MAYUR: Map conflAtion using earlY prUning and Rank join

Gorisha Agarwal
agarwal.gorisha@gmail.com
University of British Columbia

Laks V.S. Lakshmanan
laks@cs.ubc.ca
University of British Columbia

Xiaoming Gao
gaoxm@fb.com
Facebook, Inc.

Kevin Ventullo
kevin.ventullo@gmail.com

Saurav Mohapatra
mohaps@gmail.com

Saikat Basu
saikatbasu@fb.com

ABSTRACT

OpenStreetMap (OSM) is a collaborative good quality crowd-sourced geospatial database (GDB). The quality of OSM is generally very good, it lacks good coverage in many parts of the world. A natural approach for extending its coverage is to conflate missing spatial features from other GDBs into OSM, but this is laborious and time-consuming. We propose a system MAYUR solving road network conflation between two vector GDBs, representing the GDBs as a graph of road intersections (vertices) and road segments (edges). MAYUR is based on a novel map matching framework that adapts the classic *Rank Join* in databases, where each edge of the reference GDB is modeled as a relation. Our algorithm finds the best matching between a reference and target GDB, respecting the connectivity of the road network. While classic Rank Join in databases gets quickly inefficient on instances with more than 10 relations, MAYUR's enhanced Rank Join incorporates three optimizations that boost the algorithm's efficiency, making it scale to our problem setting featuring hundreds to thousands of relations. Our manual evaluation of MAYUR conflation results on sidewalks in OSM and Boston Open Data shows an impressive 98.65% precision and 99.55% recall.

CCS CONCEPTS

• **Theory of computation → Design and analysis of algorithms**.

KEYWORDS

conflation, rank join, map matching, map merging, rubbersheeting

ACM Reference Format:
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 *SIGSPATIAL '21: International Conference on Advances in Geographic Information Systems, November 02–05, 2021, Beijing, China.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Map conflation [7] is a process of merging two overlapping GDBs, combining accurate information from either database while minimizing the error. It primarily consists of two steps: (i) *Map matching*, which aligns all pairs of features (f, f') between two databases D and D' such that $f \in D$ and $f' \in D'$ represent the same physical entity on Earth; and (ii) *Map merging*, which merges the unmatched features with the matched features to obtain a conflated (i.e., extended or augmented) database. One major challenge in map conflation comes from the positional inaccuracy of the spatial features in GDBs. Secondly, for linear objects like roads, the matching can be $m:n$, i.e., m lines from one database may correspond to n lines from the other. Thirdly, a spatial feature in one database may not completely match a feature in the other. It is non-trivial to distinguish the case where a feature is partially present in a database from the case where it is completely absent, making this distinction a challenge. All these issues combine to make map matching significantly challenging. Another important concern in map matching, and consequently map conflation, is the need for scalability of the approach to large GDBs.

In this paper, we describe our principled automated map conflation system called Map conflAtion using earlY prUning and Rank join (MAYUR[1]) for conflating linear road-like spatial features that cover roads, sidewalks, cycleways, etc., hereafter referred to as just *roads*, between a pair of GDBs. Our map matching approach is based on an adaptation of Rank join in relational databases (§ 4). Classic Rank Join fails to find the matching on databases with even a few hundred features in a reasonable time. MAYUR leverages three optimizations to make rank join scalable for large GDBs while ensuring correctness (§ 4). We conduct extensive experiments on two open-source GDBs – OpenStreetMap (OSM) and Boston Open Data (BOD) road lines in Boston city. Our manual evaluation shows that MAYUR achieves high precision and recall for map matching, and that a huge majority of the conflated features are perturbed only within a small distance (§ 5.1-5.3). Our ablation studies show that the best performance is achieved by using all the three optimizations (§ 5.4). § 2 discusses related work, while § 3 defines the notions we use and the problem statement. Finally, in § 6, we present our conclusions. For additional experiments and details, the interested reader is referred to [1].

