Map Stitcher: Graph Sampling-based Map Conflation

Erfan Hosseini Sereshgi
Tulane University
New Orleans, Louisiana, USA
shosseinisereshgi@tulane.edu

Carola Wenk
Tulane University
New Orleans, Louisiana, USA
cwenk@tulane.edu

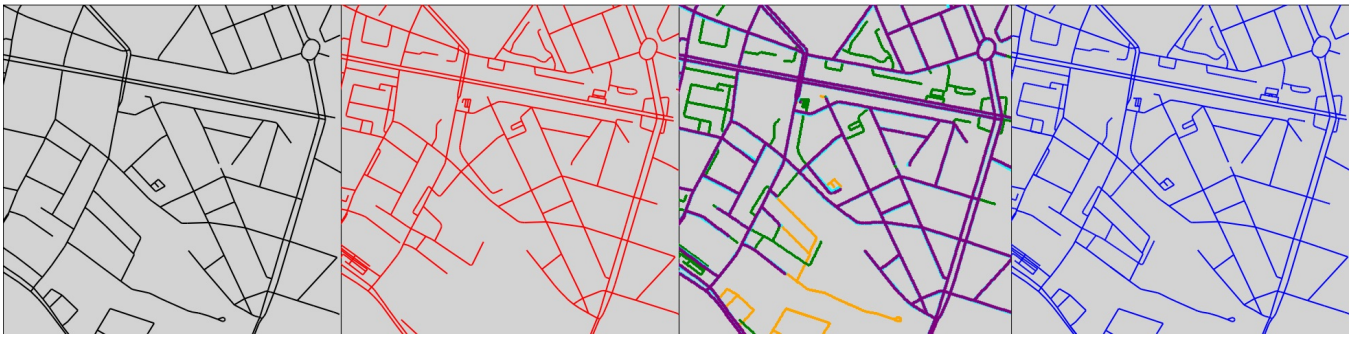

Figure 1: The Map Stitcher workflow integrates a base map (black) and a supplementary map (red) through graph sampling, identifying and visualizing correspondences (cyan and purple) and inconsistencies (orange and green). The algorithm then reconciles these differences, generating a comprehensive merged map (blue).

Abstract

The integration of two geometric graphs is a fundamental task that arises in various fields, including automated cartography, image processing, and computer graphics. Maintaining precise and up-to-date roadmaps may pose a significant hurdle for new companies, especially in the early stages when resources are limited. A notable application of this task is the process of map conflation. Map conflation or map merging involves combining roadmap data from two separate sources to create data with higher coverage and accuracy than either source which is essential for accurate navigation systems. An ideal solution would be a scalable automated technique capable of handling vast areas while safeguarding key geometric and topological details.

In this work, we explore the application of graph sampling, a common method for evaluating reconstructed maps, to the task of map conflation. Our approach employs a partial matching technique, allowing segments to be matched fractionally through graph sampling. Unlike existing methods, which require a segment to be either fully matched or not matched at all, our technique enables nuanced matching. This leads to more detailed conflated maps, avoiding excessive selectivity when adding edges and more customization. Hence, we introduce *Map Stitcher*, an automated tool that seamlessly integrates new information from a secondary map into a primary one. Our approach offers adaptable algorithms that address various scenarios encountered in roadmap data, allowing for tailored choices based on the specific datasets along with source code and sample datasets. Furthermore, we evaluate our method on three roadmap datasets, demonstrating its effectiveness in maintaining accuracy.

CCS Concepts

• **Information systems** → **Geographic information systems**; • **Theory of computation** → *Design and analysis of algorithms*; Computational geometry.

Keywords

Map Comparison, Map Conflation, Computational Geometry, Geographical Information Systems

ACM Reference Format:
Erfan Hosseini Sereshgi and Carola Wenk. 2024. Map Stitcher: Graph Sampling-based Map Conflation. In *3rd ACM SIGSPATIAL International Workshop on Spatial Big Data and AI for Industrial Applications (GeoIndustry'24), October 29-November 1, 2024, Atlanta, GA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3681766.3699604

1 Introduction

Merging geometric graphs is a process that can arise in a variety of situations. When new map companies or ride apps are founded, they often do not possess a roadmap of their own. Most use freely available maps from OpenStreetMap (openstreetmap.org) as base maps and over time improve their networks by using different sources like GPS data, satellite images or simply a roadmap from another data provider. This requires comparing a new road network to a base roadmap, and later adding it to the base roadmap. Some parts of the new road network might already be present in the base roadmap, some might be missing, and some might be partially

GeoIndustry'24, October 29-November 1, 2024, Atlanta, GA, USA

ACM ISBN 979-8-4007-1145-9/24/10
https://doi.org/10.1145/3681766.3699604

present or have different geometries. For example, the exact locations of the centerline representations of streets often differ, as do locations of street intersections and highway ramp connections. Map conflation typically involves a two-stage process: first, aligning the input roadmaps through matching, and then incorporating new features based on the established correspondences. In this paper we consider a roadmap to be a geometric graph in the plane, i.e., the graph is given together with a planar embedding which maps vertices to points and edges to line segments. We note that some of the approaches we discuss may also apply to immersed graphs which allow transversal edge intersections and hence are able to model bridges.

Fairly recently, several methods have been proposed to compare such geometric graphs, including edit distances and Fréchet-based distances; see [3] and [8] for surveys. Some but not all of these distance measures return a correspondence between the two graphs that indicate which parts of graph G correspond to graph H. Naturally, such correspondence will be beneficial for merging graphs. However, the majority of these measures come with a high computational price tag, with some being NP-complete, making them inefficient and impractical for large-scale applications. Here, we study one particular method for comparing geometric graphs, the graph sampling method [6, 7], and we present an algorithm for using the correspondence computed by this distance to merge two input roadmaps from the same area.

Being the most popular method for evaluating reconstructed maps because of its simplicity, speed and effectiveness in comparing geometric properties, the graph sampling method has been used for almost a decade. Graph sampling was first introduced by Biagioni and Eriksson [6, 7] to evaluate their results and compare their approach with the state of the art map construction algorithms. Even though it has been wrongly called a "distance" on some occasions, graph sampling is a statistical method which discretizes 2D immersed graphs into sample points and matches the samples instead. The essential idea is to first compute a set of point samples on each map, and then to match pairs of samples — one from each map — via a one-to-one matching. See our previous work [2] for details on the different design choices for implementing graph sampling and determining a matching.

For deciding whether two samples can be matched, different criteria, e.g., based on distance or orientation, can be used. Samples are placed on edges of the graph starting from a random point on each connected component following a certain interval between each consecutive pair of samples. After choosing a matching distance threshold $d_{\max}$, these samples are matched one-to-one and counted for the evaluation. The matching distance threshold only allows the samples within that distance to be matched to each other. Additionally, graph sampling takes into account the bearing conditions such that if the bearing difference of two edges is more than a certain threshold, the samples on these edges cannot be matched even if they comply with the matching distance threshold. Figure 2 shows an overview of this method.

