Qualitative Comparison of Geocoding Systems using OpenStreetMap Data

Konstantin Clemens

Service-centric Networking
Telekom Innovation Laboratories
Technische Universität Berlin, Germany
konstantin.clemens@campus.tu-berlin.de

***Abstract*—*OpenStreetMap* is a platform where users contribute geographic data. To serve multiple use cases, these data are held in a very generic format. This makes processing and indexing OpenStreetMap data a challenge. *Nominatim* is an open source geocoding system that consumes OpenStreetMap data. Nominatim processes OpenStreetMap data well. It relies on predefined address schemes to determine the meaning of various address elements and to discover relevant results. Nominatim ranks results by a global precomputed score. *Elasticsearch* is a web service on top of *Lucene* – a general purpose document store. Lucene searches for documents and ranks results according to a *term frequency – inversed document frequency* scoring scheme. In this article, Nominatim is compared to two systems populated with exactly the same data: An out-of-the-box instance of Elasticsearch, and a specialized system that builds on top of Elasticsearch, but implements a custom algorithm to aggregate house numbers on every street segment, thereby vastly reducing the index size. The three geocoding systems are throughly benchmarked with three different data sets and geocoding queries of increasing complexity. The analysis shows: Term frequency – inversed document frequency based ranking yields more accurate results, and is more robust removing the need for predefined address schemes. Also, the reduced index size of the specialized system comes at a cost, which, depending on the application scenario, may be a viable option.**

Keywords–Geocoding; Address Indexing; OpenStreetMap; Nominatim; Elasticsearch

I. INTRODUCTION

This article is based on [1], which showed that generic document stores as Elasticsearch make a good baseline for geocoding services. A more detailed view on the experiments is presented here, additional experiments are introduced, as well as a new, more complex data set. Also, a specialized approach to reduce the index size of the document store is suggested in this article. It is evaluated in the same way as the two systems that have already been presented in [1].

OpenStreetMap [2] is a collaborative platform where every registered user can contribute relevant and mappable data. These data can be of arbitrary type: Street segments or bridges, their names, speed limits or surface materials, position and outline of buildings, forests, lakes or administrative areas, radio beacons and their operators, train tracks, bicycle, hiking or bus routes, etc. That variety is possible, because OpenStreetMap data is stored in a very generic data format without a specific application in mind.

OpenStretMap data consist of the three entity types *node, way*, and *relation*. Every entity type has an *id* attribute for referencing and the attributes *timestamp*, *version*, and *changeset* for versioning. There are also the attributes *uid* and *user* specifying the user who created or modified an entity.

Nodes have values for *longitude* and *latitude* in WGS84 format [3]. Therefore, nodes model single points on the globe. Because most things on a map have an areal extent, a node alone can be used to specify the position of an entry with yet unknown area. Also, for example, nodes can specify mountain peaks, magnetic and geographic poles, or other entities with no area. Most of the nodes, however, are parts of ways and relations.

Ways compose lines through points by specifying ordered lists of node references. A way can also specify an area by referencing the same node at the begin and at the end of the node list. Using ways it is possible to model car lanes on a street, pedestrian zones, simple outlines of structures, and the like.

Relations, in turn, may reference both ways and nodes. Therefore, relations can model complex geographic features as polygons with holes as well as specify, e.g., a center point for displaying pins on the map. Relations may also reference other relations assembling abstract entities that span several 'real things' such as

groups of islands, or universities with multiple, widespread buildings.

Finally, nodes, ways, and relationships can hold an arbitrary number of *tags*. Tags are key-value pairs that specify names, categories, address elements as city, district or street name, house number ranges, data sources, speed limits, and all other attributes of real-world features that the entities model.

Clearly, this data format is flexible enough to accommodate all kinds of data. For example, properly tagged relationships could hold 3D models of buildings, where different parts of buildings are different relations. [4] describes in depth how 3D models are stored in OpenStreetMap data. On one hand, that flexibility is convenient, but on the other hand it is also an obstacle: For many use cases the data need to be preprocessed before they can be utilized. In the case of geocoding, because of the versatile structure of OpenStretMap data, address elements are often spread across different entities. For example, a node might only be tagged with a house number, while the way that references this node only holds the street name information. The way itself may be contained in a postal code area represented by a relation. Another relation could describe the area of the city. However, the relations not necessarily reference the way or each other. Therefore, to offer a geocoding service, addresses need to be assembled out of OpenStreetMap entities first.

Geocoding [5] is the process of resolving named locations such as full addresses, named areas, and sometimes even landmarks into their location. A variety of proprietary and open source geocoding services, e.g., [6], [7], or [8] offer that through their APIs. First, to offer such a service, there need to be source data to be made searchable. These data need to contain a mapping from addresses or names to the geolocation of the entry, which, most often, is represented as a pair of WGS84 coordinates. Because errors in data are exposed through the service, the quality of data determines the quality of the geocoding service. To some extent, errors in the data can be covered when the data is being indexed: To account for errors and ambiguities in queries indexing algorithms need to store the data in a way that it is fuzzy and robust at the same time – some of these techniques can be applied to the source data at indexing time too. Geocoding services use indexes to parse and split queries into address elements, possibly determine their meaning and compile a list of candidate results the query may have referred to. In a last step, the service orders the candidate list so that the most probable result is on top. If possible, unlikely candidates at the bottom of that list are cut off. Depending on the specific type of geocoder, specialized approaches are used as described in [9], [10], and [11].