2 RELATED WORK

Previous approaches for automated map matching can be broadly classified into two categories: (i) greedy heuristics [6, 8, 12], which quickly find a sub-optimal solution but with no guarantee of correctness, and (ii) optimization algorithms [3, 4, 10], which find the optimal solution using an objective function and are relatively more time consuming. These approaches suffer from one or more of the following limitations: (i) do not easily generalize to any GDBs, (ii)

[1]Pronounced Mayoor.

SIGSPATIAL '21, November 02–05, 2021, Beijing, China

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

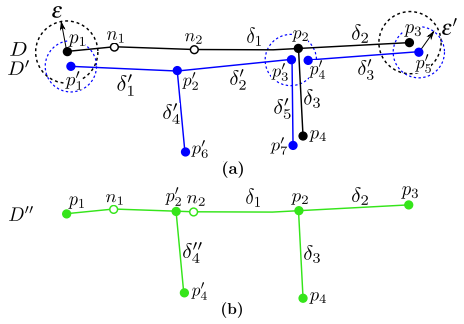

Figure 1: (a) An example of reference and target database D, D' with circular errors ϵ, ϵ'. (b) The expected conflation D'' of the GDBs in (a).

fail to scale to large GDBs, (iii) do not address general $m:n$ matching, or (iv) fail to account for spatial attributes like angle, or ignore spatial connectivity. The limitations prevent them from being applicable for scalable and high quality map conflation. Our proposed system MAYUR finds the best map matching solution, and does not suffer from any of the limitations mentioned above.

3 BACKGROUND & PROBLEM STATEMENT

We term dead-ends of roads as well as road intersections as *terminal points*. All other points are termed *intermediate points*. E.g., in Fig. 1(a), in the GDB D, nodes p_i are terminal while n_i are intermediate points. A *road segment* is an ordered sequence of zero or more intermediate points between a pair of terminal points, such that consecutive points in the sequence are connected by a geometric straight-line. *Segments form the fundamental unit of matching between GDBs.* A *polyline* is a simple path between any pair of terminal nodes. We represent the road network in a GDB D as a graph, $\mathcal{G}(D)$, where the terminal points constitute vertices, and the connections between them via segments are modeled as edges. The uncertainty in the positional accuracy of the spatial features in a GDB is termed as its *circular error*. We study the problem of conflation of road networks between two vector GDBs. Suppose that $G \triangleq \mathcal{G}(D) = (V, E)$ and $G \triangleq \mathcal{G}(D') = (V', E')$ are the graph representations of given D, D' respectively. Let $\mathcal{P}(D)$ be the set of all simple paths in D (equiv. in the graph G). In matching features between D and D', we use the *Hausdorff distance* between the features, $d_H(.)$, as a criterion. A segment in D may be matched with a polyline in D'.

Definition 3.1 (Map matching). Given a reference database $\mathcal{G}(D) = (V, E)$ and a target database $\mathcal{G}(D') = (V', E')$ with circular error ϵ and ϵ' respectively, and a distance bound Δ_p, a matching of D to D' is a function $\mu : E \longrightarrow \mathcal{P}(D') \cup \{\phi_e \mid e \in E\}$ such that $\forall e \in E$: either $(\nexists \ell' \in \mathcal{P}(D') : d_H(e, \ell') \leq \Delta_p \ \wedge \ \mu(e) = \phi_e)$ or $d_H(e, \mu(e)) \leq \Delta_p$.

In the above definition, $\Delta_p := \epsilon + \epsilon'$ is an upper bound on the maximum Hausdorff distance allowed between a pair of matched objects e and $\mu(e)$. An object $O \in D$ can have more than one candidate match $O' \in D'$, and each candidate match has a *score* that decides the ordering between the candidates. Given a reference and a target GDB D and D' respectively, a set of scored candidate matches for each segment in D, and a set of all possible matchings $\mathcal{M}$ between D and D', a partial ordering $\preceq$ on $\mathcal{M}$ is defined as follows. We associate a monotone aggregate scoring function $S(.)$ with matchings. Intuitively, $S(\mu)$, the score associated with matching μ, captures the extent of geometric proximity between D and its image $\mu(D)$ in D'. For matchings $\mu_i, \mu_j \in \mathcal{M}$, we define $\mu_i \preceq \mu_j$ iff $S(\mu_i) \geq S(\mu_j)$. Next, using the best matching, we employ rubbersheeting to merge all the unmatched segments in the target GDB with the reference GDB to obtain the conflated GDB, by perturbing them only within a certain bound. The problem we study is as follows.

