \cup \{y_1, y_2, \ldots, y_k, \sigma_2(y_k)\}$ \{$A \rightarrow y$\}
23: return $S$ as the solution.
24: end if // corresponds to the if condition in line 13
25: end if // corresponds to the if condition in line 9
26: end loop
End Algorithm.

As shown in Algorithm 4 (line 14), the candidate set $C$ needs to be reconstructed after selecting a minimum cost edit operation at each step. There are $n$ (where $n$ is $|D_2|$) edit operations in set $C$ at most, and each edit operation should be checked by the Check_Conditions function. Using an appropriate vertex labeling function $\mu$ which maps all the darts of a same cycle to a same value, this function can be done in $O(d_3 + d_2)$ time (where $d_1$ and $d_2$ are the maximal degree of $G_1$ and $G_2$ respectively). So the reconstruction of set $C$ can be finished in $O(n(d_3 + d_2))$ time, and the computational complexity of Algorithm 4 is $O(n + (d_3 + d_2))$ in the worst case (where $m = |D_1|$). As we show in experiments, the average complexity is between quadratic and cubic.

4. Experiments

In this section, we present the experimental results in three parts. The first part (Section 4.1) compares the performance of the two algorithms (the $A^*$ algorithm and the Greedy algorithm presented in Section 3) for the map edit distance problem. The second one (Section 4.2) illustrates the improvement of the error-tolerant algorithm for map matching in comparison with the exact matching algorithms. The third one (Section 4.3) demonstrates the benefits of our approach comparing to an efficient approach of graph edit distance [39]. All experiments are performed on the platform of Pentium 2.80 GHz CPU with 4.0G memory.

4.1. Experiment 1

The $A^*$ algorithm and the Greedy algorithm for the map edit distance problem were implemented and used to compute the edit distance between two groups of random maps. Here, we denote the distance computed by $A^*$ algorithm and Greedy algorithm as $A^*$ distance and Greedy distance respectively.

Numbers of darts in the maps vary from 10 to 30. For each case, we create 500 pairs of random maps using the algorithm presented in [41], and compute the $A^*$ distance and Greedy distance for each pair of maps respectively. Then, the statistical averages of $A^*$ distance and Greedy distance are obtained. In each map, we assign each vertex a label as a random integer from 0 to 20, and assign each dart a label as a random integer from 0 to 10. The cost of insertion and deletion of a vertex or a dart equals to its label, and the cost of substitution between two vertices or two darts equals to the absolute difference between the two labels. For computational simplicity, we assign the same label to the two darts sewn from the same edge.

The comparison of the computational time of the $A^*$ algorithm and the Greedy algorithm is shown in Table 1. The computational time of the $A^*$ algorithm increases remarkably with the number of darts in the maps. The running time of this algorithm is huge even for reasonable small maps. Also, as shown in Table 2, the search tree of this algorithm requires exponential memory. For larger maps (e.g. the number of darts exceeds 30), the search tree exceeds the memory capacity of our testing machine. So, it is hard to apply this algorithm in real applications due to its high running time and memory complexity. Compared with the $A^*$ algorithm, the computational time and required memory of the Greedy algorithm varies relatively slowly with the number of darts in the maps. It is obvious that the Greedy algorithm is much efficient than the $A^*$ algorithm.

Fig. 2 shows the comparison of the $A^*$ distance and the Greedy distance. As shown in Fig. 2a, both the $A^*$ distance and the Greedy distance increase linearly with the increase of the number of darts in maps. Fig. 2b shows the relative difference $d_R$ between $A^*$ distance and Greedy distance as:

$$d_R = \frac{2 \times (d_c - d_e)}{(d_c + d_e)}$$

where $d_c$ and $d_e$ denote Greedy distance and $A^*$ distance respectively. Obviously, the relative difference $d_R$ is approximately constant with the number of darts in maps. The Greedy algorithm achieves local optimum at each stage and does not guarantee a global optimal solution. However, as we can see in Fig. 2c, for small maps, it sometime reaches the global optimal solution, i.e. the Greedy distance frequently equals to the $A^*$ distance in small maps. It fails to reach the optimality in considerable large maps.

4.2. Experiment 2

The database used in this experiment includes 50 images selected from the database of free images for research purpose, which are available on Internet [http://www.cs.washington.edu/research/imagedatabase]. These 50 images are selected from 10 categories (five images per category): arborgreens (ARB), cannon-beach (CAN), cherries (CHE), colombiagore (COL), football (FOO), greenlake (GLR), Greenland (GRD), leafesstrees (LEA), swissmoutains (SWI), Yellowstone (YEL). Five images in each category are captured from the same scene at different time or from different angles, so they have similar colors and contents but are not exactly equal. We first segment images using Mean Shift algorithm [42], and then extract combinatorial maps based on segmented regions:

- Regions are extracted as vertices. A 24-dimensional color histogram of each region is assigned as its label.
- The adjacency between regions are extracted as darts preserving orientation orders. The number of adjacent pixels between each two adjacent regions is assigned as corresponding dart label.
The average number of vertices and darts in the extracted maps are 81 and 336 respectively. Fig. 4 shows examples of two categories of source images, segmented regions and extracted maps.