Nominatim [7] is an open source system implemented in several programming languages that uses OpenStreetMap data to provide a geocoding service. That means Nominatim preprocesses the OpenStreetMap data assembling full addresses from address elements spread in tags on various entities prior to building an index for geocoding. Both these processes are very time consuming, also because Nominatim precomputes global ranks for all entries at indexing time already. The source data, the geocoding index, and the ranks are all stored in a PostGIS [12] enabled PostgreSQL [13] data base. An apache server [14] invoking PHP to access the data base serves search requests via HTTP. When a search is performed, Nominatim first parses the query address according to predefined schemes. These schemes specify where various address elements as postal code, state, city, district, street name, or house number may be located in a query. Next, Nominatim queries the data base using the derived address elements. It collects candidate results ordered by the precomputed ranking score. In Nominatim there is no stage where result lists are ordered or cut in relation to a served query.

Term frequency – inverted document frequency (TF-IDF) [15], [16] is a formula to rank documents based on query terms and their distributions. For a given query term and a document, the term frequency is the number of occurrences of that term in that document. A higher term frequency implies that a document is more relevant to a query: The more often a query term appears in a document, the more likely the document corresponds to the query. At the same time, if a term rarely occurs in a document, the term may have been mentioned as a side note only, while the actual topic of the document could be a different one. Some terms are very common or have multiple meanings. Such terms appear in many documents therefore. Other terms are rare and specific. Clearly, rare and specific terms distinguish the relevant documents stronger than the generic and ambiguous ones. To incorporate that, a global term weight is computed for each term: The document frequency of a term is the number of documents that term occurs in. The more documents a term occurs in, the less distinguishable that term is. Thus, for each term a greater inverted document frequency implies a greater importance of its

term frequency value. The TF-IDF score for a term and a document is therefore computed by multiplying the term frequency and the inverted document frequency. For a query with multiple terms, the overall TF-IDF score is computed as the sum of scores of every query term for each document. There are variants of TF-IDF that differ in various details. Particularly, BM25f [17] is a variant that supports documents with differently weighted fields. Also, when computing the BM25f score, the document length is taken into account.

In contrast to Nominatim, *Lucene* [18] is a generic open source document indexing framework, that ranks results at query time. Lucene supports various ranking schemes for ordering results, including TF-IDF and BM25f. In the context of geocoding, where addresses are documents, one can easily agree that the token 'street' is less distinctive than the token 'Springfield', which – given that there are many towns called like that – is less distinctive than, let's say 'Chicago'. Therefore, for a query for 'Michigan Street Chicago' the 'Michigan Avenue in Chicago, Illinois' is scored higher than the 'Michigan Street in Springfield, Massachusetts'. The match on the rare token 'Chicago' outweighs the match on the common token 'Street'.

Elasticsearch [19] is a RESTful [20] wrapper around Lucene. That means, besides many other features it offers a simple HTTP based interface to create, read, update, and delete single documents as well as whole document collections. Internally a separate Lucene index is maintained per document collection, allowing Elasticsearch to expose a search interface as well. Thus, if populated with documents that contain addresses and their coordinates, Elasticsearch becomes an HTTP based geocoding service that uses BM25f to compute, which results match to given queries best.

A number of efforts was made to index OpenStreetMap data in Elasticsearch. The *elasticsearch-osmosis-plugin* [21] extends *Osmosis* [22], a tool for processing OpenStreetMap data, allowing it to index this data in Elasticsearch. Thereby, the entities are indexed as they are, the plug-in does not transform the data in any way. Instead it enables Elasticsearch to be used as a tool to browse original OpenStreetMap entities from the indexed source data set. *Pelias* [23] is a collection of modular tools that plug into each other. There are modules for reading data from OpenStreetMaps as well as other data formats, a module for indexing the data in Elasticsearch, and a module to offer a set of APIs and provide a geocoding service on top. Similar to the elasticsearch-osmosis-plugin, Pelias does not process the data in any way similar to Nominatim. Both systems do not harvest addresses that are spread across OpenStreetMap entities, but rather index the raw data in its generic format. *Gazetteer* [24] consists of two modules: One to parse OpenStreetMap data and derive points of interest, addresses, streets, street networks, and administrative boundaries. The other to index this data in Elasticsearch and thereby offer a geocoding service. Unlike the osmosis-elasticsearch-plugin or Pelias, the Gazetteer tool tries to assemble full addresses based on entities that contain one another, or are located nearby. Nominatim, as discussed earlier, is a geocoding service that does not rely on Elasticsearch, Lucene, or a dynamic ranking scheme. Instead, Nominatim tries to split queries into address elements in accordance to given schemes and ranks candidates by a globally precomputed score. As the Gazetteer, Nominatim is collecting address elements from various nodes as it is necessary with the generic OpenStreetMap data format.

