Geospatial Partitioning of Open Transit Data

Harm Delva*, Julián Andrés Rojas, Pieter-Jan Vandenberghe, Pieter Colpaert, and Ruben Verborgh

IDLab, Department of Electronics and Information Systems, Ghent University – imec
{harm.delva,pieter.colpaert}@ugent.be
https://idlab.technology/

Abstract. One of the guiding principles of open data is that anyone can use the raw data for any purpose. Public transit operators often publish their open data as a single data dump, but developers with limited computational resources may not be able to process all this data. Existing work has already focused on fragmenting the data by departure time, so that data consumers can be more selective in the data they process. However, each fragment still contains data from the entire operator's service area. We build upon this idea by fragmenting geospatially as well as by departure time. Our method is robust to changes in the original data, such as the deletion or the addition of stops, which is crucial in scenarios where data publishers do not control the data itself. In this paper we explore popular clustering methods such as k-means and METIS, alongside two simple domain-specific methods of our own. We compare the effectiveness of each for the use case of client-side route planning, focusing on the ease of use of the data and the cacheability of the data fragments. Our results show that simply clustering stops by their proximity to 8 transport hubs yields the most promising results: queries are 2.4 times faster and download 4 times less data. More than anything though, our results show that the difference between clustering methods is small, and that engineers can safely choose practical and simple solutions. We expect that this insight also holds true for publishing other geospatial data such as road networks, sensor data, or points of interest.

Keywords: Linked Open Data · Mobility · Maintainability · Web API engineering.

1 Introduction

People who rely on wheelchair-accessible public transportation have very specific information needs when they are looking to buy a house. Real estate websites can include this information in their item listings, but only if they can find and access relevant datasets. Fortunately, many public transit operators publish their offering as open data, often using de facto standards such as the General Transit Feed Specification[1] (GTFS) or official standards such as Network Timetable

[1] https://developers.google.com/transit/gtfs

Exchange[2] (NeTEx). However, these standards result in large data dumps: the combined GTFS feed of the public transit companies that operate in the Brussels area (SNCB, STIB, De Lijn, and Tec) is already over 1 GB of raw data. Searching for *"GTFS memory issues"* on the Web shows that many people have learned the hard way that this is more data than their personal laptops, Raspberry Pis, or entry-level VPSs can handle.

A popular use case for open transit data is route planning. The ideal route depends on many factors such as ticket prices, transfer times, walking distances, reliability, and arrival times. The value ascribed to each of these factors is ultimately subjective, and will likely change over time due to external factors such as the weather. However, contemporary route planning services offer little in terms of personalization because they sacrifice flexibility to provide better query time performance [3, 13, 15]. For example, an algorithm that relies on precomputed shortest paths is ill-suited to generate scenic routes. Alternatively, the route planning can be done directly on the client, and this has the benefit that more flexible algorithms become viable because users can only saturate their own CPUs. Client-side applications come with their challenges though, and ingesting the data is particularly difficult in this case. The European Commission reported that in 2019 the average price for 2 GB of mobile data within the EU28 was still €10 [10], which means that that client-side route planners have to be conservative in which data they download.

These examples show that the way data is published can restrict how the data can be used. What may be feasible for a corporation may not be feasible for a regular person, even though the Open Definition[3] defines open data as data that can be *"freely used, modified, and shared by anyone for any purpose"*. Our goal is thus clear: we want to improve the way open transit data is published, so that more applications become more viable for more people.

2 Related work

We identify three domains of related work which we discuss in the following subsections: (i) research in the field of Linked Data and the Semantic Web has focused on making data reusable and interoperable, (ii) existing mobility data specifications and what sets them apart, and (iii) how are public transit networks currently being partitioned and for what purpose. To close off this section we also briefly discuss Voronoi Diagrams, as our proposed method makes extensive use of them.

Note that throughout this paper we use three similar, but different, terms: *cluster*, *partition*, and *fragment*. In essence, clusters are partitions; *clustering* merges similar items while *partitioning* starts from the set of all items – so that clustering individual public transit stops partitions the network itself. A planar space, such as the world, can also be partitioned, in which case each partition

[2] http://netex-cen.eu/
[3] https://opendefinition.org/

can be called a *region* instead. Fragments on the other hand come from the field of Linked Data and refer to Linked Data Fragments, i.e. resources on the Web.