Throughout this article, we refer to the two inputs, base map and the supplementary map as G and H, respectively.

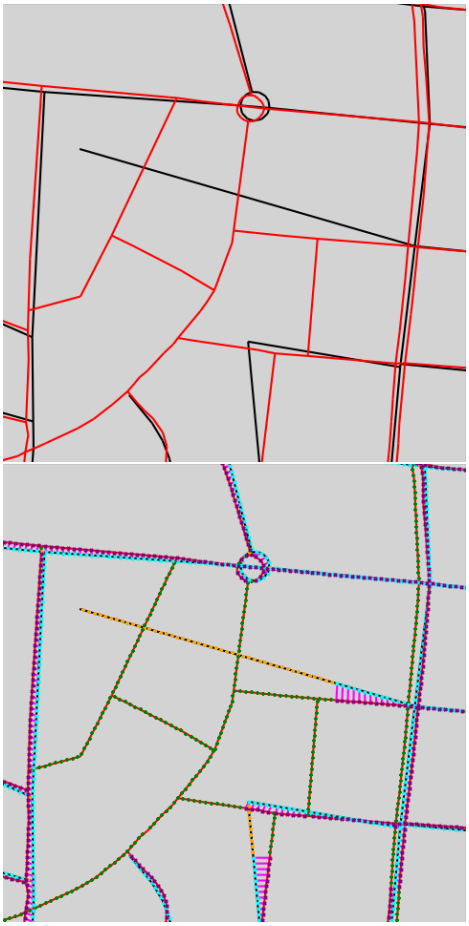

Figure 2: Two roadmaps in black and red and their graph sampling results: matched points are connected by pink line segments, unmatched points are green or orange.

1.1 Related Work

Research in this field can be broadly classified into two categories: map update and map conflation. While map update involves deriving new road information from GPS data or aerial images and integrating it into an existing base map, our work focuses on the map conflation domain, exploring the task of combining two entire road networks.

Several research papers consider the *map update* problem and leverage trajectory data to infer and add missing roads to the base map, such as [13] and [15]. In particular, [16] employs Hidden Markov matching to align trajectories with the road network. Additionally, aerial imagery has been used for updating road networks [4].

Since the 1980s, geospatial conflation has been a topic of extensive research, driven by the need to address the inherent inconsistencies between datasets [12]. These inconsistencies arise from differences in data representation, levels of detail, and spatial displacement patterns, which are introduced during surveying and

cartographic processes. Many articles utilize optimization and network flow models to add new features from a secondary roadmap to a primary one [9] [10] with [11] emphasizing on topological features. Furthermore, [1] adopts classic rank join in their map conflation approach and [17] uses a fuzzy logic inference technique.

Lastly, [14] addresses the map conflation problem in the context of map updating, proposing an algorithm that combines a reconstructed map with a roadmap, with a particular focus on precisely identifying road closures.

1.2 Problem Definition

Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two connected geometric graphs embedded in the plane. That means, each vertex is embedded at a point in the plane, and an undirected edge is embedded as the straight line segment between the points. We think of these as two roadmaps.

For the general map merging problem we wish to compute a *merged* roadmap, i.e., a geometric graph G' such that G' contains all of G and it contains the portions of H that are not yet in G. Equivalently, we can determine the portions of H that are already contained in G and then insert the remaining portions of H into G to obtain G'.

The problem then reduces to finding a *matching* between G and H that determines the portions of H that are already contained in G. Ideally we would like a (partial) one-to-one matching from G to H that is as large as possible and that preserves the length as much as possible. In practice of course one has to account for noise, so matching portions of G and H won't have exactly the same geometry or length. At the same time we want G' to capture as much of the geometric and topological information of G and H as possible.

In this paper we compute a matching using the graph sampling method for comparing two graphs [2, 6, 7] by placing point samples at regular intervals on each graph and then computing a greedy matching as described in [2]. The greedy matching algorithm comprises two stages:

(1) Initial Assignment: Each sample in graph G is assigned its nearest neighbor in graph H, resulting in a non-injective (many-to-one) assignment.
(2) Refinement: The matched pairs are sorted by their distance, and a one-to-one matching is computed greedily, starting with the closest pair. If a sample in H is already matched, the algorithm searches for the 10 nearest unmatched samples in H and selects the closest one. If no suitable match is found within the distance and bearing thresholds, the sample in G remains unmatched.

2 Method

We begin our algorithm by placing sample points on edges of both G and H, starting from a random vertex of each connected component, and placing samples at equal distance Δ. We call these two sets of samples S_G and S_H, respectively. We then proceed to match each sample s_G on G to a sample s_H on H using the greedy matching and a matching threshold of $d_{\max}$ and bearing difference threshold of θ as described in [2]. To ensure we capture the maximum amount of geometric and topological information, we need to store additional data structures. These structures will manage the samples associated with each edge. Specifically, we use hash tables to manage sample points on input graphs, linking edges to their (ordered) samples and enabling constant-time edge lookup by sample coordinate. As the algorithm progresses, two additional data structures track matchings:

(1) A matching table for samples between graphs
(2) A dictionary storing the matched sample percentage for each edge in the second graph.

In order to identify edges in H that need to be added to G, we use a threshold ρ on the percentage of matched samples for each edge in H which is adjustable.

2.1 Intersections

Since the edges of the input graphs are straight line segments, curved roads are typically made up of a series of connected edges. Thus, not all vertices of the input graphs are road intersections. Accurately identifying intersections is crucial because they reveal missing portions in the roadmap and can be used as anchor points during the merging process. To achieve this, we first establish a definition of intersections that aligns with our specific goals:

We consider any vertex on the map an *intersection* if it connects more than two roads (has a degree greater than 2). Additionally, vertices that connect exactly two roads (degree 2) are considered intersections if the roads entering and exiting differ significantly in direction (bearing difference of θ degrees or more). And finally, Vertices with a degree of 1 (dead ends) are also considered intersections.

Since roadmaps typically form a single *connected* network (i.e., one can travel between any two points), and based on our definition of intersections, **any missing portion that needs to be added to G can be identified as a path connecting two intersections**. In general, the intersections in G and H won't be contained in the sample sets s_G and s_H. Therefore we proceed to compute another partial one-to-one-matching between the intersections in G and those in H. For this we use the greedy matching algorithm again with the matching threshold $d_{\max}$.

Now those intersections in H that are *not* matched to an intersection in G are of two types:

(1) New intersection i_H whose corresponding point i_G falls on a road in G.
(2) Completely new intersection i_H that doesn't exist in G, including its connected roads.

In Section 2.2 we show how to identify type (1) intersection in H and how to add them to G. Intersections of type (2) will be added implicitly while adding new paths as described in Section 2.3.

2.2 Adding New Intersections

Here, we provide an algorithm for identifying type (1) intersections in H and for adding them to G. This requires accuracy, because they need to align perfectly with a road segment already present in G. So we check each unmatched intersection i_H in H to see if it can be matched to a point on an edge of G. In terms of edges, this means one [1] or two edges in H incident on i_H can be matched to edges in G.

