Meeting the Challenges of Integrating Large and Diverse Geographic Databases

Michael Schäfers, Udo W. Lipeck

Institut für Praktische Informatik, Fachgebiet Datenbanken und Informationssysteme
Gottfried Wilhelm Leibniz Universität Hannover
Welfengarten 1, 30167 Hannover
{schaefers,lipeck}@dbs.uni-hannover.de

Abstract: Using data matching techniques to identify multiple representations of the same real-world entity is an essential step for all data integration tasks. While matching standard data types like strings or numbers with generic methods is well-studied, approaches for non-standard data have to deal with domain-specific challenges. For geographic databases containing spatial features we face a high degree of diversity in terms of geometric and semantic modeling between data sources. Likewise, complex geometric data types and topological relations require efficient processing. Finally, geodatabases can grow very large if they cover extensive regions or whole countries.

In this paper, we present our SimMatching approach for integrating relational geodatabases that meets these challenges. In particular, we study road networks from several data sources. Our iterative algorithm matches semantically equivalent objects based on geometric and semantic attribute similarity measures. Relational similarity helps to solve difficult situations by exploiting the underlying graph structure of road networks: Already confirmed neighbouring matchings improve the similarity value of a given matching. Adaptability to diverse input data is reached by combining and weighting subsets of similarity measures. A greedy approach and an efficient end-to-end-system built upon simple and flexible components outperform previous systems in terms of runtimes while showing matching results of high quality. Scalability to large geodatabases is supported by a partitioning framework together with parallel processing. We have experimentally verified our approach with large real-world datasets.

1 Spatial Data Integration and its Challenges

Spatial data are nowadays collected and maintained by numerous organizations. Despite the complexity of establishing an own spatial dataset, this task is motivated by a variety of reasons: Public authorities have to fulfill their public contract like establishing a cadastral information system with exact spatial reference data. Companies see the economic advantages of having a proprietary dataset that specializes to their own application requirements and outperforms competitors. Finally, an open data project like *OpenStreetMap (OSM)* aims at making geographic information freely available for everyone.

These different stakeholders and their motivations result in a high degree of diversity between spatial databases. While federal data aims at high spatial exactness and completeness, datasets for creating printed maps intentionally abandon these aspects in favor of better readability. Here, *cartographic generalization* is used to simplify the real world and to align objects for an eye-friendly visualization, e.g., on paper or small screens. Finally, car navigation systems are highly depending on reliable semantic attributes, e.g., speed limits or turning restrictions, for a safe and efficient routing. All in all, differences in spatial exactness, completeness, currentness, availability and quality of attributes, etc. should be considered when analyzing and comparing geodatabases.

For example, in figure 1, geometric differences between multiple sources can be observed in visualizations of small map excerpts. A complex road network together with severe differences in object modeling shown in figure 1a makes an object-wise comparison obviously difficult, even though the same real-world situation is pictured. However, for figure 1b (showing a simple and uniform road network) this task is significantly easier. We conclude that comparability between input data highly depends on the source characteristics.

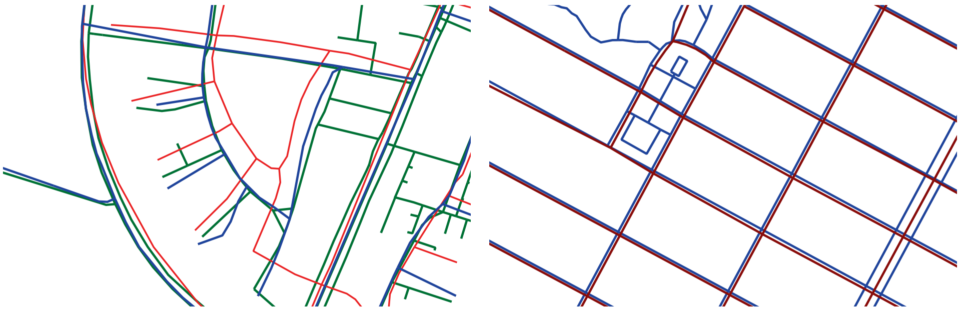

(a) Complex and diverse (Munich; OSM, ATKIS, D20) (b) Simple and uniform (Manhattan; OSM, TIGER)

Figure 1: Comparison of road networks from different geographic data sources.

With this **high degree of diversity**, we have identified one of the main challenges for *integrating* multiple geodatabases. However, this aspect is also the main cause for benefits of successful spatial data integration. The more differences we can spot between input sources, the more possibilities arise to enhance the sources with each other's information: After identifying and linking different representations of the same real-world entity, *information fusion* can be used to enhance one dataset with information from another one. E.g., speed limits don't have to be collected by a time-consuming exploration of the actual roads, but can be easily transferred from an existing database already containing this information. Next, *change detection* can be used to update contents when sources of different currentness are integrated. Objects that cannot be matched reveal added, removed or updated objects in either dataset.

As seen in figure 1a, different representations of the same real-world entity usually cannot be identified obviously. Spatial data lack global identifiers like ISBN numbers for books, and there is no notion of equality between database objects. Thus, we have to identify *semantically equivalent* objects based on *similarity* between objects in order to create an instance-level matching. A combination of several *similarity measures* has to be applied to classify objects as matching or non-matching based on their degree of similarity.

An algorithm for matching spatial data of arbitrary sources has to be **adaptable** to input sources by allowing flexible combinations of similarity measures which correspond to the source characteristics. Since every spatial feature has a geometry attribute, various measures apply geometric computations. Working with geometric attributes is far more **complex** than handling strings or numeric data types. In order to achieve low runtimes, spatial matching algorithms have to operate very **efficiently**. Keeping the overall algorithmic structure, the measures, and other components **simple**, also helps to improve efficiency and to guarantee fast processing of datasets. Finally, spatial databases are usually **very large**. They can easily grow to millions of objects when spatial features like roads, railways, or buildings for a whole country are considered. Therefore, approaches for matching large datasets need to be extremely **scalable** to handle this amount of data.

In this paper, we present our SimMatching algorithm to match spatial databases containing road networks. We meet the previously identified challenges of spatial data integration with the three main goals of our algorithm:

1. **Adaptability** to **diverse input data**: Our algorithm highly relies on similarity between database objects. Appropriate similarity measures can be chosen and combined with respect to the input source characteristics. Next to semantic and geometric attribute-based measures, we also introduce a relational measure to incorporate context information to the matching process. Similarity of a possible matching rises in our iterative approach, the more directly connected objects have already been matched in previous iterations.

2. **Simplicity** and **efficiency** to process **complex objects**: We propose a greedy approach [CLRS09] for the best trade-off between short runtime and high result quality. Our approach is built from simple components tailored together in a straightforward iterative process with a clear algorithmic structure. We leave geometric computations to tailored libraries available for in-memory programming, thus avoiding non-optimized DBMS extensions. Of course, large datasets are provided by database systems. Results (i.e., links between objects) are stored there, too.