2.1 Linked Data Fragments

To facilitate interoperability with other datasets, Open Data is often *Linked Data* as well. Tim Berners-Lee outlined the four principles of Linked Data [8]: 1) use URIs as names for things, 2) use HTTP URIs so that people can look up those names, 3) when someone looks up a URI, provide useful information using standards such as RDF, and 4) include links to other URIs so that they can discover more things. In the conceptual framework of *Linked Data Fragments* [18], this is just one interface to access Linked Data. You could also publish the data as one large data dump, or provide a querying API on top of the data. What all these interfaces have in common is that they expose a *fragment* of the entire dataset, so they can all be considered Linked Data Fragments. Data dumps and query APIs are the two extremes on the Linked Data Fragments axis [18]. This axis illustrates the trade-offs between different methods of publishing Linked Data on the Web. Data dumps put the data processing burden on the client's side, but allow the most flexibility for clients. Query APIs on the other hand put the processing burden on the server side but always restrict, in some way, the way the data can be used.

2.2 Mobility Data

The *General Transit Feed Specification (GTFS)* is, at the time of writing, the de facto standard for publishing public transit schedules. A single feed is a combination of 6 to 13 CSV files, compressed into a single ZIP archive. Its core data elements are stops, routes, trips, and stop times. Stops are places where vehicles pick up or drop off riders, routes are two or more stops that form a public transit line, trips correspond to a physical vehicle that follows a route during a specific time period, and stop times indicate when a trip passes by a stop. This data is not only useful for route planning applications, other applications include embedding timetables in mobile applications, data visualization; accessibility analysis, and planning analysis [1].

The Linked Connections specification [9] defines a way to publish transit data that falls somewhere in the middle of the Linked Data Fragments axis. Connections are defined as vehicles going from one stop to another without an intermediate halt. These connections are then ordered by departure time, fragmented into documents, and are then published over HTTP. Clients can use the semantics embedded in each fragment to solve their own queries. This, combined with the fact that each fragment is easily cacheable, make Linked Connections servers more scalable than full-fledged route planning services.

2.3 Partitioning Public Transit Networks

Researchers in the field of route planning have noted that methods based on partitioning have been successful for accelerating queries on road networks, but that

adapting those methods to public transit networks is harder than expected [6, 7]. One of the main differences is that road networks are, for the most part, topological networks. Public transit networks on the other hand are also inherently time-dependent. On top of that, it is not even clear *what* exactly needs to be partitioned as different algorithms can require wildly different data models [11].

The Scalable Transfer Patterns [5] algorithm aims to greatly reduce preprocessing times of the original Transfer Patterns [4] algorithm. The authors compared 4 different techniques to partition stops into clusters of roughly equal size: 1) *k-means* using the stops' geographical locations, 2) a merge-based clustering with a utility function that punishes big partitions and rewards pairs of partitions with high edge weights between them, 3) a general-purpose graph clustering algorithm called *METIS* [17], and 4) a road partitioning method called *PUNCH* [12]. They found that k-means, despite being completely oblivious to the network structure outperformed both METIS and PUNCH while their own merge-based approach performed the best of all. HypRAPTOR [11] is another route planning algorithm that uses METIS to partition the network graph, but which uses clusters of trips instead of stops.

2.4 Voronoi Diagrams

Voronoi diagrams are one of the fundamental data structures in computational geometry [2]. Although they can be applied to any metric space, we only consider Euclidean spaces in this paper for the sake of simplicity. Given a set of seed points in a Euclidean space, a Voronoi diagram partitions that space into regions so that each region contains exactly one seed point, and every point in a region is closer to that region's seed point than to any other region's. Formally this means that for a given Euclidean space X with distance function d, and a set of seed points $P \subset X$, each point $p_i \in P$ yields a corresponding Voronoi region $R_i \subseteq X$ where

$$R_i = \{x \in X \mid d(x, p_i) \leq d(x, p_j) \text{ for all } i \neq j\}$$

3 Method

The Linked Connections publishing scheme enables applications to access data for a specific point in time, but each data fragment still contains data from the entire transit operator's service area. Figure 1 shows that some regions served by the Flemish public transit operator, De Lijn, are more popular than others, implying that it makes sense to partition by location as well. Existing work has shown that partitioning public transit networks can improve query times of route planning services, so we investigate if similar improvements can be obtained for the publishing of raw data.