[1] This can happen when i_G is a degree-2 (turn) intersection.

In theory, both graphs G and H could contain various artifacts, such as roads that are very close together, or intersections that are very close to a road. Although such artifacts are not completely avoidable, roadmaps tend to be more well-behaved due to the practical nature and width of roads. Our description below attempts to handle some of these artifacts using bearing difference θ and threshold ρ. Figure 3 shows an example of a potential artifact.

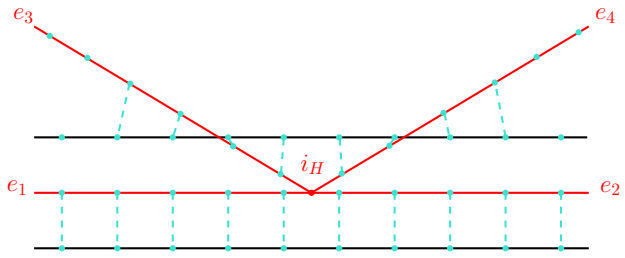

Figure 3: Two roadmaps G and H in black and red and their samples in cyan. The two edges in G are very close together (roughly $2d_{\max}$). In addition, the percentage of matched samples on edges e_3 and e_4 is only $66\% < \rho$.

For each unmatched intersection i_H we proceed as follows: We check all edges in H incident on i_H whether the percentage of its samples that are matched is at least $1-\rho$. Intuitively, these edges are matched well as they have many witnesses. For each of these edges we look up in constant time the sample closest to i_H (each edge stores a sorted list of samples), and we only keep those samples that are matched with a sample in S_G. We now consider the following cases:

- If there are zero such samples, i_H is an intersection of type (2), and we do not proceed further with i_H for now.
- If there is one such sample s_H, then i_H is of type (1). We mark the sample s_G that is matched with s_H as an intersection and match it with i_H.
- If there are two such samples s_H, s'_H then let s_G, s'_G be the matching samples in G, and e_G, e'_G their edges in G. Only if the bearing difference between e_G, e'_G is at most θ, we keep i_H as an intersection of type (1). If $e_G = e'_G$ then s_G and s'_G lie on the same edge, see Figure 4, and we create a new intersection i_G by averaging the locations of s_G and s'_G. In this case we also have to split the edge e_G into two separate edges that are then connected by the new intersection i_G. This takes linear time in the number of samples on the edge. If $e_G \neq e'_G$, then we pick the first vertex in G on a path from s_G to s'_G, mark it as an intersection (if it isn't already), and match it with i_H. This only takes constant time.
- If there are more than two such samples, we assume this is an artifact and ignore i_H for now.

2.3 Adding New Paths

In this section, we explore the general structure of our traversal algorithm, examining the various types of edges it encounters and the strategies for handling each one. Due to the connectivity property, we assert that every type (2) intersection has a path to a matched intersection in H. Consequently, for each matched intersection i_H on

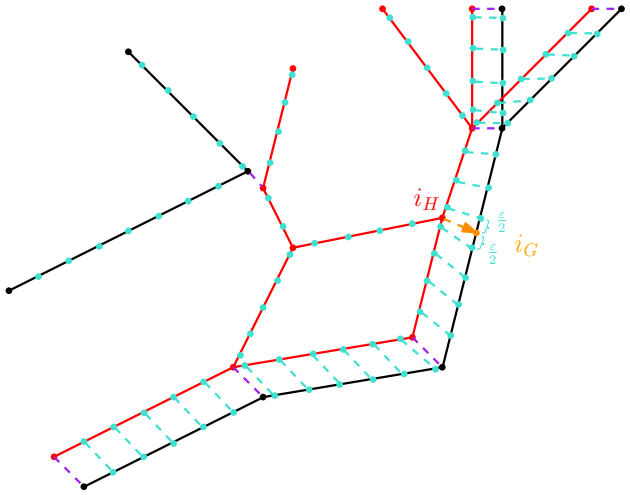

Figure 4: Two roadmaps G and H in black and red and their samples in cyan. Purple dashed lines indicate a matching between two intersections. i_H is a type (1) intersection and its corresponding point i_G on G is shown in orange. Δ is the sample interval.

H, we verify whether all adjacent edges are matched by examining the matching status of the first sample and the overall percentage of matched samples on each adjacent edge, which we stored in dictionaries during the initial graph sampling step.

If an adjacent edge (or multiple edges) connected to i_H has a matching percentage below a specified threshold ρ, the algorithm initiates a slightly modified breadth-first search (BFS) on H, starting from i_H. This search explores every connected edge, identifying paths not present in the base map and incorporating them through a process known as 'stitching'. The BFS continues until all eligible adjacent edges have been examined or it reaches another matched intersection, at which point it terminates. The matching distance and direction between the starting intersection i_H and its corresponding intersection on G define a vector $\vec{v}$ which specifies the magnitude and direction of the shift required to align the remaining samples in the path. Figure 5 shows an example of stitching, where three roads and a type (2) intersection are connected, starting from a matched intersection i_G and ending at two other matched intersections.

Traversing new paths using breadth-first search: We initiate a (FIFO) queue with vertices v_H, starting from a matched intersection i_H, and perform a Breadth-First Search (BFS) until the queue is empty. In parallel, we maintain a secondary queue to keep track of the most recently visited vertex v_G, updating it simultaneously with the main queue to ensure we have the latest vertex in the path. Additionally, we utilize a dictionary to mark visited edges in E_H. As the BFS progresses, we take specific actions based on the next vertex to be visited. As previously stated, we only consider edges with a matching percentage below the designated threshold ρ for addition. However, there is one notable exceptions to this rule, which we will discuss in detail, as a special case.

(1) *The next vertex v_H is not an intersection and the edge e_H is not visited*: This scenario is the most common, particularly in areas where roads are curved or intersecting. If the edge e_H is too short to contain a sample, we simply add the other endpoint of edge e_H to the queue (along with vertex v_G) to the queue and proceed to the next iteration. However, if e_H contains samples we call *Stitch* to add e_H to the base map G.
(2) *The next vertex v_H is an intersection but is not matched to an intersection in G and the edge e_H is not visited*: We use Stitch to add e_H to the base map G. Since v_H represents a new intersection, we must incorporate it into G. To do so, we identify the corresponding position in G using the vector $\vec{v}$, and establish a connection between v_G and this newly matched intersection. Finally, we add v_H to the queue and the new intersection becomes v_G. Similarly to the previous case if the edge lacks a sample, we bypass calling the Stitch function.
(3) *The next vertex v_H is a matched intersection and the edge e_H is not visited*: We utilize *Stitch* again if there are samples on e_H. This case prunes the BFS tree. We do not explore this branch further and therefore we do not add v_H to the queue. Nevertheless, we still need to link the last identified vertex v_G to the corresponding matched intersection on G.
(4) *The edge e_H is visited*: This indicates the presence of a cycle, so we complete the cycle by linking v_G to the last matched sample on this edge, thereby closing the loop.