For all the systems above, no qualitative analysis on their performance is available. These systems are best-effort solutions. In this article, a set of experiments is performed, which allows a qualitative comparison of geocoding systems based on BM25f as used in generic document stores and Nominatim, which relies on address schemes to split queries into address elements and derive their meaning. Existing solutions are using different data than Nominatim: They either do not preprocess OpenStreetMap data at all, or, like the Gazetteer solution, use own processing. Therefore, for this article, the data indexed in Elasticsearch is extracted directly from a Nominatim data base. This ensures that there are no data differences between the measured systems. Only differences between the various indexing algorithms are evaluated. In addition to Nominatim and Elasticsearch, an approach to reduce the index size is suggested in this article: Instead of indexing every house number address as a separate document, house numbers on the same street segment are aggregated into one document. Instead of WGS84 coordinates each document in this system contains a mapping table declaring the coordinates for every house number on that street segment. This solution requires a thin layer around Elasticsearch, which takes care of identifying and, if available, extracting the response for a requested house number. Elasticsearch is still used to retrieve street segments that match to queries. These three solutions are evaluated using the same benchmarks allowing to derive the strengths and

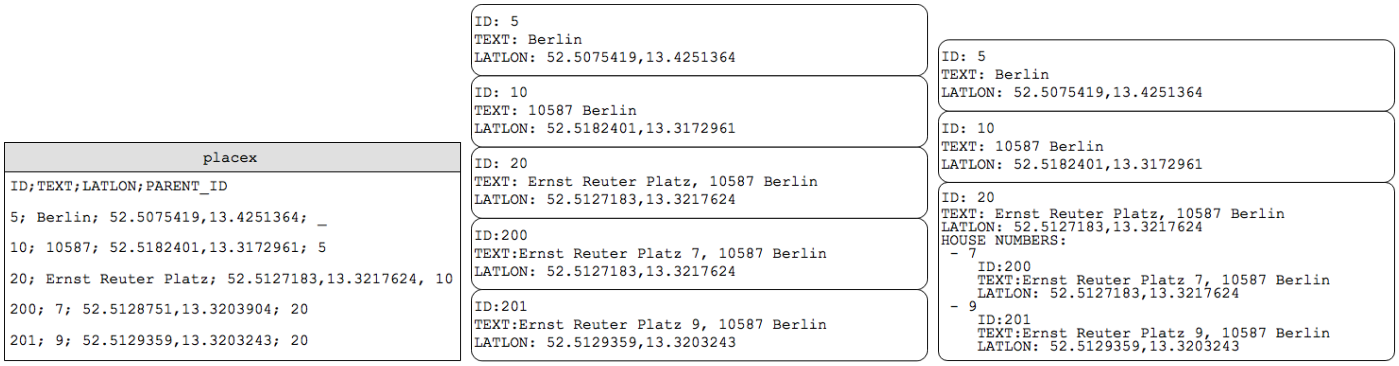

Figure 1. Schemes of indices in Nominatim (left, simplified), Elasticsearch (middle), and EAHN (right)

weaknesses of every system. All in all, BM25f strives to bring the document to the top of the result lists that, according to term hits and their respective weights, matches to a given query the best. Also, [25] shows, that address schemes often contradict each other, e.g., by some schemes assuming the house number before the street name while other schemes assume it behind the street name, which results in ambiguous parsing of address elements. Therefore, it is fair to assume that the Elasticsearch based solutions are more solid geocoding services than Nominatim.

II. Experiment

Three geocoding systems: Nominatim, Elasticsearch, and *Elasticsearch with Aggregated House Numbers* (EAHN) have been set up with the same data in their indexes. First, Nominatim has been set up with OpenStreetMap data for Europe. This long-running process is fully automated and results in a PostgreSQL data base that holds the data and the indexes for Nominatim. The entries served by Nominatim are stored in a single table, which is the output of processing OpenStreetMap data and assembling address elements spread across entities. Besides a name, e.g., the name of a street or the house number, rows of this table store their areal extent in PostGIS geometries, their IDs, types, ranks, and some additional meta-data. Additionally, each row references the rows containing the data of the next-higher administrative area: A row with a house number references the row with the street name the house number belongs to, which in turn references the row with the city district that street segment is in, and so forth. This allows entries to be normalized, not storing the names of higher level administrative areas that make up a full address. The full address hierarchy is assembled by a stored procedure, which is called by Nominatim for each result at query time.

This same stored procedure and a PostGIS procedure to derive the centroid latitude and longitude of PostGIS geometries were applied to every row of Nominatim's result table generating documents to be indexed in Elasticsearch. Documents derived this way contain the full address of an entry (which, in the case of higher level administrative areas, may not contain the most specific address parts), the entries' ID in the Nominatim data base, and WGS84 latitude and longitude coordinates. An Elasticsearch instance was set up next to Nominatim and populated with the documents extracted.