3. **Scalability** to **large datasets**: A partitioning framework supports our in-memory algorithm by splitting input data in manageable pieces. Thus, processing large datasets is feasible even with limited computation resources (e.g., main memory). Processing disjoint partitions in parallel, as well as optimizations to keep the number of match candidates low help to achieve scalability.

While existing approaches usually address only one or two of these goals, our contribution is an end-to-end system that is optimized for all three concerns. In the remainder of this paper, we first survey related work and position our approach in the next section. Section 3 introduces our algorithm with its strong key concepts, i.e., flexible similarity measures, well-dosed consideration of relational aspects, efficient blocking techniques, and partition-wise parallel processing. In section 4 we evaluate runtimes and result quality with real-world datasets. We show that our algorithm outperforms previous approaches in terms of runtimes and efficiency while delivering results of equally high or even better quality. Finally, we conclude this paper in section 5 and give a short outlook on future work.

2 Related Work

Information integration is a well-studied research area in the database community. Approaches to integrate *standard data* that contains strings or numeric values for, e.g., national census, the health sector (patient records), or business applications (customer records) reach back to the 1950s. Leser and Naumann [LN07] as well as Doan et al. [DHI12] provide comprehensive compendia on integrating distributed and heterogeneous data sources. The authors identify data integration problems (e.g., structural, semantic, and other conflicts) and present established solutions for creating a redundancy-free integrated dataset (e.g., schema and data matching). They also present relevant system architectures like *federated databases* which are introduced in detail by Conrad [Con97]. Several concepts of federated databases are considered for the system architecture presented in our paper. E.g., we keep the input sources autonomous and untouched by the integration process.

Rahm and Bernstein survey techniques for automatic *schema matching* [RB01]. Though, as our geographic data sources are structured rather simply with few attributes[1] we rely on manual schema matching. Thus, the task of *data matching* is more relevant for our work. Christen [Chr12] provides a great book on this research area. The whole matching process with its different steps like preprocessing, indexing/blocking, data comparison, and classification is explained. However, most techniques concentrate on standard data.

Most matching techniques only consider attributes of an object collection. However, several approaches prove that *relations* between objects can be very valuable for matching tasks. The clustering algorithm for relational entity resolution by Bhattacharya and Getoor [BG07] exploits relations between cooccurring entities to support attribute-based matching. E.g., for bibliographic data, relations exist between papers and authors. Two representations of the same author are more likely to form a valid matching if the connected papers have already been successfully matched before. Next, Melnik et al. [MGR02] exploit relations from a graph representation for schema matching. Their similarity flooding uses a fixpoint computation to propagate similarity values to adjacent graph nodes. I.e., results gathered in earlier iterations influence connected objects. We propose a relational similarity measure for spatial data that is inspired by both papers.

Approaches for automatic spatial data integration have also been proposed in the context of *geographic information systems (GIS)* since the 1980s. Saalfeld [Saa88] coin the term *map conflation* for matching and aligning two geographic maps. Conflation results in a single map visualization that contains features and fused attributed of both maps. The matching task is focused on point features (road intersections) to find reference points for a local rubber-sheeting transformation. The latter aligns map features to the reference dataset by moving or distorting their geometries. The same authors extend their approach to consider semantic (if available) and topology information to improve the matching of corresponding point features. Line features (e.g., roads or rivers) are more significant for most maps than points. Therefore, several approaches propose to use their geometries as anchors, e.g., as in [DFE01]. This conflation approach exploits various criteria (e.g., shape and distance) to match line features.

[1]Most likely because ESRI shapefiles, a de-facto standard for file-based storage and exchange of spatial data, only allow a simple relational structure with one single attribute table.

All these classic map conflation approaches seem optimized for two fixed input sources. They are also primarily based on "visual" observations (derived from a map representation). The feature aligning is usually highly tailored to the matching step. With the possibility to keep several datasets in an integrated database instead of having to align them for a single map visualization, we strive to follow the architecture of a *federated database* [Con97] and prefer to leave input sources unchanged. Matchings between database objects representing the same real-world entity are determined by links between their original features. This also allows transferring geometries along links in either direction, though in a separate later fusion step.

With the wider availability of structured attributed vector datasets, Walter and Fritsch [WF99] propose a statistical approach for matching road networks. A growing buffer around a given road object identifies possible match partners (other roads lying within this buffer area). Several geometric characteristics are then employed to find a unique set of final matchings. The authors also consider matchings of higher cardinality than *1:1*, i.e., *n:m*-matchings: Objects have to be aggregated for matching, if sources differ in topology. The matching quality is satisfying for this approach, but very long runtimes make processing full real-world sources in reasonable time unfeasible.

Relying only on attribute-based characteristics for matching objects often leads to critical situations. Multiple nearby objects (e.g., parallel roads with similar shape) may lead to ambiguous matchings if a matching algorithm only considers a local perspective with autonomous objects. Context information – usually gathered from the dataset topology – can be incorporated into the matching process to improve matching results. Zhang [Zha09] suggests to follow continuous polylines, so-called "strokes", in road networks. Here, fewer geometric comparisons and no semantic information are exploited, but the graph structure of the dataset is of high importance. However, algorithms and data structures are highly optimized for specific sources and seem rarely adaptable to diverse input data.

Several other approaches are based on polylines as well. E.g., Safra et al. [SKSD13] concentrate on high performance with their "ad hoc" matching. They achieve low runtimes by simplifying line geometries to straight line representations and matching line endpoints only. Though, from our experience, such simple geometric comparisons just work for simple and homogeneous road networks. Lots of valid matchings are missing if generalization or missing crossroads lead to different geometric object modeling between input data.

Mantel et al. [ML04] adapt and improve the original *buffer growing algorithm* by [WF99] to integrate federal and proprietary spatial data within a project funded by the German Federal Agency for Cartography and Geodesy. Their core idea is to catch possibly matching objects within a buffer around increasing aggregations of given objects. Similarity computation is based on attributes only: Aspects of the geometry (e.g., location or length) and semantic information (e.g., object classes) are considered. Because structural information is neglected completely, result quality is critical for nearby, similar-shaped lines.

Tiedge et al. propose a *graph matching algorithm* [TL07] that operates on the underlying graph structure carried by every road network. Thus, they consider relevant context information. First, they compose a product graph from both input graphs. Next, the product graph is reduced to an unambiguous set of resulting matchings by applying structural, topological, geometric, or semantic rules. Thus, the algorithm considers attribute and relational (graph-based) similarities, but encodes them rather implicitly. Although this combination significantly enhances result quality, the dominance of graph handling leads to high runtimes and to a highly complex system of reduction rules.