Adding a path via Stitching: The *Stitch* function constructs vertices in the graph G based on unmatched samples along an edge e_H and connects them sequentially. If the function encounters already matched samples, it connects the newly generated portion to the existing matched portion of the edge. Furthermore, whenever the function visits a matched sample, it updates the vector $\vec{v}$. See Algorithm 1 for more details.

Figure 6 illustrates the same example during the process of adding the final edge to the base map.

A special case: In rare instances, the final edge(s) used to link the new path to an intersection may already exist in the base map, potentially resulting in partial or complete matching. These pre-existing edges manifest as degree-1 intersections in the base map G, indicating their connection to a single adjacent edge. To avoid disrupting the flow of the base map, we need to take careful steps to maintain the local connectivity within the base map. Therefore in cases (1) and (3), we also consider adding edges with a matched sample percentage exceeding ρ, provided they meet the following conditions:

(1) They are not initial edges in the BFS tree (not directly connected to the root node): This helps to eliminate small, isolated segments near curved intersections, which can appear as tiny dead ends
(2) Following the BFS direction, these edges point outwards from already matched samples towards unmatched samples: This ensures that edges are not added in isolation or incorrectly positioned, maintaining the connectivity and accuracy of the graph

As illustrated in Figure 7, e_G, which is partially matched with edge e_H from map H, is an additional edge in G. Note that the endpoint of

Algorithm 1: Stitch

Input : An edge $e_H \in E_H$, its endpoint v_H, and its ordered list of samples $S^e_H \subseteq S_H$. A dictionary M_H connecting each sample s_H to its corresponding matched sample s_G or 0 if it is not matched. The last identified vertex $v_G \in V_G$ and the vector $\vec{v}$
Output : The next identified vertex $v_G \in V_G$

```
stitches ← 0
flag ← False
for all s_H ∈ S^e_H :
    if M_H[s_H] = 0:
        newpoint ← M_H[s_H] + v⃗
        if v_H is an intersection and
          bearing(v_G, newpoint) ≁ bearing(e_H):
            // If the ending point of e_H is an intersection,
            avoid extending the road too much and passing
            the intersection.
            break
        Add newpoint as a vertex to G
        Add an edge to G between v_G and newpoint
        M_H[s_H] ← newpoint
        stitches ← stitches + 1
        v_G ← newpoint
        flag ← False
    else
        newpoint ← M_H[s_H]
        if bearing(v_G, newpoint) ≁ bearing(e_H):
            // Avoid adding ill-positioned matched points near
            the intersections
            continue
        v⃗ ← the vector from s_H to M_H[s_H]
        if flag = False:
            // Do not stitch over already matched portions
            Add newpoint as a vertex to G
            Add an edge to G between v_G and newpoint
            v_G ← newpoint
            flag ← True
Update matched sample percentage for e_H using stitches
return v_G
```

e_G, intersection i_G with degree 1, is not matched to the intersection i_H due to their distance. In this case, the Breadth-First Search (BFS) algorithm does not recognize e_H as a new edge since half of its samples are already matched, resulting in the failure to connect the corresponding point of i_H to i_G.

2.4 Runtime Analysis

Greedy matching requires the construction of a R-tree to store the samples of H. Assuming m and n are the number of samples on G and H, respectively, constructing the R-tree takes $O(n \log n)$. For each sample on G we do at most 11 queries on the R-tree which results in total runtime of $O(m \log n)$.

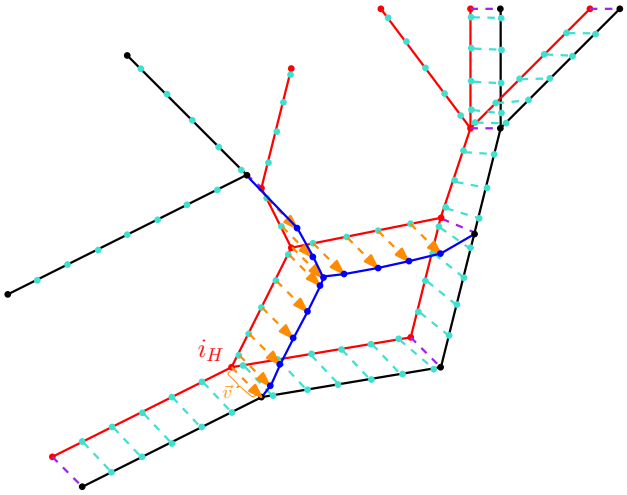

Figure 5: Two roadmaps G and H in black and red and their samples in cyan. Purple dashed lines indicate a matching between two intersections. i_H is a matched intersection but is missing an edge. The orange vector $\vec{v}$ is used to find the new vertices' locations. The added parts are shown in blue.

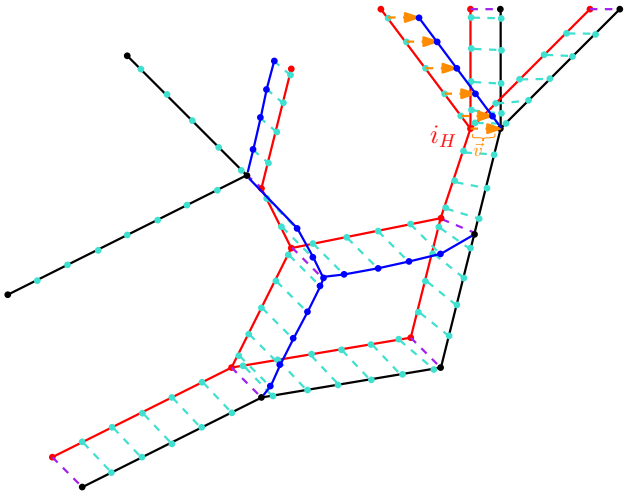

Figure 6: Two roadmaps G and H in black and red and their samples in cyan. Purple dashed lines indicate a matching between two intersections. i_H is a matched intersection but is missing an edge. The orange vector $\vec{v}$ is used to find the new vertices' locations. The added parts are shown in blue.