One drawback of indexing every address in Elasticsearch is the vast amount of denormalized data: Street names, districts, cities, postal codes, etc. do not differ for many addresses, but are all stored and indexed as parts of separate documents for every house number address. This requires additional space and makes the process of picking the right document harder: The BM25f scores of different documents are less spread apart if the documents scored contain redundant data. EAHN tries to approach this problem, by only creating separate documents for separate street segments. Thus, in EAHN BM25f is only used to find the right street segment. All house numbers of that street segment are stored within that document. They are, however, not indexed in Elastcisearch and extracted using string matching in a second step.

To set up EAHN with the same data, the same data base table was used. First, as for documents in Elasticsearch, for every row the full address for its entry as well as the entries' ID and WGS84 coordinates have been assembled. Entries representing a house number address were separated from non house number entries using a flag Nominatim stores in the data base as part of each row's meta data. House number entries were grouped by the ID of their parent, which, according to

Nominatim's data base structure, would be the street segment the house numbers would be in. Non house number entries were used to generate documents to be indexed in EAHN. In a final step, documents of entries referenced by house number entries were extended to contain a mapping from each house number to the respective house number address, its ID and its WGS84 coordinates. These documents with aggregated house numbers as well as all documents that were not referenced by a house number entry were indexed in an Elasticsearch instance. Note that by default Elasticsearch would index every part of the document. To achieve the reduction in index size, the instance used for EAHN was explicitly configured to index the address text of street segments only, ignoring house number mappings and their addresses.

Figure 1 presents the simplified schemes of the three systems. While Nominatim uses a normalized table with entries referencing their parents, Elasticsearch contains a denormalized document per entry. EAHN aggregates documents representing house numbers in their parent document, containing a denormalized document for each entry that is not a house number. Looking at the schemas in Figure 1 makes it obvious that Elasticsearch is a geocoding system already: A response to a search request contains addresses with their WGS84 coordinates that match to a given query. That is not the case for EAHN: It requires an additional step between its Elasticsearch instance and a client that unwraps documents representing house numbers if appropriate. This layer has been implemented as follows:

1) Forward every query to Elasticsearch, request it to highlight those parts of the query that have matched to parts of the document.
2) For every candidate in the returned list, for every query token that has not been highlighted as matched, if the token is a key in the house number mapping, return the text, the ID, and the WGS84 coordinates the key maps to.
3) Fall back to returning the text, the ID, and the WGS84 coordinates of the first candidate on the result list if no suitable house number result has been found.

For EAHN, ideally, a street segment would be the entire part of a street referenced by the same address, i.e., the segment has the same city, the same district, the same postal code, etc. However, OpenStreetMaps defines street segments on a finer-grained level: Multiple equally addressed street segments exist next to each other. These segments represent different sides or multiple chunks of a lengthy street, or parts of a street that vary in other attributes as speed limits. Some house numbers would be attached to one street segment, while others would be attached to another. For this article, the street segments, as they are defined in OpenStreetMaps were used. Therefore, it is not sufficient for EAHN to only look at the house numbers attached to the first candidate returned by Elasticsearch. Without house numbers being part of the scoring scheme BM25f cannot differentiate between otherwise equal documents in a meaningful way. Thus, it is inevitable to look at all equally named street segments. For experiments in this article, EAHN was configured to query Elastcsearch for 250 candidates to look at – a number small enough to be processed and big enough to cover almost all of the cases with multiple equally named street segments. On the downside, requesting many candidates for a street that does not have many segments exposes the risk to match a house number on a segment of a different street than the one the query referred to. To account for that, a house number match has only be returned as stated in (2) if the number of query tokens matching the candidate was equal or greater than the number of matched tokens in the first candidate. Thereby, every matched token was counted only once, even if the indexed document contained that token multiple times. That was required to deal with documents that stated address elements multiple times. Such documents appeared as artifacts of Nominatim collecting address elements from incorrectly tagged OpenStreetMap entities. For example, entities for city districts often list the postal codes contained in those districts repeatedly. Another caveat of EAHN are house number ranges. Since there is no standardized way to specify them, various separators are used between the leading and the tailing number in OpenStreetMap data as well as in the data sets used for experiments. Because house number ranges were not normalized by EAHN, some times, a requested house number range would not be retrieved, even though it was available in the data. Similarly, letters added to house numbers were not normalized and not found if the index had a different capitalization or whitespace variant of the same house number. Generally, because house numbers have not been indexed in the documents, they did not contribute to the BM25f score of the document leaving a greater chance of false documents appearing on top of the result list.

Given all these downsides of EAHN, its main feature only becomes obvious after it has been populated with data: Because instead of every house number address

TABLE I. Index sizes

system	row or document count	index size on disk
Nominatim	61 198 211	(366 GB)
Elasticsearch	61 198 211	15.7 GB
EAHN	33 678 783	6.2 GB

being a separate document house numbers are aggregated into the document of their street segment, the number of indexed documents is drastically reduced. Also, house numbers themselves are not indexed in EAHN at all. Therefore, while all of the data is still available the index size is much smaller too. Table I shows the index sizes of the three systems benchmarked in this article after they have been populated with OpenStreetMap data for Europe. Nominatim collected and indexed 61.2M addresses from OpenStreetMap data. Every single entry in Nominatim is indexed as a separate document in Elasticsearch. Aggregating the house numbers on the same street segment into one document nearly cuts the number of documents to be indexed in EAHN in half. Because the house number tokens are not indexed any more, and because overall there are less documents to reference, the index size of EAHN shrinks by more than 50%. Note that the index size of Nominatim is not comparable with the sizes of the other two indexes. The given size of the PostgreSQL data base after Nominatim's processing of OpenStreetMap data also includes the source entities as well as preliminary results of the processing. To operate Nominatim as a geocoder, many of these tables are not required. Also, Nominatim contained and indexed the full set of address translations, which have not been exported or indexed. This feature has not been evaluated on Elastcisearch and EAHN, though it is fair to assume that address translations can be handled in the same way as addresses in their local languages.