Within the project, our group furthermore gathered some experience in handling larger datasets: A framework for partitioning and recomposition tasks is proposed by Warneke et al. [WL12]. Their techniques make processing data for a whole federal state of Germany feasible – though, with runtimes of several hours when using the graph matching algorithm.

Based on this know-how, we conclude that only a combination of geometric, semantic, *and* relational criteria can guarantee high quality matching results for integrating arbitrary datasets. Furthermore, efficient and simple components make processing large datasets feasible. We have also learned that spatial databases are slow for geometric computations and geometry-changing operations. Therefore, we rely on in-memory processing on client machines and use optimized geometric libraries. On the server side we use a spatially enhanced database management system to store input data and matching results. Spatial indexes support efficient spatial queries to access partitioned input data.

3 Similarity-based Spatial Matching

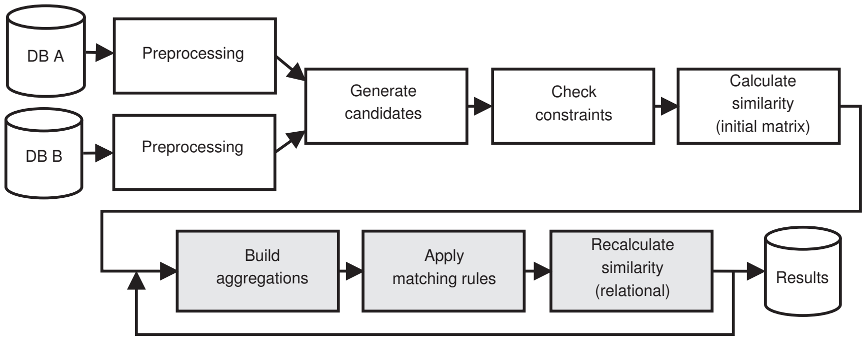


Figure 2: Flowchart of the overall SimMatching algorithm (without partitioning).

The *SimMatching* algorithm (figure 2) is mainly based on a selection of similarities between complex objects. To achieve both high efficiency and result quality, we have chosen a *greedy* strategy, i.e., choosing the locally optimal choice at each stage [CLRS09]. Our approach is also iterative to make use of previous results: Relational similarity rises, the more matchings in the vicinity of possible matchings are already confirmed as valid matchings. The algorithm is highly adaptable by configuring similarity measures, weights, thresholds, and other parameters. A partitioning framework is wrapped around the algorithm to split larger input data, process manageable pieces at a time, and recompose the results.

After preprocessing both input datasets, an initialization phase generates match candidates, validates them by checking constraints, and calculates initial similarity values. Then the algorithm enters its iterative part: For the currently best possible matching we build aggregations of higher cardinality to see if similarity improves by doing so. Finally, we apply matching rules within a classification step to confirm or reject matchings with respect to their similarity values. Afterwards, subsequent recalculations of similarity might be needed. The algorithm ends, if no more matchings can be confirmed. Next, all components of the algorithm representing its single steps will be introduced in detail.

Preprocessing

Errors in the input data and different object modeling rules require a *preprocessing* before the actual matching algorithms starts. Our SimMatching approach assumes that the given road network represents a *connected planar graph* in order to take advantage of reliable topological information. We remove small gaps or overlappings and segment lines at intersections to meet this requirement. Special treatment of multi-lane-roads helps to further improve comparability. In detail, the following steps are done for both input sources separately:

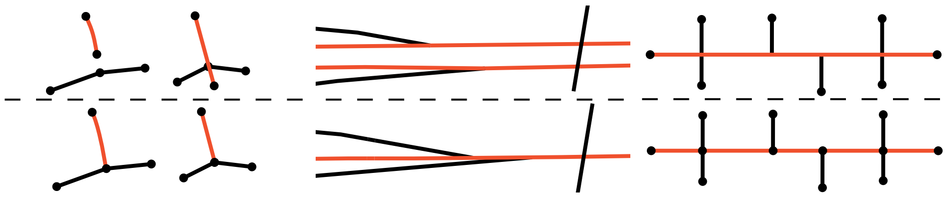

(a) Topological correction. (b) Multi-lane replacement. (c) Line segmentation.

Figure 3: Original (top) and preprocessed (bottom) input data.

Topological errors like small gaps (undershoots) and overlapping line ends (overshoots) can occur, if data is captured or modeled inexactly (figure 3a). E.g., hand-drawn vector maps might be styled by visual aspects only. Errors remain invisible for human operators if they use coarse tools or thick visualizations to create the skeleton of road networks. However, such errors make a dataset unusable for, e.g., routing purposes, when connections between lines are missing. We correct these errors by harmonizing coordinates for line endpoints and merging nearby points.

Some larger roads, e.g., highways, are modeled with **multi-lane-representations** in some sources, while others just use **single-lane-representations** (figure 3b). Here, we unify modeling rules and the dataset structure by replacing occurrences of parallel lines (with “one-way” attributes in the opposite direction) by a middle axis representation. Meeting lines (i.e., crossroads and highway ramps or exits) are adjusted to the new topological structure after computing the middle axis.

Next, a **line segmentation** algorithm (figure 3c) splits longer lines into smaller segments. Intersecting lines have to meet in known graph nodes that are represented by line endpoints only. This line segmentation also helps to unify object modeling rules. Choosing equally defined “smallest parts” for all input data helps to improve comparability. Our segmentation algorithm increases the number of preprocessed objects compared to the original dataset cardinality. If, e.g., two lines are intersecting each other, four segments are created as seen in figure 3c.

Finally, **neighbours** for every input object are calculated and stored for fast access during subsequent steps. We extract information about incident edges from the underlying graph representation of the road network. The resulting incidence lists are stored in an object-based representation in order to encode the graph structure.

Generate Candidates

The first step for every matching process is the generation of *match candidates*, i.e., pairs of objects which are considered possibly matching and therefore kept for further computations in subsequent steps. When matching two datasets of cardinality n and m respectively, the naive approach to build these candidates would be to calculate the *cartesian product* which means to compare every single object from the first source with every single object from the second source. Processing the complete resulting candidate set of size $n \times m$ is only feasible for small input data and does not scale to larger datasets. Thus, the amount of match candidates has to be reduced in the very beginning of the matching process.

Blocking techniques [Chr12] are applied to group the input objects into smaller subsets of the original data. A simple and fast function with very low computation costs is needed to achieve this pre-sorting. E.g., for customer records, it might be useful to consider only customers with a coincident first letter of the surname as match candidates. This obviously reduces the amount of considerable candidates, but also includes the possibility of loosing valid matchings, if, e.g., a spelling mistake appears exactly in the data used for blocking.