When adding new intersections of type 1, as described in Section 2.2, most of the involved operations take constant time. Summing these operations, including the check over all edges incident to the intersection, yields time linear in the complexity of the graph H. However, the case where $e_G = e'_G$ requires splitting the list of samples on an edge, which in an (unrealistic) worst-case scenario could result in many repeated splits on the same edge, resulting in worst-case runtime of $O(m^2)$. For adding new paths (Section 2.3), we only perform linear-time breadth-first search to traverse new

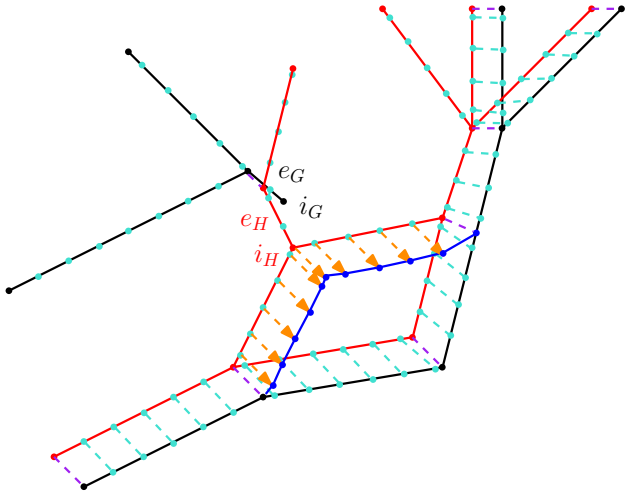

Figure 7: Two roadmaps G and H in black and red and their samples in cyan. Purple dashed lines indicate a matching between two intersections. i_H is a newly matched intersection however the BFS won't consider e_H for addition because it is partially matched. The orange vector $\vec{v}$ is used to find the new vertices' locations. The added parts are shown in blue.

paths, and then add the path via stitching. The stitching algorithm for adding a path takes constant time for the body of the for-loop (lines 5-24 in Algorithm 1), thus a total of linear time over all samples in H. Thus the total runtime is $O(m^2 + (m+n)\log n)$.

3 Experimental Evaluations

We provide qualitative and quantitative measure using visual examples and a precision and recall evaluation. To achieve our outputs we have used the default values in [2] for matching threshold $d_{\max} = 15$ meters, bearing threshold $\theta = 45$ degrees and sample placement interval $\Delta = 5$ meters. We set matched percentage threshold to $\rho = 0.1$ in our experiments, meaning that edges in the second graph with a matched percentage above 10 percent are disregarded, and only those with less than 10 percent matched samples are considered for addition. This threshold controls the aggressiveness of merging, and can be adjusted based on data quality and similarity in common areas to include more details.

3.1 Data and Code

In our experiments we used roadmaps from OpenStreetMap (OSM) for the *Berlin*, *Athens* and *Chicago* data sets that are available on mapconstruction.org. For *Berlin*, we have roadmaps from TeleAtlas (TA) from 2007 and OpenStreetMap (OSM) from 2013. Similarly, for *Athens*, we have large TA maps from 2007 and OSM maps from 2010. The *Chicago* OSM map is from 2012. The number of vertices and edges for these datasets and their respective total length are shown in Table 1. Unfortunately, the TA maps are not publicly available. However, we provide pictures of the input and output maps (Figure 13 and Figure 14) for transparency. Moreover, we have used the Graph Sampling Toolkit from [2] for evaluation and visualizations.

Map Stitcher was implemented in python and is available on our repository https://github.com/Erfanh1995/MapStitcher.

Datasets	Source	Vertices	Edges	Total Length (km)
Berlin	OSM	5894	6839	357.72
	TA	9848	11165	412.25
Athens	OSM	32212	39699	2000.47
	TA	66753	75695	2226.18
Chicago	OSM	8924	11338	596.50

Table 1: Attributes of the input datasets *Berlin*, *Athens* and *Chicago*.

3.2 Visual Demonstration

Before we attempt to measure the quantitative quality of our outputs, let us first showcase the visual quality of our maps using examples from several locations in our datasets. Here, we examine various corner cases that can be challenging. These include:

(1) Intermediate connections between lanes
(2) Roundabouts
(3) Missing or partially-present edges that would complete a loop
(4) Short and misplaced highway ramps
(5) Right-turn lanes near intersections

One such notable location is ramp entrances leading onto highways or expressways. Due to the presence of small edges with only one sample, our algorithm detects these edges for addition. While it's difficult to entirely avoid adding these edges, as they can occur anywhere in the datasets, even in the middle of long straight roads, our algorithm attempts to self-correct. After adding such an edge, it connects the endpoint back to the main road, forming a small loop and preventing short dead ends in these areas. Figure 8 presents two examples of this scenario, demonstrating our algorithm's ability to navigate complex situations.

Maintaining geometric properties: Map Stitcher preserves the original length, angles, and shapes of the maps, even when the roadmaps are slightly misaligned. Figure 9 showcases our method's capability to effectively handle complex shapes and geometries, maintaining their integrity and accuracy.

Preserving topological structure: Our approach effectively manages unclosed loops, avoiding unwanted artifacts around connections. However, due to strict constraints, it occasionally overlooks extending dead-end roads. Figure 10 illustrates an area with multiple loops requiring closure, and as seen, our algorithm handles them successfully, with only one exception.

3.3 Automated Precision and Recall Evaluation

Assessing the effectiveness of map conflation methods poses a significant challenge due to the absence of ground-truth references for comparison. This lack of information renders most evaluation metrics inadequate. In academic literature, researchers have relied on precision and recall to evaluate matchings and conflation, involving manual examination of matched and unmatched road segments to count "correct" and "incorrect" matches based on visual features or metadata. While this approach provides some insight into the quality of the matching, it does not assess the conflation. Moreover, it fails to account for topological and geometrical features of the output map or attributes that are not necessarily visually apparent, such as duplicated edges or disconnected road endpoints that happen to be in the same location. These attributes are crucial for a quality roadmap and require more comprehensive evaluation methods beyond manual assessment. In this section, we will discuss an alternative evaluation method that may be more suitable for assessing the effectiveness of map conflation techniques.

Precision and recall are also the two primary metrics used to evaluate the outcomes of graph sampling methods [2]. Important

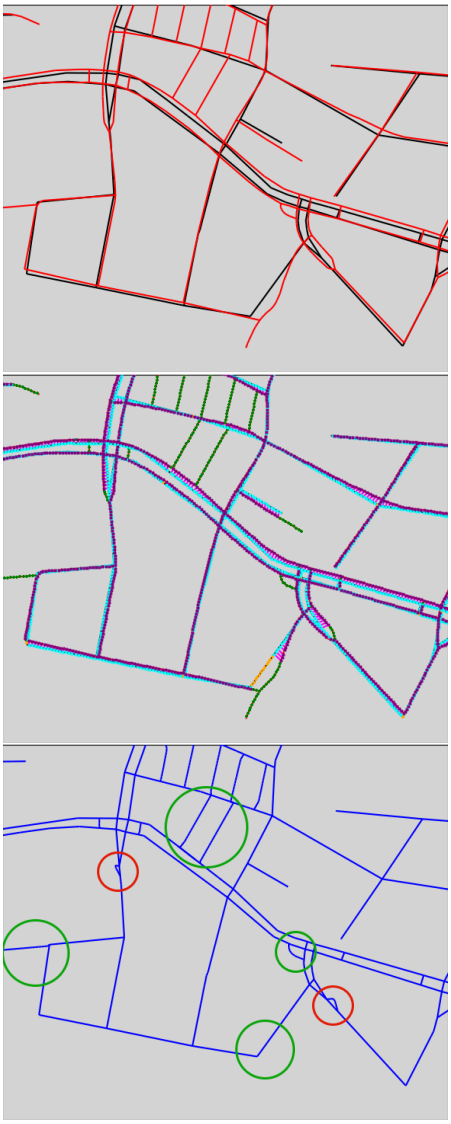

Figure 8: The input roadmaps OSM and TA are shown in black and red, respectively. Matched samples on G and H are in cyan and purple, unmatched samples are in orange and green. The blue roadmap is the output. Green circles highlight successes, red circles indicate shortcomings.