As a final difference between Nominatim and the other two systems, abbreviations are to be mentioned. Nominatim implements a defined set of operations to, e.g., normalize STREET to ST in both the indexed data and a served query. These operations have not been ported to Elasticsearch or EAHN. Because Nominatim returned addresses are not abbreviated, the documents indexed in Elasticsearch and EAHN did not contain any abbreviations implicitly. To mimic the feature, test sets were manually processed to spell out every abbreviated address element.

At the end of the set up process three geocoding

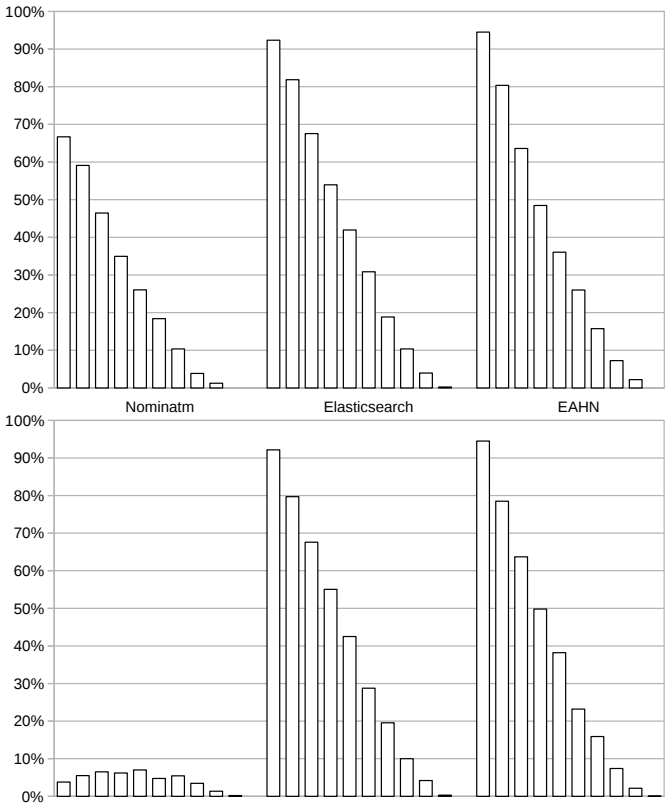

Figure 2. Percentages of successfully geocoded addresses with increasingly less address tokens for **addresses extracted from Nominatim** with address tokens in order (top) and shuffled (bottom)

systems with the same addresses indexed were running next to each other: Nominatim with precomputed rank scores as well as BM25f based Elasticsearch and EAHN. To compare these three systems three benchmarks have been designed to gradually increase the complexity of requests issued. The benchmarks allow to observe how the percentage of successfully served requests decreases for the three systems. Three data sets have been generated for the three systems:

1) A data set of 2000 addresses randomly sampled from the Nominatim data base.
2) A data set of 1500 addresses of pharmacies in big German cities.
3) A data set of 2000 addresses randomly sampled out of addresses sourced from a German on-line portal for real estate.

In accordance to the set up, set (1) of addresses extracted from Nominatim (and, therefore, indexed in each system) contain samples from various European countries. The addresses were extracted with their IDs so that right or wrong responses from the geocoding systems

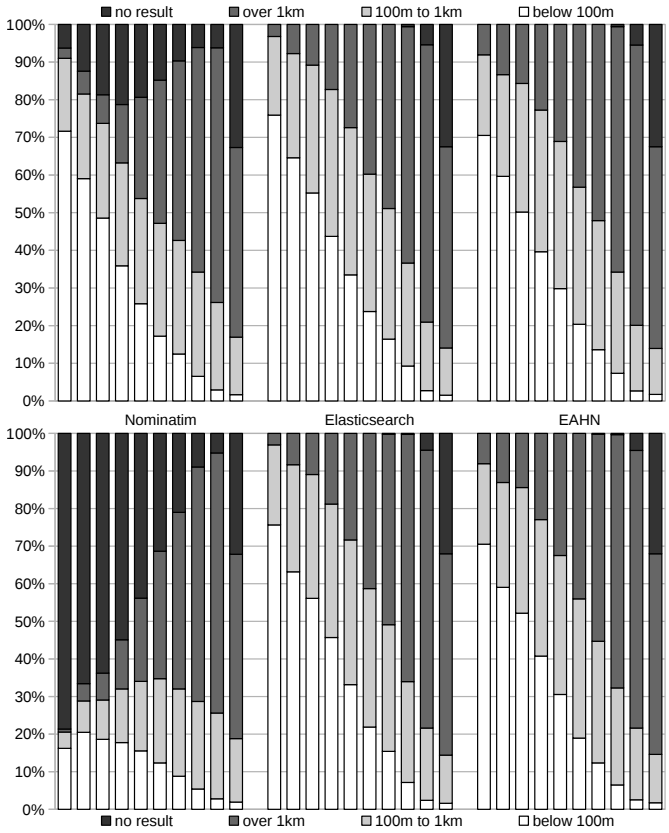

Figure 3. Percentages of buckets of geocoding responses to requests with **formally correct postal addresses** and increasingly less address tokens with address tokens in their original order (top) and shuffled (bottom)