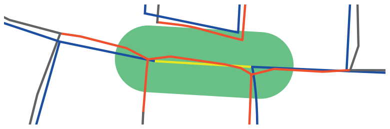


Figure 4: Match candidates are generated within a fixed-size buffer.

For road networks, we can use the spatial position of objects to generate match candidates only in the vicinity of a given object. Even for sources with a high level of generalization or inaccurate spatial exactness, it is very unlikely to find valid matchings for objects exceeding a certain distance to each other. We call this the *locality* of matchings. A **buffer** is a fixed-size area around a given object. All other objects that are completely contained inside this area or at least intersect its extent are considered as relevant vicinity of a given object. In figure 4, the green region around the currently evaluated yellow line represents the buffer area. Only those roads from the other dataset which are intersecting the buffer area (red lines) are chosen as match candidates and kept for further computations. To support adaptability, the buffer size can be set with respect to the input data. Section 4 explains heuristics to adjust the buffer size to given inputs. A fixed size buffer (independent from the cardinality of input data) helps to achieve a small and most notably *constant* number of match candidates per input object which is a crucial condition for scalability.

Check Constraints

Wagstaff et al. [WCRS01] introduce *constraints* to enhance clustering algorithms: On the one hand, *cannot-link constraints* specify that two instances must not be placed in the same cluster; on the other hand, *must-link constraints* specify that two instances have to be in the same cluster. We adapt this basic idea and both groups of constraints. Constraints improve the overall matching results and reduce the number of match candidates even further. *Must-match* and *cannot-match* constraints are used to classify matchings obligatorily as confirmed or rejected. Relying exclusively on similarity measures when comparing objects can cause problems (even if a combination of different similarity measures respects multiple characteristics): A match candidate might still obtain a high similarity value, even if one or more measures return complete dissimilarity. Imagine real-world situations where field tracks run in parallel to a highway. Even though object classes and street names do not match at all, geometric similarity (e.g., length, angle, etc.) can be high enough to erroneously match a field track and the highway. Thus it is desirable to formulate stronger conditions for matchings that can rule out similarity values.

We mainly use *rule-based* cannot-match constraints. Incorporating these rules as early as possible helps to reduce the amount of match candidates and thereby improves the overall performance. E.g., our `ObjectTypeConstraint` blacklists invalid combinations of object types and rejects candidates accordingly to these rules. Also, minimum similarity values for criteria evaluated by similarity measures can be forced by rules. E.g.:

> "Reject all pairs of objects with a corresponding line length of less than 20%."

Next, *instance-based* constraints of either type that list specific object combinations (a_i, b_j) are important for manual interaction. In preprocessing (before the actual matching algorithm starts) and postprocessing (after the algorithm stopped) these lists of object instances are queried to force and deny matchings respectively. Human operators build these lists when they correct a matching result or clarify difficult matching situations in advance to support the algorithm in its decisions.

Calculate Similarity

Matching algorithms usually rely on *similarity* between objects to identify multiple representations of the same real-world entity. Thus, the degree of similarity between two given objects needs to be expressed by a numerical value. *Similarity measures* apply a *similarity function* to given input data that returns a *similarity value*. As in, e.g., [Chr12] and [CMZ09], we calculate a numerical similarity value $s = sim(a,b)$ for objects or attributes a and b with a similarity function in a *metric space*. General requirements for these metrics may include symmetry ($sim(a,b) = sim(b,a)$) or other conditions, if needed [CMZ09]. Similarity functions should be normalized (i.e., $s \in [0,1]$) before combining multiple measures. For standard data like strings, various techniques from letter-wise or common substring comparison to phonetic similarity have been introduced [Chr12].

For our algorithm, we need to handle more complex data and introduce more sophisticated *attribute-based* and *relational* similarity measures to compare objects. The former take one single aspect of an object into account. For geodatabases we can evaluate either *semantic* attributes like, e.g., street names (strings) and speed limits (numeric values), or the *geometric* attribute. We can derive several aspects from a geometry attribute, e.g., spatial position, shape, size, etc. which might all be used by specific measures. Relational measures highly depend on relationships between entities, i.e., context information available in a domain. In [BG07] relational similarity enhances a clustering algorithm to match bibliographic data based on the following idea: If two representations of a book, each with an associated author, match, it is easier to match other persons in a co-author-relationship for the given books. For geographic data, the topology of datasets (i.e., the underlying graph structures of road networks) can be used to inspect topological relations between spatial objects.

Instead of relying on a single measure to compute a similarity value for two given objects, we use a combination of multiple measures. A *weighted sum* combines multiple geometric and semantic attribute-based as well as relational measures *sm*:

$$sim(a,b) = \sum \omega_i * sm_i(a,b), \ \sum \omega_i = 1$$

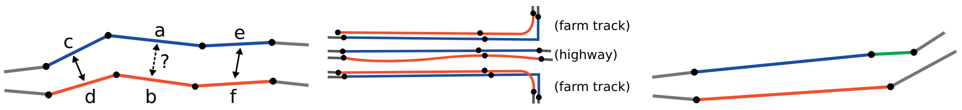
It is important to find a combination of measures covering multiple aspects of spatial objects. I.e., at least one measure each should take care of spatial position, shape, semantic properties (if available), and context information. Some of these aspects are correlated, e.g., the line length of course depends on the shape of an object. Though, our experience has shown that even a combination of correlated similarity measures can help to improve similarity computations. Next, we introduce specific similarity measures from our library of measures:

LengthSimilarity: Using the geometry of every feature we compute a numerical value for the *line length*. We relate the shorter line length ($length_{inf}$) to the longer one ($length_{sup}$) to compute a normalized similarity value: $sim_{length}(a,b) = {}^{length_{inf}}/_{length_{sup}}$.

AngleSumSimilarity: Following a single line geometry from start to end we survey the change in direction at each intermediate point. Angles α_i between incident line segments are summarized in a normalized manner ($\pi - \alpha_i$). Clockwise and counter-clockwise turns with positive and negative values that might neutralize each other are honored. Comparing the sum of all α_i for two line objects as above returns the desired similarity value.

StreetNameSimilarity: Street names are commonly used semantic attributes for road network data. Names are usually stored as simple string values. Thus, generic string similarity measures can be applied. E.g., we apply the well-known Levenshtein distance [Chr12] with fixed costs of 1 for each edit operation. Similarity values are normalized to the interval $[0,1]$ with respect to the longer string length: $sim_{streetname}(a,b) = 1 - \left({}^{levenshtein(a,b)}/_{maxLength(a,b)}\right)$. Enhancements are added by resolving abbreviations commonly used in street names, i.e., *st./street*, *rd./road*, etc.