Problem 1 (Map Conflation). *Given D and D', find the best (i.e., top-1) matching μ of D to D' and obtain the corresponding merging of D and D'.*

4 MAYUR

Given a reference GDB D and a target GDB D', we find the set of candidate matches $R(\delta)$ for every segment $\delta \in D$. The candidate matches in $R(\delta)$ are polylines in D' with their endpoints within the circular error of the endpoints of δ. We discuss the limitations of the classic rank join algorithm, followed by our proposed optimizations to make it efficient and scalable for map matching.

4.1 Transformation to rank join

There has been extensive research on efficiently finding the top-k results of the join of relational tables containing scored objects, without exhaustively computing the join [2, 5, 9, 11]. We refer the reader to [9] for a description of the generalized Pull Bound Rank Join (PBRJ) template, on which our approach is based.

Each segment in D may have 0 or more scored candidate matches in D'. By regarding $R(\delta), \delta \in D$, as relations, we compute the rank join of the relations. The top join result intuitively corresponds to the best matching of the database D with D'. The join conditions are based on equality of the endpoints of the segments. One complication is that a pair of segments connected in D may have their counterparts in D' disconnected owing to the positional inaccuracy in data collection. For example, in Fig. 1, we have the following matches: $\delta_1 \mapsto \ell'_1 = \langle \delta'_1 \cdot \delta'_2 \rangle$; $\delta_2 \mapsto \delta'_3$; $\delta_3 \mapsto \delta'_5$. Segments δ_1 and δ_2 are connected at p_2 but their matches aren't. To account for this, we permit a *relaxed join*: two tuples (matches) join provided their terminal nodes are within the circular error ϵ' of D'. A second complication is that D' may not have a match for a segment $\delta \in D$, making $R(\delta) = \emptyset$, which renders the result of rank join empty by definition. We get around this by introducing a dummy segment in every relation $R(\delta)$. Dummy segments have score 0 and join with any polyline of D', by definition. This ensures that the rank join will always find a non-empty result. Finally, rank join algorithms assume a monotone aggregate scoring function. In our context, we use the weighted average of the segment scores as the overall score. Thus, we compute the best matching of D into D' as the top-1 result of the rank join of the relations $R(\delta)$, for segments $\delta \in D$.

4.2 Limitations, Challenges, and Optimizations

(1) The PBRJ algorithm has been shown to be *instance optimal*, which intuitively means its performance is no worse than n times that of the optimal algorithm for a given instance. This result is significant and useful when the number of relations being joined is small: rank join is typically considered for < 10 relations [2, 5]. *In the map matching setting, where there are hundreds to thousands*

of segments, with each having a small number of matches (e.g., < 5), instance optimality is not useful as a performance guarantee. (2) The PBRJ algorithm relies on a "corner bound" (CB): it assumes that the next tuple joins with the best scored tuples from all other relations. *As a result, the global upper bound does not decrease fast enough to enable computation of rank join of hundreds of relations in reasonable time.* (3) As a consequence of the bounding above, PBRJ pulls the next tuple from the relation with the *least* decrease in the global upper bound. These tuples may not contribute to the top-k results, *leading to a prohibitive amount of time to compute joins between a large number of relations, generating a large number of intermediate results.* We next describe briefly our optimizations to the bounding scheme and pulling strategy.