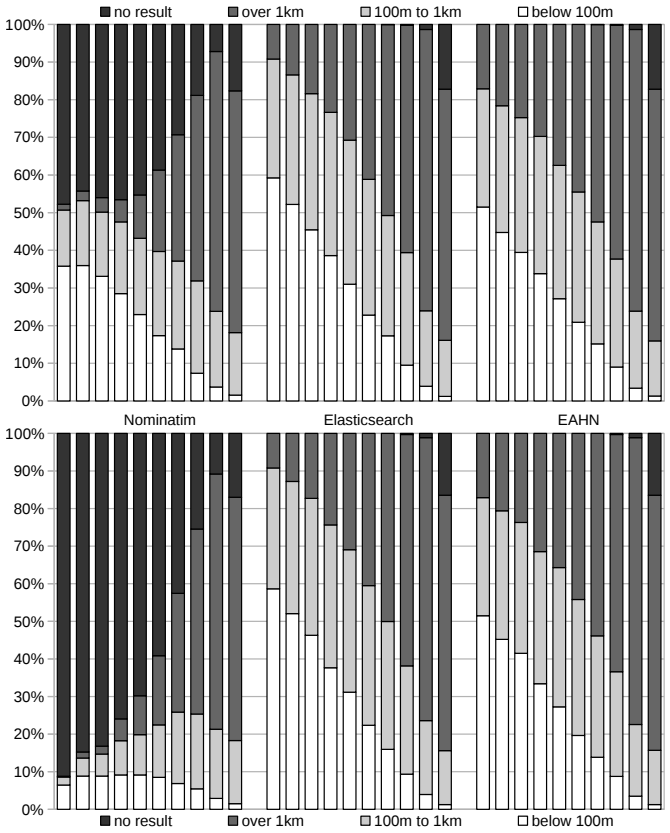

Figure 4. Percentages of buckets of geocoding responses to requests with **user input addresses** and increasingly less address tokens with address tokens in their original order (top) and shuffled (bottom)

were identified by simple ID comparison. The addresses were extracted exactly as Nominatim presents them to clients. This presentation is usually verbose: It names additional address elements as district or county names that are not usually part of a postal address. In contrast to that, set (2) contains formally correct postal addresses of pharmacies. Note that for some of the pharmacies OpenStreetMap data (and, therefore, the three geocoding systems) contain the addresses as well as the pharmacies them selves as separate entries. This does not interfere with the approach used to evaluate if a request with this address has been successfull or not: The addresses have been geocoded with the Google's geocoding system as reference first. Because Google's geocoding system uses different data than Nominatim, Elasticsearch, and EAHN, results were not expected have equal coordinates. Instead results were grouped into buckets depending on the distance to the reference. The buckets *within 100m, 100m-1000m, further than 1000m*, and *no result* have been used. Because of these distance based buckets, it did not matter if a geocoding system returned the result for an address or the pharmacy itself: The positions of both entries are close enough so that their distance to the reference position would up in the same bucket for either of the two cases. For a general purpose geocoder (and, therefore, in this article) only results from the first bucket are close enough to be considered as successfully served ones. Set (3) contains yet again a different type of addresses: The on-line portal is asking agents to input addresses of the real estate they are offering. Thus, as actual real estate is offered, all the addresses are most likely to exist and be comprehensible for human beings. More importantly, however, these addresses are not necessarily formally correct, as addresses in (2). Instead, addresses are spelled out as humans refer to them when communicating with each other. The on-line portal is also asking their users to specify geocoordinates of addresses by clicking on a map. As these WGS84 coordinates are part of the data set, relying on a reference geocoding system is not necessary. As for pharmacies, results of the real estate addresses were grouped into the same four buckets, depending on their distance to the user

specified coordinates. Again, only results in the *within 100m* bucket can be considered correct.

For every data set multiple experiments have been conducted: First, full addresses as they are in the data sets have been issued as geocoding requests to each system. Next, in steps of 10% more and more randomly picked address tokens have been removed from requests, up until requests contained only 10% of all address tokens. The consequence of dropping tokens from the query are less precise and more ambiguous requests. This behavior mimicked users that query for incomplete addresses. Finally, the whole iteration of increasingly ambiguous queries has been repeated with randomly shuffled address tokens. This behavior mimicked users that do not adhere to a formal address standard. In total for each of the 5500 addresses 10 queries with different token counts were issued with tokens in their original and in shuffled order to each of the three systems under test. Note that all queries were composed from house number addresses. A benchmark examining a production geocoding system would incorporate a contingent of queries for named areas that is proportional to the number of queries for named areas the system has to serve. In this article, however, no productive system specifies the portion of queries for named addresses. Also, it is fair to assume that geocoding house number addresses is a more complex scenario.