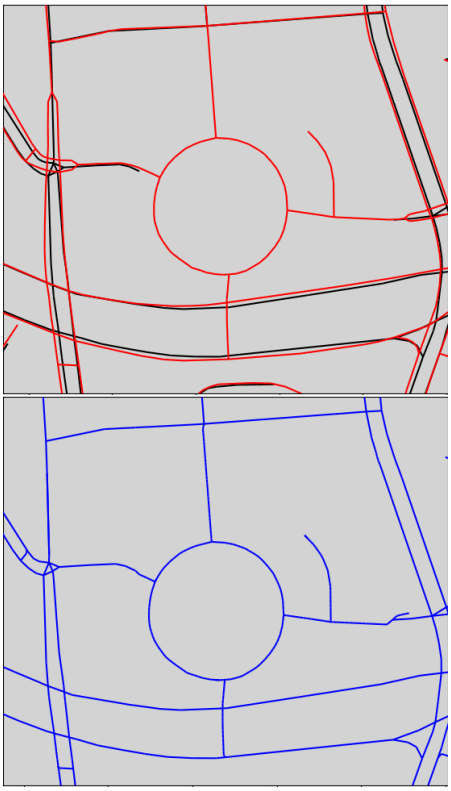

Figure 9: The input roadmaps OSM and TA are shown in black and red, respectively. The blue roadmap is the output.

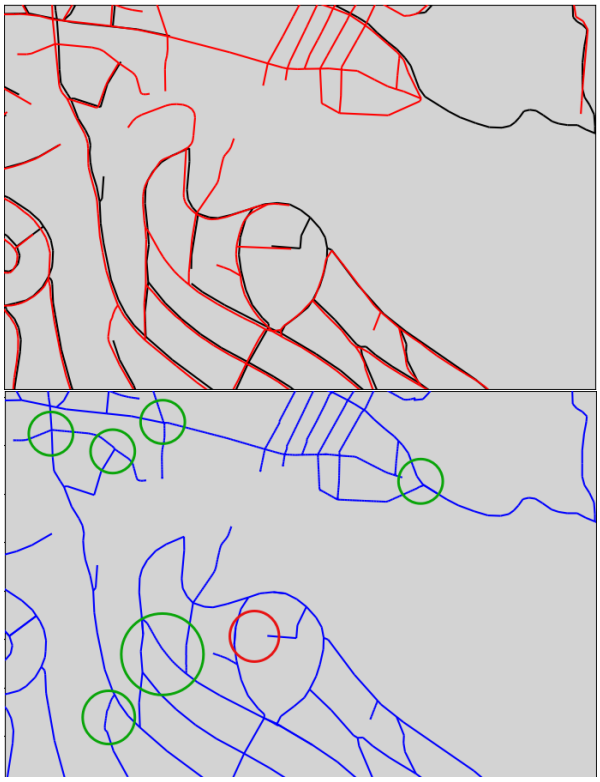

Figure 10: The input roadmaps OSM and TA are shown in black and red, respectively. The blue roadmap is the output. Green circles highlight successes, red circles indicate shortcomings.

to note that these precision and recall values differ from the ones we discussed above, and should not be confused with those. In this context, when comparing two graphs, G and H, precision is calculated as the ratio of matched samples to the total number of samples in H (the second roadmap), while recall is calculated as the ratio of matched samples to the total number of samples in G (the first roadmap). This enables more thorough assessments, as segments can be matched partially, and their geometric characteristics, including angle and length, are taken into account, influencing the evaluation outcome. Hence, we first show the results of the comparison between the input roadmaps in Table 4 and Table 2 using graph sampling. These results provide valuable insights into the similarities and differences between the two input maps and can be used for understanding and interpreting the evaluation of our output. An analysis of the precision and recall values reveals that, in the case of *Berlin*, the TA map exhibits a high recall value, indicating that it encompasses most of the roads present in the OSM maps. However, the precision value suggests that the OSM map does not similarly capture all the roads in the TA map. In contrast, the *Athens* maps display greater diversity and cover a significantly larger area, making direct comparisons more complex.

Knowing the information above, we applied Map Stitcher to the *Athens* and *Berlin* data sets, using OpenStreetMap (OSM) as the base map and supplementing it with TeleAtlas (TA) maps, which offer greater detail. In our initial evaluation approach, we employ graph sampling to assess the merged map by comparing it to each of the input roadmaps. Assuming G as one of the input roadmaps and H as the output we have computed the values on Table 5 and Table 3.

It's expected that our recall values would be high when compared to OSM maps, since we don't make substantial changes to the base map. However, the notable increase in recall values when compared to the secondary map indicates a significant enhancement in coverage. Our output achieved exceptional recall values on both datasets, successfully covering 97 percent of the secondary map, demonstrating a substantial improvement in map completeness.

When examining precision values in relation to the OSM map, the results offer limited insight. However, the notable improvements in precision when compared to the supplementary map indicate a significant enhancement in the consistency and accuracy of the added segments, demonstrating a higher degree of precision in our output.

3.4 Constructing A Dataset With Ground Truth

In this section, we aim to "artificially" construct a dataset by splitting a roadmap G into two sub-roadmaps G_1, G_2 sets such that (a) their union covers G, so, $G_1 \cup G_2 = G$, and (b) their intersection is not empty, so, $G_1 \cap G_2 \neq \emptyset$.

Berlin	Samples	Matched	Precision	Recall	F
OSM	71,515	67,936	0.82	0.95	0.88
TA	82,423				

Table 2: Graph sampling evaluation for Berlin OSM vs. TA, using $d_{\max} = 15m$.

Berlin	Samples	Matched	Precision	Recall	F
OSM	71,515	71,002	0.82	0.99	0.90
Merged	86,397	80,076	0.93	0.97	0.95
TA	82,423				

Table 3: Graph sampling evaluation for Berlin OSM vs. the Merged output and TA vs. the Merged output, using $d_{\max} = 15m$.

Athens	Samples	Matched	Precision	Recall	F
OSM	399,898	349,725	0.79	0.88	0.83
TA	445,092				

Table 4: Graph sampling evaluation for Athens OSM vs. TA, using $d_{\max} = 15m$.

Athens	Samples	Matched	Precision	Recall	F
OSM	399,898	396,383	0.81	0.99	0.89
Merged	491,720	420,380	0.85	0.94	0.90
TA	445,092				

Table 5: Graph sampling evaluation for Athens OSM vs. the Merged output and TA vs. the Merged output, using $d_{\max} = 15m$.