We randomly select one image from each category as a query image (10 query images in total), and search relevant images for each query image from the remaining 40 images. The searching results are conducted by comparing corresponding maps using exact matching and inexact matching approach respectively. In exact case, we extend the algorithm for submap isomorphism presented in [20] to labeled maps, and two images are considered as "matched" if one extracted map is subisomorphic to the other. In inexact case, for each query image, we compute the Greedy distance and the similarity factor $d$ (see Section 3.1) between it and the remaining 40 images, and select four images with most maximal $d$ as "matched".

Ideally, each query image should match the four images in the same category and reject all the images in other categories. Actually, in exact case, no "matched" image is found for each query image despite there are some images in the same category as it having similar colors and contents. For example, although the five

<table>
<thead>
<tr>
<th>The number of darts</th>
<th>10</th>
<th>12</th>
<th>14</th>
<th>16</th>
<th>18</th>
<th>20</th>
<th>22</th>
<th>24</th>
<th>26</th>
<th>28</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>A* algorithm (ms)</td>
<td>&lt;1</td>
<td>2</td>
<td>7</td>
<td>17</td>
<td>43</td>
<td>165</td>
<td>429</td>
<td>1408</td>
<td>8531</td>
<td>24510</td>
<td>56316</td>
</tr>
<tr>
<td>Greedy algorithm (ms)</td>
<td>0.03</td>
<td>0.03</td>
<td>0.04</td>
<td>0.04</td>
<td>0.05</td>
<td>0.06</td>
<td>0.07</td>
<td>0.09</td>
<td>0.10</td>
<td>0.10</td>
<td>0.11</td>
</tr>
</tbody>
</table>

Table 1

The average computational time of A* algorithm and Greedy algorithm.

<table>
<thead>
<tr>
<th>The number of darts</th>
<th>10</th>
<th>12</th>
<th>14</th>
<th>16</th>
<th>18</th>
<th>20</th>
<th>22</th>
<th>24</th>
<th>26</th>
<th>28</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>A* algorithm (MB)</td>
<td>&lt;1</td>
<td>&lt;1</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>15</td>
<td>37</td>
<td>96</td>
<td>245</td>
<td>627</td>
<td>1.683</td>
</tr>
<tr>
<td>Greedy algorithm (MB)</td>
<td>0.08</td>
<td>0.12</td>
<td>0.15</td>
<td>0.16</td>
<td>0.17</td>
<td>0.18</td>
<td>0.19</td>
<td>0.21</td>
<td>0.22</td>
<td>0.23</td>
<td>0.24</td>
</tr>
</tbody>
</table>

Table 2

The size of memory used by A* algorithm and Greedy algorithm.

Fig. 2. Comparison of the edit distance and the Greedy distance.

![Diagram](image-url)
images in the CHE category shown in Fig. 3 have high similarities, the remaining four images are still rejected by the query image CHE 49 because they are captured at different time and are not exactly equal.

In inexact case, most similar images in the same category are matched. The searching result is shown in Table 3. There are 27 good matches and 13 bad matches. The searching result lies significantly on the segmented regions of images. As shown in Fig. 3, although the five images in the ARB category have similar colors and contents, the segmented regions and corresponding extracted maps vary significantly, so there are two bad matches of the image ARB 39. On the contrary, the image CHE 49 matches all the
4.3. Experiment 3

In this section, we compare the proposed Greedy algorithm with an efficient algorithm for edit distance computation of planar graphs proposed by Neuhaus and Bunke [39]. Since the resulting edit path of Neuhaus's algorithm strongly depends on the seed substitution, we compute \( K \) times with different seed substitutions and choose the one with minimum edit cost.

We first make the experiment on a set of random maps of which the numbers of darts vary from 50 to 600. The method of assigning labels to maps is the same as that in Experiment 1 (Section 4.1). For each case, we create 500 pairs of random maps, and compute the edit paths for each pair of maps by the proposed Greedy algorithm and Neuhaus's algorithm respectively. The costs of the resulting edit paths are shown in Table 5, in which \( K \) denotes the number of seed substitutions. From this table, the resulting edit paths of Neuhaus's algorithm are optimized significantly with \( K \) increasing when \( K \) is small, and the changes are unremarkable when \( K \) is relatively big (e.g. when \( K > 30 \)). Obviously, the resulting edit path of the proposed Greedy algorithm is more close to the optimal solution than that of Neuhaus's algorithm. Fig. 4 shows the comparison of the computational time of both algorithms. Neuhaus's algorithm has much higher computational efficiency than the proposed Greedy algorithm.