III. Results

Measurement results for all three systems and all three data sets are presented in Figures 2, 3, and 4. Every chart has three blocks of bars – one for the reference geocoding system Nominatim, one for the document store Elasticsearch, and one for Elasticsearch with aggregated house numbers. The leftmost bar of each block shows the success rate for issuing geocoding requests with all address tokens. Every next bar shows results for requests with additional 10% address tokens dropped. Each block consists of exactly ten bars with the rightmost bar showing the success rate for geocoding requests with only 10% of address tokens. In each figure the upper chart is showing the results of stating requests with address tokens in their original order, while the bottom chart is showing the results of geocoding queries with shuffled address tokens. Note that for ordered and shuffled queries, different query tokens were dropped at random: A query for an address with address tokens in their original order would therefore contain different tokens than a shuffled query for the same address. This statistical noise explains, e.g., why in Figure 2 EAHN seems to perform slightly better with 60% of address tokens shuffled, rather than in their original order. An insight in some specific numbers is given by Table II. It lists the percentages of successful requests for all systems and data sets with requests containing 100% and 50% of address tokens. Every value pair gives the success rate for requests with address tokens in their original order first, followed by the success rate for requests with shuffled address tokens.

Results for geocoding addresses extracted from the index are presented in Figure 2. The upper chart shows that Elasticsearch and EAHN are outperforming Nominatim with full addresses already. With less and less address tokens in the queries and, therefore, with less distinctive queries, the performance of all the three systems decreases linearly, keeping Nominatim below the two BM25f based systems constantly. Another picture is shown by the lower chart: While Elasticsearch and EAHN are almost not affected by shuffling of address tokens at all, Nominatim's success rate drops to below 5% for full addresses. With increasingly less tokens, Elasticsearch and EAHN behave as if the address tokens were in order. In contrast, Nominatim seems to become better at first, reaching its maximum at requests composed with 60% of address tokens. Even at this maximum Nominatim still stays below its own performance with address tokens in order. Interestingly, EAHN performs better than Elasticsearch for full addresses only. For addresses with missing tokens, Elasticsearch takes the lead.

Figure 3 shows response buckets of requests with formally correct postal addresses of pharmacies. The bars for the various buckets are shaded differently and stacked adding up to 100% of requests. The general trend on this figure is similar to that shown on Figure 2: Nominatim does not handle shuffled address tokens well; Elasticsearch and EAHN constantly outperform Nominatim. A closer look, however, reveals that Nominatim is more successful in dealing with formally correct postal addresses than with queries for addresses extracted from the index. Table II highlights that for the two BM25f based systems the opposite is true: On indexed addresses these systems perform best. Another thing to note on Figure 3 is that Nominatim often returns no results, while the other two systems return some responses for most of the queries. That behavior can be good or bad, depending on the use case scenario.

Results of the experiment using user input addresses in Figure 4 confirm the previous observations. Again,

TABLE II. Select rates of successfully served geocoding requests (ordered - shuffled)

	indexed addresses	**formally correct postal addresses**	**user input addresses**
100% address tokens in query			
Nominatim	66.7% - 3.8%	71.6% - 16.2%	35.8% - 6.4%
Elasticsearch	92.3% - 92.1%	**75.9% - 75.6%**	**59.2% - 58.6%**
EHN	**94.5% - 94.5%**	70.5% - 70.5%	51.5% - 51.5%
50% address tokens in query			
Nominatim	18.4% - 4.7%	17.2% - 12.3%	17.35% - 8.5%
Elasticsearch	**30.8% - 28.7%**	**23.7% - 21.9%**	**22.8% - 22.3%**
EHN	26.0% - 23.2%	20.3% - 18.9%	20.9% - 19.6%

Nominatim is constantly outperformed by the other two systems. Again, only Nominatim is impacted negatively if address tokens are shuffled. Clearly, the user input data set was the hardest to geocode: All three systems are least successful with geocoding these data.

IV. Conclusion

The experiments show that Nominatim requires queries to adhere to the implemented addressing schemes to function well: It performs best on the data set of formally correct postal addresses. However, if query tokens are in arbitrary order, Nominatim is likely to not find the right result. This is a strong limitation, given that there is not one universal, but many different and some times contradicting address schemes in use. Because there are less ways to shuffle less address tokens, Nominatims performance at first increases when less tokens are used to issue a query. Addresses with missing tokens are less distinctive, which is why Nominatim does not reach the same performance as with query tokens left in their order. For the same reason the performance of the BM25f based systems decreases linearly proportional to the number of address tokens dropped. The experiments prove that BM25f is an approach suitable to select proper geocoding results. It is independent of the token order and generally performs better than precomputing global ranks. The vast superiority of the two BM25f systems on indexed addresses is somewhat artificial: Querying a system that matches documents to queries using queries generated from documents indexed is very likely to produce a high success rate. The only reason why Elasticsearch does not achieve full 100% hits is data: As discussed, some entries are repeated in the OpenStreetMaps data set, thus a duplicate to the right result has a different ID and is therefore not recognized as a correct response. This is also the reason why EAHN outperforms Elasticsearch for full addresses: By design EAHN looks into multiple candidates for a house number match, which makes it more likely to derive the correct result. Obviously, the actual performance of a geocoding system highly depends on the actual queries it faces. When developing geocoding systems, however, Elasticsearch makes a good base line to compete with.