4.2.1 Look-ahead Corner Bound. We tighten the global upper bound by doing a *look-ahead* of one tuple and using the score of the next "unseen" tuple in every relation. We refer to this as *Look-ahead Corner Bound* (LCB for short). Intuitively, this helps in accelerating the rate at which the global upper bound decreases. In our experiments, we found that this results in a computational saving of 20% over PBRJ using CB (§ 5.4). In place of the corner adaptive bound (CAB) pulling strategy, we propose *Look-ahead corner-Adaptive-Bound* (LAB) strategy that probes a relation with the highest LCB bound.

4.2.2 Groups of relations. In classic Rank Join, the upper bound of every relation is calculated assuming the next tuple in the relation joins with the top tuple from *every* other relation. When computed over a large number of relations, this greatly limits how quickly the upper bound can decrease. To mitigate this, we partition the set of segments in D into disjoint groups such that the segments in each group are connected in D. Instead of directly joining the tuples in relations, we join the tuples obtained from groups via *intra-group join*, where each group contains a small number of relations (~ 5).

4.2.3 Dynamic Re-Grouping. To minimize unnecessary probes, we exploit the fact that if the size of an intermediate result is zero, then some group in the join tree has failed to join with the group chosen in a certain iteration. We then find the group in the subtree that fails to join with the probed join tuple and re-group the relations into one group and obtain the join tuple using intra-group join. More details of our optimized algorithm, RJ++, can be found in [1].

5 EXPERIMENTAL EVALUATION

5.1 Datasets

We evaluate MAYUR on the sidewalks conflation of two real-world GDBs – OpenStreetMap[2] (OSM) and Boston Open Data[3] (BOD). The statistics of sidewalks in the two datasets are in Table 1. BOD

Matching Areas	OSM	BOD
# sidewalks (ways)	28,321	110,031
Total length of sidewalks (km)	1,845.4	3,400.6
# terminal nodes	32,569	96,495
# segments	38,162	109,497

Table 1: Datasets.

has a much larger coverage of sidewalks than OSM for the same area: we choose BOD (resp. OSM) as the target (resp. reference) GDB. We divide the entire area into 20 similar test areas to facilitate in-memory processing. They differ in the number of segments per unit area. We empirically choose the circular error to be 5 meters for both GDBs. We run our experiments on a standard Intel(R) Xeon(R) CPU X5570 machine with 94G RAM.

[2]https://www.openstreetmap.org/

[3]https://bostonopendata-boston.opendata.arcgis.com/datasets/sidewalk-centerline/

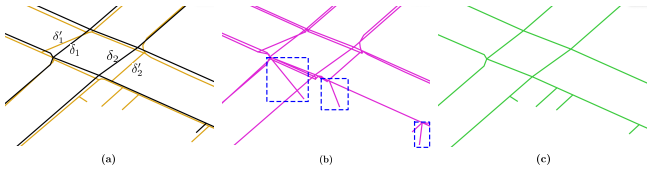

Figure 2: An example of (b) Hootenanny and (c) MAYUR conflation results for GDBs (a), at (lon, lat) = (-71.0346, 42.3701).

5.2 Evaluation results

We group the 20 test areas of Boston into 4 zones to facilitate scalability tests. Zones 1-4 have increasing number of OSM segments for which they have matches (see Table 2).

Zones	OSM #segments		BOD
	total	with matches	#segments
1	3,169	**192**	13,414
2	11,709	**1,724**	27,334
3	5,417	**3,816**	22,324
4	17,867	**10,833**	46,425

Table 2: Characteristics of each group of test areas.

5.2.1 Map Matching. We manually evaluate the results obtained by MAYUR and report the precision and recall of its map matching performance in Table 3 (see only the MAYUR columns for now). The overall precision (resp. recall), computed using the sum of each of correct, incorrect and missing matches [1], across all zones is 98.65% (resp. 99.55%). The running time of MAYUR for matching is reported in Table 4. It includes the time taken to 1) find the candidate matches of OSM segments, 2) RJ++ to find the top-1 matching in BOD. Notice that *the running time of RJ++ is only a tiny fraction of the time taken to find the candidate matches.*