The two roadmaps G_1, G_2 are then given to *Map Stitcher* as inputs, and the output is compared with the initial roadmap G (ground truth). While this approach is not flawless, as input maps often differ in their overlapping regions as well, it can still demonstrate the consistency of our method. For this purpose we use the well-known *Chicago* roadmap [5]. We split the roadmap randomly into two roadmaps via an automated program [2] (see Section A for details) while making sure the two have overlapping areas. Figure 11 illustrates the outcome of this process, displaying two graphs generated through this method, with the final image depicting the merged result. We utilize Graph Sampling to evaluate the merged result by comparing it to the ground truth, as shown in Figure 12. This comparison yields an evaluation summary in Table 6, revealing a striking similarity of approximately 97 percent.

3.5 Runtime

Table 7 shows the runtimes of *Map Stitcher* given *Athens* and *Berlin* datasets on a 2017 3.1 GHz Dual-Core Intel Core i5 Macbook Pro with 16 GB 2133 MHz LPDDR3 RAM. Note that the time required for matching increases when G and H have more extensive overlapping areas, and conversely, the time needed for merging grows when the disparity between G and H is more significant.

[2]This program and the constructed dataset are also available on our repository.

Chicago	Samples	Matched	Precision	Recall	F
Ground-truth	119,170	116,245	0.967	0.975	0.971
Merged	120,230				

Table 6: Graph sampling evaluation for Chicago ground-truth vs. Merged, using $d_{\max} = 15m$.

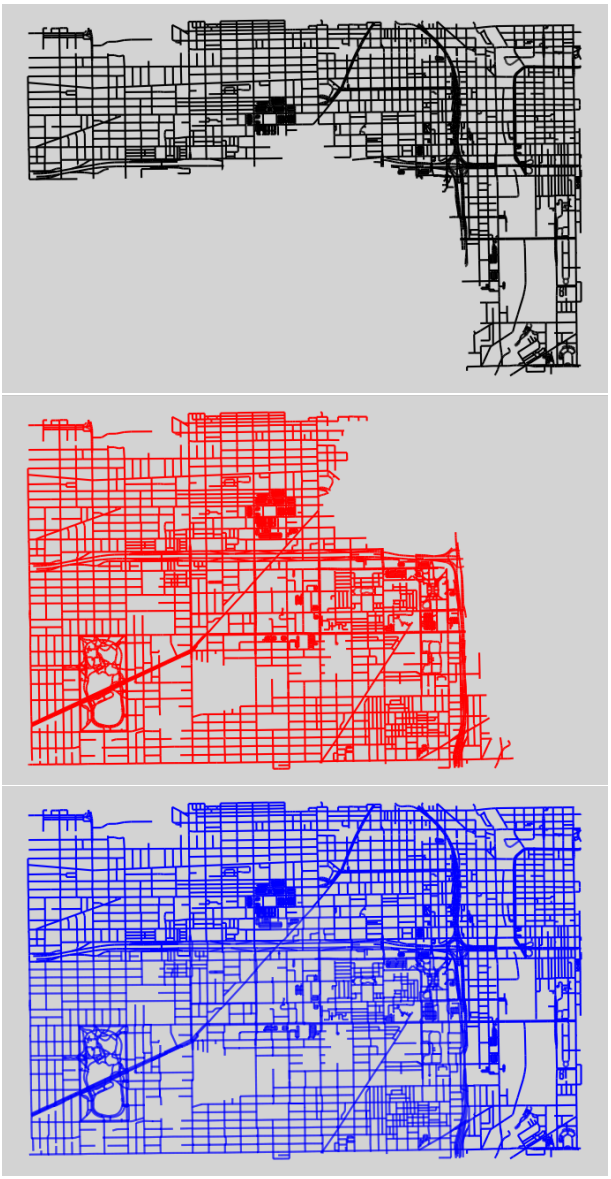

Figure 11: The two input roadmaps are shown in black and red while the merged roadmap (output) is blue.

Datasets	Matching (s)	Merging (s)
Berlin	24	228
Athens	386	9125
Chicago	18	1653

Table 7: Runtime in seconds for *Berlin, Athens* and *Chicago*

4 Conclusion and Discussion

We presented Map Stitcher, an automated tool for merging two roadmaps. Map Stitcher ensures the preservation of topological features, such as intersections and their vertex degrees, by deliberately identifying and addressing missing intersections in the second map. This cautious approach also effectively handles small edges near intersections, roundabouts, and misplaced highway ramps (As previously shown in Figure 8). Furthermore, our approach maintains geometric features, including lengths, angles, and relative feature locations, by leveraging graph sampling's inherent preservation of approximate length and angle (via Δ and θ). The Stitch function Algorithm 1 utilizes this property to reconstruct the original geometry. Additionally, the location of new features is seamlessly integrated into the base map by translating their position using the vector of the closest matched pair in the current path, ensuring a consistent relative location with existing features.

Map Stitcher achieves "Differential Conflation" by seamlessly integrating new features from a secondary map without altering the existing base map features. However, handling areas that have undergone significant changes, such as repurposing or reconstruction, poses a significant challenge, requiring additional investigation. More future work could work out a deeper understanding of the existence of artifacts in practice and further optimize the handling of them. In addition, further work on developing more automatic methods for evaluating merged maps would be helpful for the community. If available, one approach could use GPS trajectory data and compare how well those trajectories map-match to the individual maps and to the merged maps.

Acknowledgments

E.H.S. and C.W. were both partially supported by National Science Foundation grant CCF 1637576.

References

[1] Gorisha Agarwal, Laks V.S. Lakshmanan, Xiaoming Gao, Kevin Ventullo, Saurav Mohapatra, and Saikat Basu. 2021. MAYUR: Map conflAtion using earlY prUning and Rank join. In *Proc. 29th ACM SIGSPATIAL GIS* (Beijing, China). 550–553. https://doi.org/10.1145/3474717.3484258

[2] Jordi Aguilar, Kevin Buchin, Maike Buchin, Erfan Hosseini Sereshgi, Rodrigo I. Silveira, and Carola Wenk. 2024. Graph Sampling for Map Comparison. *ACM Trans. Spatial Algorithms Syst.* 10, 3 (2024), 1–24. https://doi.org/10.1145/3662733

[3] M. Ahmed, S. Karagiorgou, D. Pfoser, and C. Wenk. 2015. *Map Construction Algorithms*. Springer. https://doi.org/10.1007/978-3-319-25166-0

[4] Favyen Bastani and Sam Madden. 2021. Beyond Road Extraction: A Dataset for Map Update using Aerial Images. arXiv:2110.04690 [cs.CV]

[5] James Biagioni and Jakob Eriksson. 2012. Inferring Road Maps from Global Positioning System Traces. *Transportation Research Record: Journal of the Transportation Research Board* 2291, 1 (2012), 61–71.

[6] J. Biagioni and J. Eriksson. 2012. Inferring Road Maps from Global Positioning System Traces: Survey and Comparative Evaluation. *Transportation Research Record: Journal of the Transportation Research Board* 2291 (2012), 61–71.

[7] J. Biagioni and J. Eriksson. 2012. Map inference in the face of noise and disparity. In *Proc. 20th ACM SIGSPATIAL GIS* (Redondo Beach, California). 79–88.