However, first we should consider what is necessary to make publishing fragmented data viable in the real world. We make a distinction between data owners and data publishers, with a clear distinction between their responsibilities.

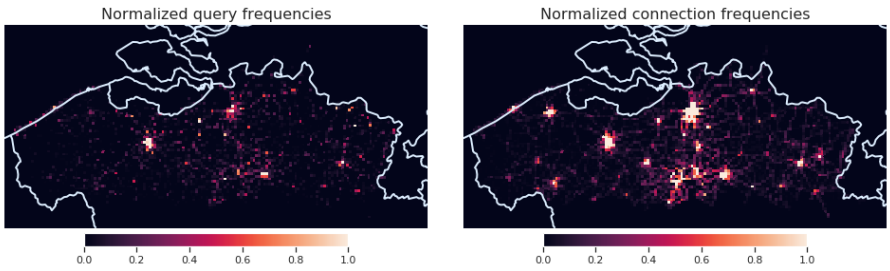

Fig. 1. Visualized on the left are the departure and destination locations, based on one week of query logs from the Flemish public transit operator De Lijn. Visualized on the right are the locations of all connections in their network during the same time period. Note that there are many places with a considerable amount of connections that are in low demand.

A *data owner* focuses on maintaining the data quality, while a *data publisher* focuses on making the data accessible. Both roles come with their own challenges, and as such it is not uncommon for data owners to outsource the data publishing to third parties. This means that data publishers may not have control over the actual data – they have to adapt when the data changes. For example, public transit operators routinely add and remove temporary stops due to maintenance works, and these changes have to be reflected in the published data with as little friction as possible.

3.1 Rationale

Existing work has focused on clustering stops, or trips, into discrete sets of objects. If a data publisher were to follow this approach, they would have to explicitly assign a label to every new stop the data owner adds. Failing to do so would cause them to publish incomplete data, as unlabeled stops will not be in any published cluster. This labeling of new stops is relatively easy for clustering algorithms such as k-means, but for algorithms such as METIS [17] this involves recomputing the entire clustering.

Rather than searching for an algorithm that supports updates, we propose to publish the clusters in a robust way by partitioning the physical world instead of creating discrete sets of stops. The resulting partitions are published as separate resources, allowing any agent to infer to which cluster every stop belongs. In other words, data publishers do not have to explicitly label every stop themselves – the data speaks for itself. This benefits both the data publishers and consumers: the maintenance effort required by the publisher is lower, and data consumers have access to complete and factual data.

3.2 Data

Guided by the insights provided by Figure 1, we will focus on the Flemish public transit network for the remainder of this paper. To provide some context: Flanders is a small region within Europe, but with 487 inhabitants/km^2 in 2019, it is also one of the most densely populated [14]. The public transit network is also dense; at the time of writing there are 35,791 stops spread out over 13,522 km^2 for a density of 2.6 stops/km^2. There are roughly 1 million connections on a regular weekday, and the corresponding Linked Connections data results in over 10 million RDF triples per day. We use data from the first whole week of December 2019 as the input data for the methods discussed in this section.

3.3 Clustering

We start by adapting two clustering methods that are often used to partition transit networks: k-means and METIS. However, both methods disregard one important feature of transit networks; k-means does not consider network connectivity and METIS does not consider physical locations. This leads us to propose an additional method, called *Hub*, which clusters stops by their proximity to important transportation hubs. As others have shown good results from hierarchical methods, we also consider a merge-based adaption of Hub, appropriately named *Merged*. The remainder of this subsection discusses how each method is used to generate a geospatial partitioning.