5.2.2 Map Merging. Table 5 reports the performance of the map merging module of MAYUR, using the measures: *Segments (not) within λ meters*: Percentage of segments $\delta' \in D'$ for which the Hausdorff distance to the corresponding merged δ'' in the conflated database is $\leq \lambda$ ($> \lambda$) meters. Since $\epsilon = \epsilon' = 5$ meters, we report the accuracy for $\lambda = 5m$ and $\lambda = \epsilon + \epsilon' = 10m$. For the complete dataset, 97.63% (resp. 99.22%) of the segments in BOD are within a Hausdorff distance of 5 (resp. 10) meters w.r.t. their merged counterparts in the conflated GDB. On the other hand, $< 0.01\%$ (resp. 0.004%) segments are displaced more than 5 (resp. 10) meters after conflation.

5.3 Comparison with State of the art

We compare MAYUR with Hootenanny, an open-source[4] map conflation framework. The overall precision and recall of Hootenanny is 93.21% and 91.78% respectively, which is approximately 5% and 8% lower than MAYUR. In Fig. 2 and Fig. 3, part (a) shows OSM, BOD in black, orange lines, while parts (b) and (c) show the corresponding conflation results for Hootenanny and MAYUR respectively. Unlike MAYUR, Hootenanny's conflation contains two major errors compared to MAYUR: 1) the presence of duplicate sidewalks from OSM and BOD due to missing matches, 2) criss-cross of sidewalks with distorted geometries.

[4]https://github.com/ngageoint/hootenanny

Zones	# Matches						Precision		Recall	
	correct		incorrect		missing					
	MAYUR	Hotenanny	MAYUR	Hotenanny	MAYUR	Hotenanny	MAYUR	Hotenanny	MAYUR	Hotenanny
1	189	137	2	32	1	23	98.95%	81.07%	99.47%	85.63%
2	1, 688	1, 445	30	116	6	154	98.25%	92.57%	99.65%	90.37%
3	3, 747	3, 396	52	169	17	240	98.63%	95.26%	99.55%	93.40%
4	10, 644	9, 161	139	713	50	821	98.71%	92.78%	99.53%	91.78%

Table 3: Comparative Performance of MAYUR and Hootenanny: map matching.

Zones	Avg # candidate matches per segment	Total time for map matching (s)		Map matching speed
		Finding segment matches (s)	Time for RJ++ (s)	
1	2.71	18.54	0.09	170.11
2	3.52	215.63	4.49	53.19
3	3.76	1,152.44	17.81	4.63
4	5.94	14,065.93	540.38	1.22

Table 4: Running time of MAYUR: for map matching.

Zones	Road segments within 5 m	Road segments not within 5 m	Road segments within 10 m	Road segments not within 10 m
1	99.27%	0.006%	99.59%	0.004%
2	96.32%	0.015%	98.50%	0.007%
3	98.10%	0.005%	99.57%	0.002%
4	97.71%	0.009%	99.36%	0.003%

Table 5: Performance of MAYUR: map merging.

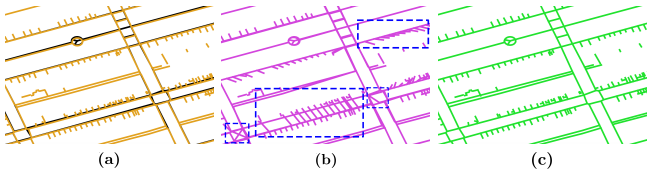

Figure 3: An example of (b) Hootenanny and (c) MAYUR conflation results for GDBs (a), at (lon, lat) = (-71.0824, 42.3505).