[8] M. Buchin, E. Chambers, P. Fang, B.T. Fasy, E. Gasparovic, E. Munch, and C. Wenk. 2023. Distances Between Immersed Graphs: Metric Properties. *La Matematica* 2 (2023), 197–222. https://doi.org/10.1007/s44007-022-00037-8

[9] Ting Lei and Zhen Lei. 2019. Optimal spatial data matching for conflation: A network flow-based approach. *Transactions in GIS* 23, 5 (2019), 1152–1176. https://doi.org/10.1111/tgis.12561

[10] Ting L. Lei. 2021. Large scale geospatial data conflation: A feature matching framework based on optimization and divide-and-conquer. *Computers, Environment and Urban Systems* 87 (2021), 101618. https://doi.org/10.1016/j.compenvurbsys.2021.101618

[11] Zhen Lei and Ting L. Lei. 2024. Towards Topological Geospatial Conflation: An Optimized Node-Arc Conflation Model for Road Networks. *ISPRS International Journal of Geo-Information* 13, 1 (2024). https://doi.org/10.3390/ijgi13010015

[12] Alan Saalfeld. 1988. Conflation Automated map compilation. *International journal of geographical information systems* 2, 3 (1988), 217–228. https://doi.org/10.1080/02693798808927897

[13] Zhangqing Shan, Hao Wu, Weiwei Sun, and Baihua Zheng. 2015. COBWEB: a robust map update system using GPS trajectories. In *Proc. ACM International Joint Conference on Pervasive and Ubiquitous Computing* (Osaka, Japan). 927–937. https://doi.org/10.1145/2750858.2804286

[14] Rade Stanojevic, Sofiane Abbar, Saravanan Thirumuruganathan, Gianmarco De Francisci Morales, Sanjay Chawla, Fethi Filali, and Ahid Aleimat. 2018. Road Network Fusion for Incremental Map Updates. In *Progress in Location Based Services 2018*, Peter Kiefer, Haosheng Huang, Nico Van de Weghe, and Martin Raubal (Eds.). Springer, 91–109.

[15] Tao Wang, Jiali Mao, and Cheqing Jin. 2017. HyMU: A Hybrid Map Updating Framework. In *Database Systems for Advanced Applications*, Selçuk Candan, Lei Chen, Torben Bach Pedersen, Lijun Chang, and Wen Hua (Eds.). Springer, 19–33.

[16] Yin Wang, Xuemei Liu, Hong Wei, George Forman, Chao Chen, and Yanmin Zhu. 2013. CrowdAtlas: self-updating maps for cloud and personal use. In *Proc. 11th ACM Annual International Conference on Mobile Systems, Applications, and Services* (Taipei, Taiwan). 27–40. https://doi.org/10.1145/2462456.2464441

[17] Xu Zhang and Mei Chen. 2023. Methodology for Conflating Large-Scale Roadway Networks. *Transportation Research Record* 2677, 3 (2023), 189–202. https://doi.org/10.1177/03611981221115085 arXiv:https://doi.org/10.1177/03611981221115085

A Details About the Program Used In Section 3.4

The construction of the *Chicago* ground truth dataset involves the following process: Initially, two random edges from graph G are selected and assigned to graphs G_1 and G_2, respectively. To ensure connectivity in G_1 and G_2, adjacent edges are stored in separate lists (c_1 and c_2) for each graph. The algorithm then enters a loop, repeatedly selecting random edges from c_1 and c_2 until all edges from G have been processed. In each iteration, if an edge is found in both c_1 and c_2, it is added to both G_1 and G_2; otherwise, each graph receives the edge selected from its respective candidate list and c_1 and c_2 are updated.

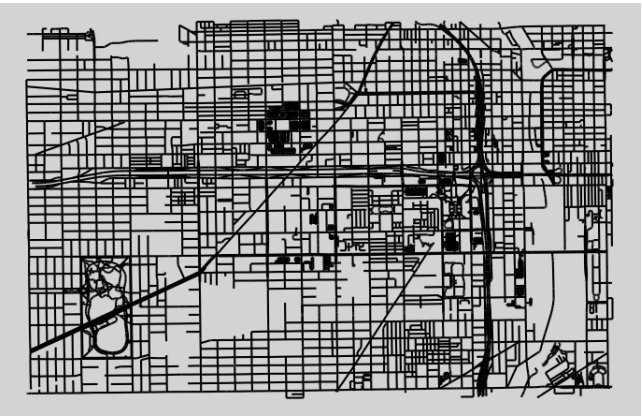

Figure 12: Chicago OSM ground truth (black)

Figure 14: Berlin OSM (black), TA (red), and Merged (blue).

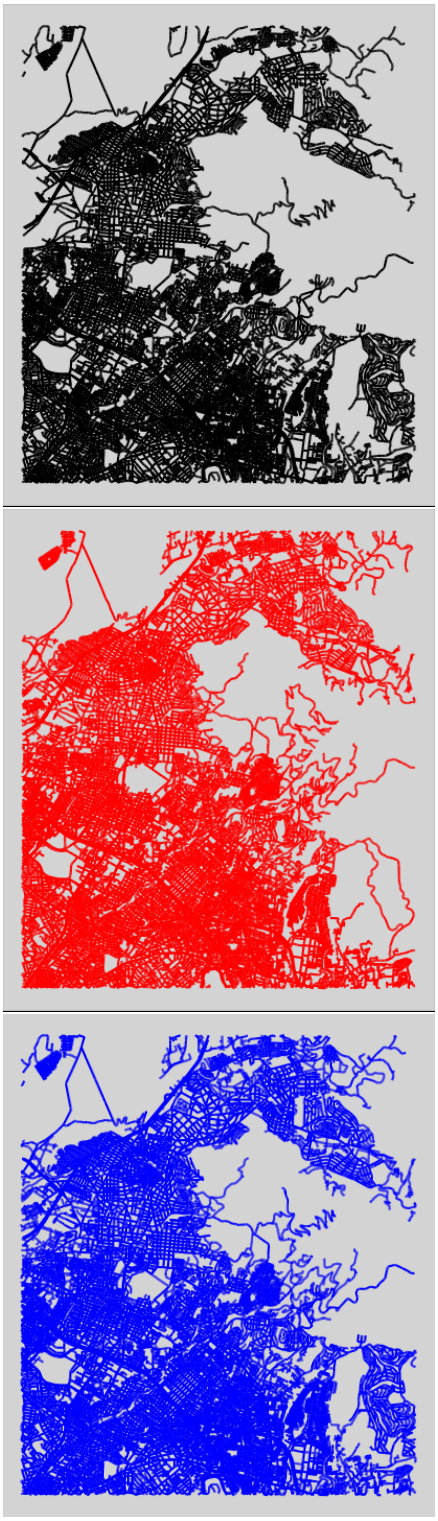

Figure 13: Athens OSM (black), TA (red), and Merged (blue).

Received 14 September 2024