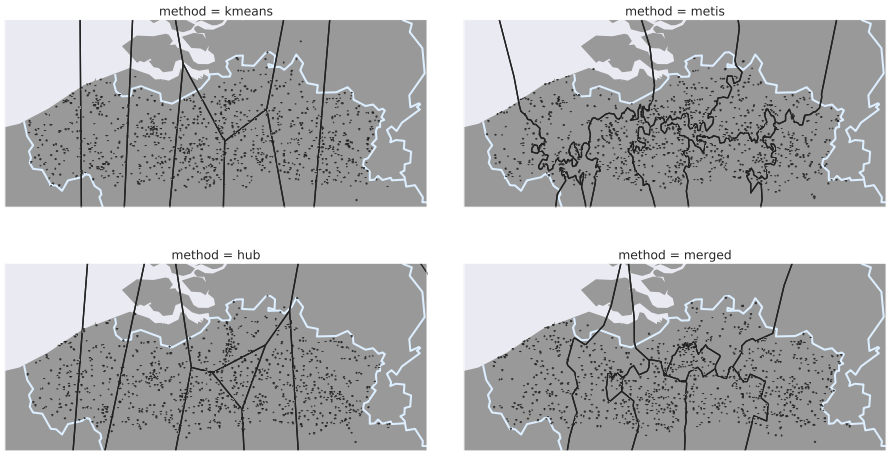

Fig. 2. The 8 partitions each evaluated method creates. Note that the two methods on the top row create regions of roughly equal sizes, while the approaches at the bottom create regions of varying sizes. The approaches in the left column create regions with simple shapes, while the ones on the right create irregular shapes.

k-means Despite its simplicity, existing work has found k-means to be competitive with more complex methods [5], so we consider it among the state of the art for this particular use-case. As the name implies, this algorithm distributes a given set of points in exactly k clusters, where every point belongs to cluster with the nearest cluster mean. Iterative heuristics exist to compute this clustering, and we used the implementation from scikit.learn[4] with default parameters and using the stops' WGS84 coordinates as input.

To obtain a spatial partitioning from this, we create a Voronoi diagram using the cluster means as seed points. Because the Voronoi cells of two adjacent points on the convex hull share an infinitely long edge, we add some extra padding points that represent the bounding box of the operator's service area – and then discard all infinite edges.

METIS METIS is another algorithm that is used to partition public transit networks [5, 11], so we consider it to be among the state of the art as well. Since it is a graph clustering algorithm, we must represent the public transit network as a graph. We follow the conventional approach of creating a vertex for every stop, and connecting them with an edge if they are connected through a single connection. Every edge is assigned a weight that corresponds to how many connections connect those stops. We used a Python wrapper[5] of the reference implementation to compute the clustering, using the `contig` option to force contiguous partitions.

The METIS algorithm only sees the network as a connectivity graph though – it does not know anything about the physical location of the stops. This means that even though it creates contiguous partitions, those partitions are not contiguous in the physical world. We obtain a clean spatial partitioning using an additional post-processing step that 1) creates the Voronoi diagram of all stops, 2) merges all Voronoi cells that belong to the same cluster, and 3) merge isolated areas into the surrounding cluster.

Hub Hub is the first of our own methods that aims to incorporate both the geospatial and the graph-like nature of public transit networks. It iteratively selects the stops based on which trips pass through it. In the first iteration it selects the stop with the most unique trips, in the subsequent iterations it selects the stop with the most unique trips that the previous stop(s) do not have. After k iterations it contains the k most important hubs, which lead us to name this method *Hub*. These selected stops are then used as seed points to create a Voronoi diagram. To illustrate the simplicity of this approach, Listing 1 contains all the necessary code to implement this, up until the creation of the Voronoi diagram.

[4] https://scikit-learn.org/0.20/modules/clustering.html#k-means
[5] https://metis.readthedocs.io/en/latest/

```python
def hub(k):
    done_trips = set()
    selected_stops = []
    for _ in range(k):
        best_stop = None
        best_stop_score = 0
        for stop, trips in stop_to_trips.items():
            stop_score = len(set(trips) - set(done_trips))
            if stop_score > best_stop_score:
                best_stop = stop
                best_stop_score = stop_score
        selected_stops.append(best_stop)
        done_trips.update(stop_to_trips[best_stop])
    return selected_stops
```

Listing 1. The *Hub* method can be implemented in just 14 lines of Python code.