DirectNeighbourhoodSimilarity: We propose a *relational* measure that considers already confirmed matchings in the direct vicinity of a given match candidate to evaluate its similarity. The general idea is that similarity rises, the more directly connected matchings (*neighbours*) have been already confirmed as valid matchings. This is also justified by the *principle of continuity* [HGM02], used in data visualization. It suggests that continuous shapes are usually recognized as a unit. In the sense of creating a continuous, closed line of confirmed matchings, a possible matching in between two already confirmed matchings should be confirmed (figure 5a). Additionally, situations with multiple possible matchings can be caused by (parallel) road segments with similar appearance (figure 5b). These situations can usually be solved if neighbouring roads are inspected first and the critical situation is matched afterwards. Finally, relational similarity can help to close small gaps (figure 5c) to create a continuous network of matchings from a global perspective.



(a) Solving critical situations with low attribute similairty. (b) Solving ambiguous situations. (c) Closing gaps in the graph structure.

Figure 5: Matching situations showing the usefulness of our relational similarity measure.

We can gather topological context information by exploiting the road network's underlying planar graph structure. Incident edges represent neighbouring line objects that share a common starting or end point. Previously confirmed matchings are available due to the iterative nature of our algorithm. We calculate the similarity value by relating confirmed neighbouring matches to all (i.e., confirmed and possible) neighbouring matches.

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
b_1			0.95							
b_2										
b_3					0.92				0.23	
b_4			0.4				0.8			
b_5	0.1									

Figure 6: A sparse 5×10 matrix with similarity values for objects from datasets A and B.

We store all possible matchings and their similarity values in a data structure called *similarity matrix*. Though, the number of possible matchings for each input object is small due to the heavy use of blocking techniques and constraints for optimization purposes. Thus, the complete $n \times m$ matrix is a rather sparse matrix as shown in figure 6: Here, only 6 out of 50 matrix elements contain similarity values, while the rest remains empty (grey cells). Sparsity increases even more, the larger n and m grow. We have chosen a priority queue to implement the "similarity matrix" to guarantee fast access to the currently best matching: Only relevant matrix entries – possible matchings with their attached similarity value – are stored, always accessible from high to low similarity. Empty entries are omitted (figure 7). Thus, the priority queue optimizes our algorithm in terms of space and time.

0.95	0.92	0.8	0.4	0.23	0.1
(a_3, b_1)	(a_5, b_3)	(a_7, b_4)	(a_2, b_4)	(a_9, b_3)	(a_1, b_5)

Figure 7: A priority queue stores matchings and their similarity values.

Implementing the Greedy Paradigm

The SimMatching algorithm now enters its iterative part which is pictured as a loop in figure 2. In every iteration, the currently best matching, i.e., the matching with the highest similarity value, is selected from the priority queue. Here, the algorithm implements the *greedy paradigm* of making the locally optimal choice at each stage. Difficult decisions are postponed for later. For matching tasks it is not critical to find not always the global optimum. Thus, this approach promises the best trade-off between runtime and result quality. The ordered storing of matchings in our similarity matrix (efficiently implemented as priority queue) supports the greedy strategy.

Build Aggregations

With our preprocessing step and especially the line segmentation algorithm, we try to unify geometric object modeling as much as possible. However, there are lots of remaining situations where *1:1*-matchings do not reflect the real-world situation. These situations require a special form of many-to-many-matchings. Imagine a longer road that is intersected by two smaller crossroads in one dataset which results in three segments after preprocessing. If these crossroads are missing in the other dataset, e.g., because of a lack of completeness, there is no reason for a segmentation. A *3:1*-matching is required to reflect the real-world situation. We introduce *n:m*-matchings by allowing *aggregations* of objects.

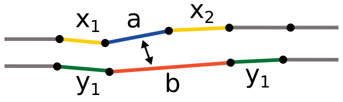


Figure 8: Possible aggregations for matching (a,b).

All neighbouring objects x_i form possible aggregation candidates $a+x_i$ for a given object a. An aggregation $a+x_i$ is considered valuable and added to the similarity matrix, if the similarity value of matching $(a+x_i,b)$ exceeds the former similarity value of matching (a,b). Otherwise, it is immediately rejected. In the next iteration of our algorithm, this step is repeated. We now consider matchings $(a+x_i+x_j,b)$ and $(a+x_i,b+y_i)$ for neighbouring objects x_j and y_i. Here, we follow the greedy paradigm again: Only the best matching and then the best aggregation are considered. If this step is repeated while similarity increases, aggregated objects can grow to n or m parts, which build a new object. The basic algorithm requires no changes as we handle the resulting *n:m*-matchings as *1:1*-matchings between aggregations.

Apply Matching Rules

In the final step of each iteration, *Matching rules* classify all possible matchings – one matching per iteration. The greedy strategy suggests to classify the currently most confident choice first. In each iteration, only the currently best matching, i.e., the top of the similarity matrix, is either confirmed or rejected. Matching rules usually base this decision on similarity values and other knowledge gathered in previous steps. Though, they may also incorporate new information that is considered by the rules. Combined with all implicit rules set by constraints, simple threshold rules provide excellent results. E.g.: "Possible matchings are confirmed while their similarity value is higher than 0.75". The threshold depends on input data, selected measures, and their weights (see section 4).

We perform various *clean up* operations for the similarity matrix, each time a matching (a,b) is confirmed: The matching itself and all possible matchings including a or b are removed from the matrix, thus implicitly rejected. We furthermore remove aggregated objects built from a or b, or parts of a or b if either are aggregations themselves. These operations avoid conflicts in the final matching result which could appear if object a is part of multiple confirmed matchings. They also help to decrease the size of the similarity matrix with every classification step.

Classification continues while more matchings can be confirmed. The algorithm exits its iterative part and ends if there are no more matchings that fulfill the given matching rules. I.e., for threshold rules, all remaining possible matchings have a similarity value lower than the given threshold. The set of confirmed matchings together with two sets of unmatched objects – one for each input source – are called *matching result*. Due to previous clean up operations performed on the similarity matrix the result is always unambiguous and conflict-free: No object can be part of multiple matchings, multiple aggregations, matching and aggregation, or matched and unmatched at the same time.

Recalculate Similarity

By confirming more and more matchings based on their good attribute similarity, we can now start to take advantage of context information. Neighbouring objects profit from previously confirmed matchings as their relational similarity increases. In every iteration, we have to update the similarity matrix to the new context situation by locally *recalculating similarity*, i.e., updating relational similarity. Though, we do not recompute the whole matrix: Updates are needed only for the direct vicinity of the confirmed matching and are limited to all neighbouring objects of the confirmed matching.