5.4 Ablation Study

We present an ablation study on RJ++ by measuring its running time under each of the eight combinations of the three optimizations proposed in § 4.2. We denote each variant by a number from 0 to 7 (000 to 111 in binary); each bit represents an optimization being turned on (1) or off (0). The most, second most, and least significant bit corresponds to dynamic re-grouping, LCB scheme, and grouping of relations respectively. The average size of the groups, when grouping is turned on, is 5. Since some variants do not terminate in a reasonable amount of time, we allow a maximum time of 100× the time taken by the most optimized variant 7 (111), before terminating a variant. Variant 0 (classic rank join) does not get the top-1 matching in a reasonable time for even zone 1, which has the fewest segments! This demonstrates that the classic rank join does not scale to map matching instances containing hundreds of relations (segments). We analyze the performance of top-3 variants 4, 6 and 7 in Fig. 4. The highly optimized variant 7 takes the least time. Variants 4 and 6, that differ in the LCB scheme, show that the LCB scheme provides a performance improvement of more than 20% on the problem instances tested. On the other hand, variants 6 and 7 differ in the grouping of relations. Grouping relations takes less time than joining individual relations. RJ++ (variant 7 (111)) finds the best matching in less than a second for as many as 3000

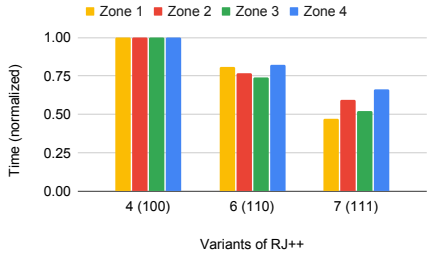

Figure 4: Time taken by the top three variants of RJ++.

relations in one of the areas of zone 4! Thus, RJ++ is practically the most efficient version based on the rank join algorithm.

6 CONCLUSIONS AND FUTURE WORK

We propose an efficient framework MAYUR for map conflation. Our matching module uses a novel adaptation of classic rank join studied in databases to find the best matching. While classic rank join does not scale to the task in hand, our enhanced algorithm RJ++ uses key optimizations to efficiently find the top-1best matching result. W.r.t. precision and recall achieved, RJ++ outperforms Hootenanny, a state-of-the-art approach. Our experiments also show that in the conflated database obtained by MAYUR, a huge majority of the features are spatially shifted by a very small amount.

REFERENCES

[1] Gorisha Agarwal. 2021. *A Principled Approach to Automated Road Network Conflation.* Master's thesis. University of British Columbia. https://bit.ly/3yW2MMo
[2] Ihab F Ilyas, Rahul Shah, Walid G Aref, Jeffrey Scott Vitter, and Ahmed K Elmagarmid. 2004. Rank-aware query optimization. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data.* 203–214.
[3] 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.
[4] Linna Li and Michael F Goodchild. 2011. An optimisation model for linear feature matching in geographical data conflation. *International Journal of Image and Data Fusion* 2, 4 (2011), 309–328.
[5] Davide Martinenghi and Marco Tagliasacchi. 2011. Cost-aware rank join with random and sorted access. *IEEE Transactions on Knowledge and Data Engineering* 24, 12 (2011), 2143–2155.
[6] Sébastien Mustière and Thomas Devogele. 2008. Matching networks with different levels of detail. *GeoInformatica* 12, 4 (2008), 435–453.
[7] Alan Saalfeld. 1988. Conflation automated map compilation. *International Journal of Geographical Information System* 2, 3 (1988), 217–228.
[8] Michael Schäfers and Udo W Lipeck. 2014. SimMatching: adaptable road network matching for efficient and scalable spatial data integration. In *Proceedings of the 1st ACM SIGSPATIAL PhD Workshop.* 1–5.
[9] Karl Schnaitter and Neoklis Polyzotis. 2008. Evaluating rank joins with optimal cost. In *Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems.* 43–52.
[10] Volker Walter and Dieter Fritsch. 1999. Matching spatial data sets: a statistical approach. *International Journal of geographical information science* 13, 5 (1999), 445–473.
[11] Shengqi Yang, Fangqiu Han, Yinghui Wu, and Xifeng Yan. 2016. Fast top-k search in knowledge graphs. In *2016 IEEE 32nd international conference on data engineering (ICDE).* IEEE, 990–1001.
[12] Meng Zhang and Liqiu Meng. 2008. Delimited stroke oriented algorithm-working principle and implementation for the matching of road networks. *Geographic Information Sciences* 14, 1 (2008), 44–53.