Merged Instead of stopping the Hub algorithm after k iterations we can also let it terminate, and then use the Jaccard similarity coefficient to merge the two most similar adjacent Voronoi regions until only k remain. As there is a finite amount of trips, this algorithm has a clear termination condition: it stops when all trips are covered by one of the selected stops. This makes the process more complex, but existing work has shown good results using hierarchical clustering techniques [5]. We have named this approach *Merged*, for obvious reasons.

3.4 Hypermedia Controls

As discussed at the beginning of subsection 3.1, we want our published data to be easy to maintain. Our idea is to publish the partitioning itself, so that clients have all the information they need to decide to which cluster every stop belongs. We have already discussed how to obtain the partitionings, now we discuss how to publish them.

The partitions are published on the Web as stand-alone resources using the Hydra[6] and GeoSPARQL[7] vocabularies. Hydra is used to describe a partitioning as a collection of regions, and the `wktLiteral` datatype from the GeoSPARQL vocabulary is used to describe individual regions. GeoJSON is another common way to define geometries, but since GeoJSON polygons are incompatible with JSON-LD we chose to use the simpler string representation: WKT. Listing 2 contains a JSON-LD snippet of a single partition resource.

These partition resources are then used to fragment Linked Connections data. This two-step approach allows for reusing existing partitions, such as administrative regions. A modified Linked Connections server can ingest a given par-

[6] http://www.hydra-cg.com/spec/latest/core/
[7] http://www.opengis.net/doc/IS/geosparql/1.0

```
{
  "@id": "https://example.org/clusters/hub_4",
  "hydra:member": [
    {
      "@id": "https://example.org/clusters/hub_4/1",
      "geo:asWKT": "POLYGON ((4.170761972221639 50.7079439...
    }, ...
  ], ...
}
```

Listing 2. JSON-LD representation of a partitioning. Note that both the partitioning and the individual regions are separate resources, allowing other datasets to refer to them.

titioning, and fragment the data accordingly. The server creates one view per region, and then creates an index of all generated views using the tree[8] vocabulary. This vocabulary is used to link every view to the geospatial area it covers. Listing 3 contains a JSON-LD snippet of such an index.

```
{
  "@id": "https://example.org/connections",
  "@type": "tree:Node",
  "tree:relation": [
    {
      "@type": "tree:GeospatiallyContainsRelation",
      "shacl:path": "geo:contains",
      "tree:node": "https://example.org/connections?cluster=
    ↪ https%3A//example.org/clusters/hub_4/1",
      "tree:path":[
        "lc:departureStop",
        "geo:asWKT"
      ],
      "tree:qualifiedValue": {
        "tree:value": {
          "@id": "https://example.org/clusters/hub_4/1"
        },
        "tree:path":"geo:asWKT"
      }
    }, ...
  ], ...
}
```

Listing 3. JSON-LD representation of a view index. The `tree:node` property points to a data page from the original Linked Connections specification. The `tree:qualifiedProperty` property defines which geospatial area that page covers by referring to an existing published geospatial partition.

[8] https://github.com/TREEcg/specification

4 Evaluation

In the introduction we declared our intent to make more applications viable by improving the way we publish data. We gave client-side route planning as an example of a use case that needs to be conservative in the amounts of data they download, so we focus on this application to evaluate our data.

We have adapted an existing library for client-side route planning that uses Linked Connection data, so that it can interpret our hypermedia controls. This library uses the *earliest arrival time* variant of the Connection Scan Algorithm. This algorithm, similar to Dijkstra's algorithm, builds a list of which stops are reachable and how long it takes to reach them. A client that knows the location of each stop can also infer which clusters are reachable, so our adapted route planner simply fetches data for all reachable clusters – slowly growing its list of data sources. We focus on the use-case of client-side route planning because this a relatively demanding application.

As mentioned in section 3, we use 1 week of Linked Connection as input for the clustering algorithms. We then use each method to create 4, 8, 16, and 32 clusters. A redis-backed server creates an ordered list of all connections within every generated region, and exposes these using the hypermedia controls defined in the subsection 3.4. The same server also hosts a version of the data with one cluster that contains all the data, i.e. without any geospatial partitioning. Altogether we test 17 (4 partitionings for each of the 4 methods, and the baseline) different partitionings, and each data fragment contains 20 minutes of data.