Partitioning and Parallel Execution

Within our previous work [ML04, TL07] we experienced very long computation times, if matching computations – especially geometric computations – were executed directly inside a database management system with spatial extension. We therefore have designed our SimMatching approach as an *in-memory algorithm* running on standard client machines with dedicated geometry libraries, but also with restricted hardware resources. The amount of data that can be processed at once is limited by the size of the available main memory. To process larger input data, we rely on a *partitioning* framework [WL12] that splits larger input data into manageable pieces called *partitions*. Partitioning is justified by the locality of matchings as explained before. Runtime and memory consumption depend more on the number of processed objects (the cardinality of the input datasets) than on the size of the processed area. Thus, we propose an *adaptable partitioning system* that adapts the number and size of partitions to the number of contained objects.

All objects inside the extent of a certain partition are queried from a spatial database system by a spatial range query supported by a spatial index. After applying our matching process in-memory, a *recomposition* phase combines the results for multiple partitions. Conflicts, especially near partition borders, have to be resolved before writing the results back into the database. We expand every partition with a buffer operation to create a soft border that contains context information for relational similarity measures. Besides this overlapping, partitions remain disjoint which allows us to process them independently in parallel. *Parallel execution* on multi-core systems leads to astonishing performance enhancements and decreases execution time proportionally to the number of used threads.

4 Results

In this section, we apply our SimMatching approach to various real-world data sources covering parts of Germany and the United States. We analyze these input sources with their characteristics and describe our testing configuration. Result quality is evaluated with quality measures from the information retrieval domain. Performance tests prove efficiency and scalability of our approach and allow a comparison to existing systems.

Real-world Input Data Description

ATKIS: The *Authoritative Topographic-Cartographic Information System* is established and administered by a federal German institution. It provides a semantically and geometrically rich large scale landscape model and is very well documented. An *object catalog* lists all available object classes together with their specific attributes and well-defined object modeling rules. Exactness of spatial positions is very important for this dataset.

OSM: *OpenStreetMap* is a collaborative open data project to create a free worldwide map with the help of a global community. OSM data can be geometrically precise and semantically rich if contributors follow the project guidelines. Though, data quality varies as both novices and experts contribute. A high number of volunteers frequently updates OSM data for high currentness. Owners of proprietary datasets might be interested to utilize this currentness for improving their data quality by matching.

D20: This proprietary dataset supplies cartographically generalized road networks for producing printed maps. Objects are shaped manually by visual aspects and do not follow a strict modeling scheme. Spatial exactness is needed only so far that signatures appear correctly. Besides street names and object classes, semantic richness is low.

NAV: Created for routing purposes, this proprietary dataset is mainly used in car navigation systems. Hence it includes only car-accessible roads that were highly cartographically generalized for on-screen presentation. Attributes specialized for driver's needs include, e.g., speed limits, turning restrictions, and many others, but just coarse object classes.

TIGER: Road network data for the United States are supplied by the US Census Bureau and released into the public domain. Data are of high currentness (yearly updates) and attributed with administrative information, e.g., ZIP codes, and address ranges. TIGER data was formerly used as base for US OSM data, but differs now after edits in both sources.

Heuristics for parameter optimization

Parameters for our algorithm can be customized for every matching task and thereby support adaptability and efficiency of the algorithm. E.g., matching OSM with ATKIS data requires different parameters than matching OSM and D20 data due to the diversity of the input sources. Configurable parameters are the buffer size for candidate generation, the choice and the weights of similarity measures, and the threshold for classifying matchings. Finding appropriate parameter values starts with a detailed analysis of the input sources and their characteristics. The buffer size depends on the exactness of spatial positions in the input data and the resulting distance between different road representations. A higher degree of cartographic generalization (e.g., see figure 1a) requires larger buffers than more comparable sources (e.g., see figure 1b). Measuring average distances on a sample basis and adding few meters tolerance is an appropriate choice. The buffer size influences efficiency which depends on a suitable amount of match candidates. Too large buffers result in a too high number of candidates – too small buffers cause missing matchings.

Weights for similarity measures depend on the availability, significance, and quality of attributes. Geometric measures tend to be more unreliable with stronger generalization. Street names might be incomplete in many datasets and are then not usable for similarity computations. Measures based on object types should be weighted lower, if there are just few and coarse object types available, like in our NAV dataset. These heuristics give an evidence of a "good" choice of measures and weights. They should be empirically verified and further optimized by test runs with subsequent plausibility checks. Finally, the threshold can be approximated by manually classifying matchings repeatedly as confirmed or rejected. After doing this on a sample basis, the threshold converges to a value lower/higher than all similarity values of accepted/rejected matchings.

Testing Configuration

The SimMatching algorithm is implemented in *Java 7*. *OpenJUMP*, an open source Geographic Information System (GIS), is used to visualize input data and matching results. It provides an extensive class library for spatial data types (based on the *JTS Topology Suite*). On the server side we use a *PostgreSQL 9.3* database with the spatial *PostGIS* extension and its geometry data types and functions. Java thread pooling helps to prepare tasks for each partition, schedule their execution and merge threads for result collection. We also use pooled database connections to allow one exclusive connection for each active thread. Of course, (spatial) indexes are used extensively, particularly to speed up range queries to select all features inside a given partition.

We chose D20 and OSM as input sources with high diversity for testing purposes. Parameters were optimized following the heuristics introduced in the former section. We achieved best results for matching D20 and OSM data when using the following measures with their respective weights and a threshold of 0.55: `LengthSimilarity` (0.3), `AngleSumSimilarity` (0.1), `AngleSimilarity` (0.1), `Starting-` and `EndPointSimilarity` (0.05 each), `StreetNameSimilarity` (0.1), and `DirectNeighbourhoodSimilarity` (0.3). All tests were performed on regular desktop hardware based on a year 2013 quad core CPU.

Result Quality Evaluation

We evaluated the quality of matching results created by our SimMatching approach by inspecting a visualization of matchings, unmatched objects, and original input data for either source. Figure 9b shows a small map excerpt containing matchings (green) and unmatched objects – red objects originating from the dataset pictured in figure 9a and blue objects originating from the other (figure 9c). Most objects are matched correctly. There are correct unmatched objects as well: The single red and blue objects miss a respective counterpart from the other dataset and therefore shall remain unmatched. However, on the far left side (figure 9b) two roads showing a high geometric similarity are also unmatched. After inspecting other attribute data not pictured here, we have to decide if the algorithm failed here and classify the matching possibly as missing, i.e., not found by our algorithm.