We also perform the experiment on a real image database used in [40], which is available on the web site http://wang.ist.psu.edu/docs/related/. This dataset consists of 1000 images equally divided into 10 groups, which are aborigines, beach, ancient building, bus, dinosaur, elephant, flower, horse, mountain, and foods in the dish, respectively. We extract combinatorial maps from images based on segmented regions using the same method as in Experiment 2 (Section 4.2). We randomly select 10 images from each group as query images (100 query images in total), and retrieve relevant images for each query image by the proposed Greedy algorithm and Neuhaus's algorithm respectively. Suppose a query image \( q \) is given and it returns a set of images \( A(q) \) as the answer. Let \( R(q) \) denote all the images in the same group as the query image \( q \). The precision of the answer is the fraction of the returned images that is indeed relevant for the query:

\[
p = \frac{|R(q) \cap A(q)|}{|A(q)|}.
\]

Fig. 5 shows the comparison of the mean precision of the top \( N \) \( (N \) ranges from 5 to 60) results for both algorithms. The precision of both algorithms decreases with \( N \) increasing, and the changes are remarkable when \( N \) is relatively small. It is shown that our approach provides higher precision than Neuhaus's algorithm. This is due to that the proposed Greedy algorithm provides better solution than Neuhaus's algorithm as shown in Table 5. From this experiment, our approach provides better results than Neuhaus's algorithm in both artificial databases and real image databases, so it is suitable for image retrieval applications.

<table>
<thead>
<tr>
<th>Number of darts</th>
<th>Neuhaus's algorithm</th>
<th>The proposed Greedy algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( K = 1 )</td>
<td>( K = 10 )</td>
</tr>
<tr>
<td>100</td>
<td>1052</td>
<td>844</td>
</tr>
<tr>
<td>200</td>
<td>1698</td>
<td>1564</td>
</tr>
<tr>
<td>300</td>
<td>2551</td>
<td>2387</td>
</tr>
<tr>
<td>400</td>
<td>3277</td>
<td>3130</td>
</tr>
<tr>
<td>500</td>
<td>4094</td>
<td>3940</td>
</tr>
<tr>
<td>600</td>
<td>4866</td>
<td>4716</td>
</tr>
</tbody>
</table>

Fig. 5. The mean precision of the top \( N \) results for the proposed Greedy algorithm and Neuhaus's algorithm.

remaining 4 images in the CHE category because the segmented regions and corresponding extracted maps in the CHE category are more visually similar.

To build the confusion matrix of this database, we classify each image according to the five nearest neighbors returned by the proposed Greedy algorithm. The confuse matrix is shown in Table 4. Most images are well classified, and the overall accuracy is 82%.

In real world applications, it is usually necessary to match objects with small differences. From the experiments, the exact technique is intolerant to structural differences. On the contrary, the inexact map matching approach is able to handle dissimilarity between patterns and produces better search results. So the proposed inexact matching approach can be used in broader applications than the exact matching techniques.

5. Conclusion

This paper addressed the problem of error-tolerant map matching. We extended the concept of edit distance to combinatorial maps, which can be used to measure the distance or the similarity between two maps. We defined the mapping of maps to describe how a map can be transformed into another. Then we showed that the search for a minimal cost sequence of edit operations can be reduced to a search for a minimal cost mapping. Furthermore, we proposed an optimal approach to compute map edit distance based on tree search, with exponential computational complexity. To reduce the computational complexity, we subsequently proposed an approximate approach based on Greedy algorithm. The Greedy algorithm gains high computational efficiency at the cost of that it usually fails to produce optimal solution for large maps.

The problem of comparing maps is important, and some works for exact map matching have been presented. In real applications, we always need to find similar patterns while tolerating little difference. As shown in the experimental results in Section 4.2, the exact approach can find only exactly equal patterns, while the
inexact approach presented in this paper is more effective for real applications.

In the future, we plan to extend this work to reduce the computational complexity on the premise of guaranteeing optimal solution, and to optimize the solution obtained by the approximate approach.

Acknowledgments

This work is supported by the Open Foundation from Computer Application Technology in the Most Important Subjects of Zhejiang (No. 1020001), the National Nature Science Foundation of China (Nos. 61272352 and 61100142), the Jiangsu Provincial Natural Science Foundation of China (No. BK20111454), the Fundamental Research Funds for the Central Universities (Nos. 2012JBM040 and 2011JB218), and the Human Sixth Sense Programme at the Advanced Digital Sciences Center from Singapore’s Agency for Science, Technology and Research (A*STAR).

References