We make extensive use of letter-value plots [16] because our results have a long tail, which causes visualizations such as box plots to label many results as outliers. These plots show the median value as a black line, and then show the 75%, 87.5%, ... quantiles as separate boxes, making it easy to compare these statistics.

4.1 Efficiency

As a proxy for how easy the geospatially fragmented data is to use, we measure how much work a client needs to do to solve a query. Specifically, the time it takes for the same client to solve the query with a given partitioning, as well as how much data was downloaded. We compare those values to those of the baseline; the unpartitioned data.

5,000 queries were randomly selected from a query log that was given to us by the transit operator itself. All these queries were received on the same day, but throughout the day. We eliminate as many variables as possible to isolate the impact of the partitioning; the client and server run on two separate machines on the same local network, a constant 20ms of latency is added per response, and the client only processes one query at a time.

Figure 3 shows that having just a few clusters already significantly improves the query performance, but that adding more clusters has diminishing returns, because even without the overhead of ingesting unnecessary data the client still has to compute the actual route. The METIS results are somewhat surprising;

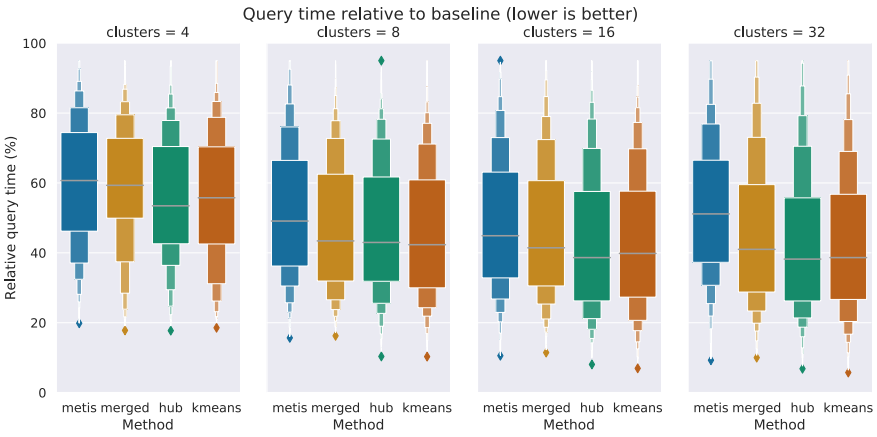

Fig. 3. The median query time with just 4 clusters is already 58% that of the original query times, and using 8 clusters further improves this to 45%. Note the diminishing returns as more clusters are added, using 16 or 32 clusters reduces the relative query times to 41% and 42%, respectively. The Hub and k-means methods yield very similar results, while METIS performs significantly worse.

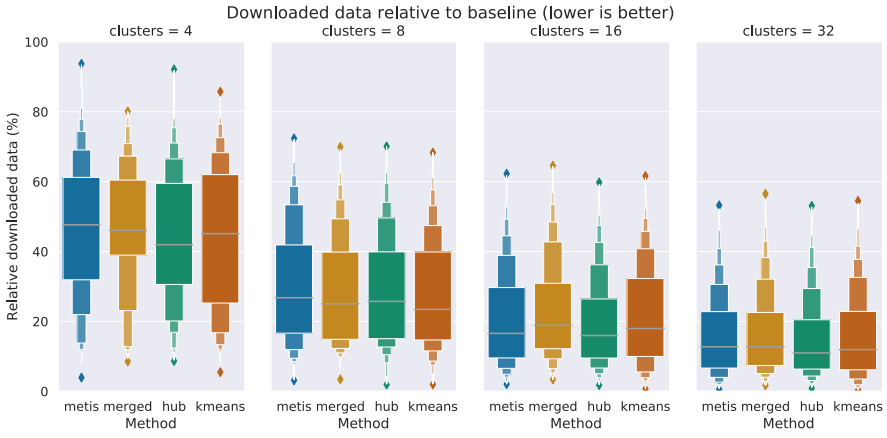

Fig. 4. Using just 4 clusters is enough to reduce the amount of downloaded data to 45% of the original amount of downloaded data, and adding more clusters consistently improves this metric. Although all methods seem competitive in this metric, the Hub method has a consistently low median and 75% percentile.

they are slightly worse across the board, and even become worse when going from 16 to 32 clusters. As Figure 2 shows, the clusters from METIS are more complex than those from other methods, which makes them harder to interpret for a client. Figure 4 shows that the amount of downloaded data does keep decreasing by adding more clusters – theoretically we can avoid all unnecessary data by creating a cluster per stop.