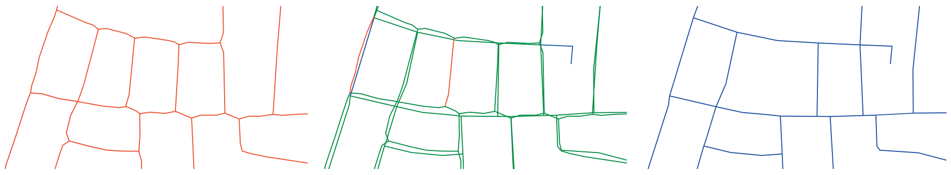

(a) Input data "red". (b) Matchings (green) and unmatched objects (red and blue). (c) Input data "blue".

Figure 9: Visualizing input data and a matching result.

For a sophisticated evaluation of result quality, we manually analyzed a map excerpt around *Munich*, Germany, for OSM and D20 data. Our dataset *Munich* (figure 10) covers an area of approximately 7.2 km^2. There are 1 065 OSM and 1 104 D20 objects contained in the original input data which resulted in 2 920 OSM and 1 114 D20 objects after preprocessing. Using a process configuration as described in section 4, a total of 969 matchings were confirmed and the execution took below 4 seconds.

We applied measures known from the information retrieval domain, *precision* and *recall*, to have a comparable quality measure. Precision indicates the rate of correctly classified confirmed matchings in all found matchings. We identified 32 situations where wrong matchings were confirmed or a better solutions (e.g., a larger aggregation) would have been more suitable. In relation to a total of 969 matchings, we reached a precision value of 97%. Next, recall describes the rate of correctly classified confirmed matchings found in relation to all valid matchings. Considering missing and non-optimum matchings, we determined a recall value of 97% as well. In a previous evaluation [SL14], both values evaluated to 95% each. An enhanced preprocessing step with multi-lane handling as well as tuned measures and constraints, slighly improved these values.

Performance Evaluation

For a comprehensive performance evaluation we have matched several map excerpts of increasing size for different regions of Germany (figure 11). The number of objects is determined by the cardinality of the larger database (here: OSM) before preprocessing.

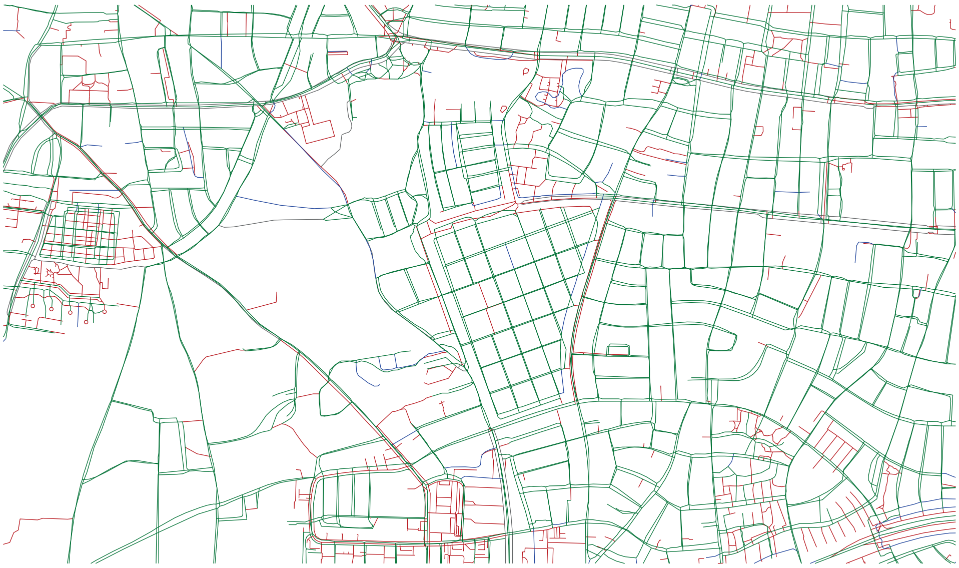


Figure 10: Matchings (green), unmatched OSM objects (red), and unmatched D20 objects (blue) for testing area *Munich*.

Total runtimes include partitioning, loading data from a spatial database, preprocessing, the actual matching step, and finally recomposing and writing back the results. All given times are average values generated from 20 independent test runs per dataset.

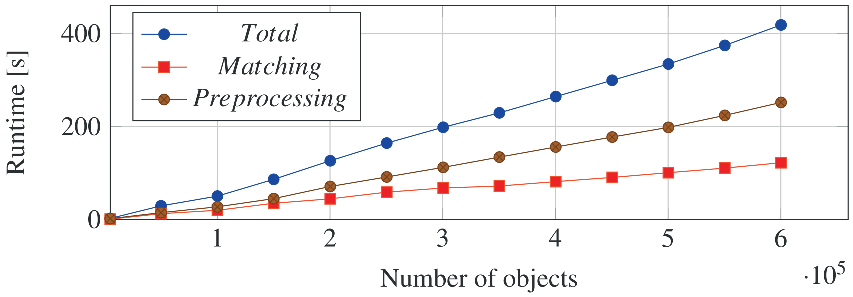


Figure 11: Runtimes for input data of increasing size.

While we see total runtimes below one minute for 100 000 input objects, they increase to below three and a half minutes for 300 000 objects. The largest dataset of 600 000 objects took approximately seven minutes to be fully processed. All in all, we assume a linear runtime for our approach based on these experiments. Therefore we have proven excellent scalability to very large geodatabases. Likewise, the two listed subprocesses show a linear time complexity, too. Preprocessing takes almost twice the time of the actual matching as more geometry-changing operations are performed. This step also includes more database communication as preprocessed data has to be written to the spatial database system and is organized there, i.e., by enabling database constraints and (spatial) indexes.

Performance Comparison

Directly comparing the results of our performance tests with other approaches is usually not feasible as most input data is distributed under proprietary licences and not available to be tested with our system. Next, the cardinality, dimension, and complexity of used testing areas is not always clear. We also have to consider the evolution of hardware resources when comparing runtimes. However, we try to respect test results and complexity analysis whenever they are available. We propose the unit *objects per second* for measuring how many input objects can be processed in one second by the respective algorithm.

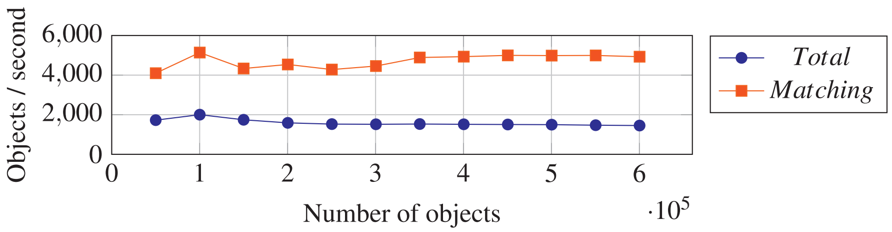


Figure 12: Processed objects per second for input data of increasing size.