EAHN has proven to be a valuable approach to offer geocoding services too. As expected, it yields slightly less accurate results than Elasticsearch, because less parts of the address are indexed. Still, the observed impact has shown to be a tolerable cost, especially taking into account the gains EAHN offers: Smaller index size and less indexed documents promote responsiveness and scalability of a system. Also, a smaller index is easier to update. Future work could assess how to incorporate house numbers into the indexing and scoring methods to close the performance gap to Elasticsearch. Also, aggregating equally named street segments would lead to further reduction of the index size. Ideally, an approach can be found that utilizes the hierarchical structure of addresses on other levels as well. For example, by aggregating districts of a city in the same way house numbers of a street have been aggregated for EAHN would allow to store index data in a more compact and efficient way. Finally, to further improve EAHN, normalization logic for house number ranges could be developed. For example, house number ranges to be indexed could be unfolded at indexing time mapping every house number in the range to the entire range it is part of. At query time it would suffice to use only the starting or only the ending house number of a range to retrieve the entire range address.

REFERENCES

[1] K. Clemens, "Geocoding with openstreetmap data," GEOProcessing 2015, 2015, pp. 1–2.

[2] "OpenStreetMap," http://wiki.openstreetmap.org, Nov. 2015.

[3] National Imagery and Mapping Agency, "Department of Defense, World Geodetic System 1984, Its Definition and Relationships with Local Geodetic Systems," in Technical Report 8350.2 Third Edition, 2004.

[4] M. Goetz and A. Zipf, "Openstreetmap in 3D–detailed insights on the current situation in Germany," in Proceedings of the AGILE 2012 International Conference on Geographic Information Science, Avignon, France, 2012.

[5] D. Goldberg, J. Wilson, and C. Knoblock, "From Text to Geographic Coordinates: The Current State of Geocoding," URISA Journal, vol. 19, no. 1, 2007, pp. 33–46.

[6] "Google Geocoding API," https://developers.google.com/maps/, Nov. 2015.

[7] "Nomatim," http://wiki.openstreetmap.org/wiki/Nominatim, Nov. 2015.

[8] "GeoNames," http://www.geonames.org, Nov. 2015.

[9] Open Geospatial Consortium, "Gazetteer Service – Application Profile of the Web Feature Service Best Practice," Feb. 2012.

[10] D. Yang, L. Bilaver, O. Hayes, and R. Goerge, "Improving geocoding practices: evaluation of geocoding tools," Journal of Medical Systems, vol. 28, no. 4, 2004, pp. 361–370.

[11] L. Can, Z. Qian, M. Xiaofeng, and L. Wenyin, "Postal address detection from web documents," in Web Information Retrieval and Integration, 2005. WIRI'05. Proceedings. International Workshop on Challenges in. IEEE, 2005, pp. 40–45.

[12] "PostGIS," http://postgis.net/, Nov. 2015.

[13] "PostgreSQL," http://www.postgresql.org/, Nov. 2015.

[14] "Apache HTTP Server," http://httpd.apache.org/, Nov. 2015.

[15] G. Salton and C.-S. Yang, "On the specification of term values in automatic indexing," Journal of documentation, vol. 29, no. 4, 1973, pp. 351–372.

[16] G. Salton, C.-S. Yang, and C. T. Yu, "A theory of term importance in automatic text analysis," Journal of the American society for Information Science, vol. 26, no. 1, 1975, pp. 33–44.

[17] S. Robertson, H. Zaragoza, and M. Taylor, "Simple BM25 extension to multiple weighted fields," in Proceedings of the thirteenth ACM international conference on Information and knowledge management. ACM, 2004, pp. 42–49.

[18] "Lucene," http://lucene.apache.org/, Nov. 2015.

[19] "Elasticsearch," https://www.elastic.co/, Nov. 2015.

[20] A. Rodriguez, "Restful web services: The basics," IBM developerWorks, 2008.

[21] "elasticsearch-osmosis-plugin," https://github.com/ncolomer/elasticsearch-osmosis-plugin/wiki/Quick-start, Nov. 2015.

[22] "Osmosis," http://wiki.openstreetmap.org/wiki/Osmosis, Nov. 2015.

[23] "Pelias," https://github.com/pelias/, Nov. 2015.

[24] "gazetteer," https://github.com/kiselev-dv/gazetteer/, Nov. 2015.

[25] K. Clemens, "Automated processing of postal addresses," in GEOProcessing 2013, The Fifth International Conference on Advanced Geographic Information Systems, Applications, and Services, 2013, pp. 155–160.