The SimMatching approach is able to process between 1 450 and 2 000 objects per second based on the total computation time (figure 12). For the actual matching only, the values increase to 4 100 to 5 100 objects per second (figure 12). There are minor variations for the different datasets as many factors, that cannot be easily unified, influence the algorithm performance. Next to the cardinality this includes, e.g., the network complexity. On average, values of 1 650 and 4 700 (matching only) objects per second are reached.

The original buffer growing algorithm [WF99] was used to process a tiny test area of $2km \times 2km$. The authors expected an exponential time complexity. Even for the small map excerpt containing only 963 objects, the computation time was almost as high as 7 hours. This evaluates to a performance value of 0.2 objects per second. For the enhanced buffer growing [ML04] developed by our group, internal technical reports show an increase to 1.2 objects per second. The graph matching [TL07] is able to process about 10 objects per second for a dataset of 100 000 objects in a map excerpt of $40km \times 40km$.

In 2009, Zhang [Zha09] tested his approach on German datasets of similar size like in figure 11. He assumed a linear time complexity – dependent on the number of input objects. His approach was experimentally proven to be scalable to large datasets. A matching performance of 500 objects per second is mentioned in [Zha09]. The *ApproxMatching* algorithm proposed in [SKSD13] for ad hoc matching was optimized for performance. With their fastest variant (not necessarily bringing the best recall and precision values), the authors were able to match 18 264 objects in about 110 seconds. This evaluates to a total of 166 objects per second. However, it is not clear if, e.g., the creation time for used indexes or communication costs were included in these values. Next, the input data (Manhattan, similar to figure 1b) seemed to be of low structural complexity. The authors concluded an "almost linear" runtime, but did not test any larger datasets than the one mentioned above.

All in all, we can conclude that our SimMatching approach outperforms selected existing approaches in terms of matching performance. Even algorithms that are optimized for processing input data in short time [Zha09, SKSD13] and that are estimated to have a linear time complexity with respect to the cardinality of input data, perform worse in terms of processed objects per second.

5 Conclusion and Future Work

In this paper, we have presented an adaptable, efficient and scalable solution to the challenging task of integrating diverse, complex and large geodatabases. Our SimMatching approach provides an end-to-end-system based on a simple algorithmic structure with flexible components and configurable parameters. By choosing appropriate similarity measures and respective weights, the algorithm adapts to arbitrary input data. A combination of geometric, semantic and relational measures produces high quality results in terms of recall and precision when applying the algorithm to several real-world data sources. In each iteration, previously confirmed matchings and context information from the underlying graph structure of road networks help to improve matching decisions in critical situations. Though, without any negative performance impact seen elsewhere when graph handling becomes dominant.

We extensively use blocking with spatial buffers and other techniques to keep the number of candidates low. This guarantees short runtimes and an excellent matching performance. This is further supported by a greedy strategy to process locally best solutions first. We have shown that our algorithm outperforms other approaches in terms of processed objects per second while result quality stays competitive. This efficiency together with a partitioning framework wrapped around the actual matching algorithm leads to a high scalability to large datasets. Tests with input data of increasing cardinality have shown that the algorithm scales very well. We habe experimentally verified a linear runtime complexity.

For future work, we have to validate the linear runtime observed in our experiments with an in-depth theoretical complexity analysis. Furthermore, we plan to employ and improve machine learning techniques for configuring our approach. Currently, parameters are manually optimized: Appropriate combinations of similarity measures have to be chosen and other parameters have to be adjusted dependent on the input data. This is a time-consuming task, which is done by a domain expert. First tests with Support Vector Machines (SVM) have been promising and have shown that weights are converging when enough sample data is supplied by a human operator within an active learning system.

References

[BG07] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):Article 5, 2007.

[Chr12] P. Christen. *Data Matching*. Springer, 2012.

[CLRS09] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction To Algorithms*. MIT Press, 3rd edition, 2009.

[CMZ09] S. Chen, B. Ma, and K. Zhang. On the similarity metric and the distance metric. *Theoretical Computer Science*, 410(24-25):2365–2376, 2009.

[Con97] S. Conrad. *Föderierte Datenbanksysteme*. Springer, 1997.

[DFE01] Y. Doytsher, S. Filin, and E. Ezra. Transformation of datasets in a linear-based map conflation framework. *Surveying and Land Information Systems*, 61(3):165–176, 2001.

[DHI12] A. Doan, A. Halevy, and Z. Ives. *Principles of Data Integration*. Elsevier, 2012.

[HGM02] G. Hake, D. Grünreich, and L. Meng. *Kartographie*. De Gruyter, 2002.

[LN07] U. Leser and F. Naumann. *Informationsintegration: Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen*. Dpunkt.Verlag GmbH, 2007.

[MGR02] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: a versatile graph matching algorithm and its application to schema matching. In *Proceedings of the 18th International Conference on Data Engineering*, pages 117–128, 2002.

[ML04] D. Mantel and U. W. Lipeck. Matching Cartographic Objects in Spatial Databases. In *Proceedings of the 20th Congress of the ISPRS*, pages 172–176, 2004.

[RB01] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.

[Saa88] A. Saalfeld. Conflation – Automated map compilation. *International Journal of Geographical Information Systems*, 2(3):217–228, 1988.

[SKSD13] E. Safra, Y. Kanza, Y. Sagiv, and Y. Doytsher. Ad hoc matching of vectorial road networks. *Int. Journal of Geographical Information Science*, 27(1):114–153, 2013.

[SL14] M. Schäfers and U. W. Lipeck. Spatial Data Integration Using Similarity-based Matching. In H.-M. Haav, A. Kalja, and T. Robal, editors, *Proc. of the 11th Int. Baltic Conference on Databases and Information Systems*, pages 89–100. TUT Press, 2014.

[TL07] M. Tiedge and U. W. Lipeck. Graphbasiertes Matching in Räumlichen Datenbanken. In *Proc. of the 19th GI-Workshop Grundlagen von Datenbanken*, pages 42–46, 2007.

[WCRS01] K. Wagstaff, C. Cardie, S. Rogers, and S. Schroedl. Constrained K-means Clustering with Background Knowledge. In *Proceedings of the 18th International Conference on Machine Learning*, pages 577–584, 2001.

[WF99] V. Walter and D. Fritsch. Matching spatial data sets: a statistical approach. *International Journal of Geographical Information Science*, 13(5):445–473, 1999.

[WL12] H. Warneke and U. W. Lipeck. Ein Partitionierungsdienst für Geographische Daten in Räumlichen Datenbanken. In *Proceedings of the 24th GI-Workshop Grundlagen von Datenbanken*, pages 35–40, 2012.

[Zha09] M. Zhang. *Methods and Implementations of Road-Network Matching*. PhD Thesis, Technische Universität München, 2009.