4.2 Cacheability

Another important feature of Linked Connections is the cache effectiveness of the fragments, which gives a Linked Connections server its scalability. As we are making the data more fine-grained, we have to measure the impact this has on the cache effectiveness. Unfortunately, the query logs we use do not contain any form of user ID, which makes it hard to simulate a real-world scenario where there are client-side and server-side caches. Instead, we measure how fast a cache warms up in every configuration, and what the hit rate of a warm cache is. These two metrics give an indication of how cacheable the partitioned data is, and how this compares to the cacheability of the original data.

While running the benchmarks for the usability metrics, we also record which resources are fetched. We then replay these requests, running them through a simulated LRU cache to measure the hit rates. To measure the hit rates on a warm cache we first run all requests through a cache, and then create 1,000 samples of 500 requests to measure the overall hit rate of each sample. The hit rates on a cold cache are obtained by doing the same starting from a cold cache, and by varying the amount of requests per sample. We set the cache size to 20 MB, and each partitioning results in roughly 70 MB of gzipped data, so we expect to see many cache evictions.

Figure 5 and Figure 6 show that partitioned data can improve the cache hit rate, but that caches take longer to warm up. The cache effectiveness when using 8 clusters surpasses that of the baseline at around 350 requests, regardless of the clustering method. The average query downloads 9 resources at this granularity, so that the cache effectiveness is better than the baseline's if the data is used to answer more than 39 queries per day.

5 Discussion

In the introduction we stated that our goal is to improve the way public transit data is published to make more applications viable. We found related work in the field of route planning, where the data is fragmented to improve query-time performance. However, our findings show that results from this field do not easily translate to publishing data on the Web, because, as stated in section 3, we want the processed data to stay in sync with the raw data. We resolve this by moving some of the clustering logic to the client, which in return can then avoid downloading and parsing a lot of irrelevant data. Knowing which clusters to publish is just as important as knowing how to publish them though, so we also

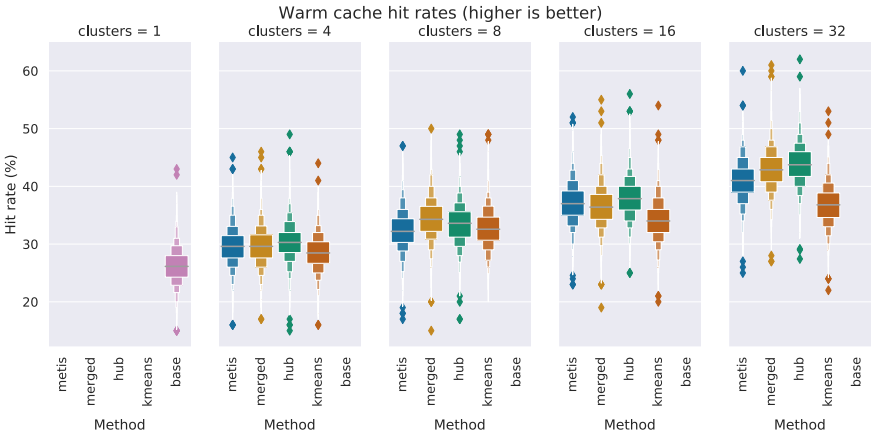

Fig. 5. Less valuable cache space is wasted on irrelevant data by using a fine-grained partitioning. The median hit rate on a warm cache using the unpartitioned data is 26%, the highest hit rate, 44%, is obtained using the Hub method with 32 clusters. The k-means method scores noticeable worse than the other methods.

compare different clustering algorithms – and how they affect the performance of a client-side route planner.

The number of clusters has a noticeable impact on all evaluated metrics. Even a small amount of clusters can make a client-side route planner twice as fast. More clusters do not necessarily lead to better results though, as we quickly see diminishing returns in terms of query times. The amount of downloaded data does keep decreasing, but at the cost of cacheability. Interestingly, even when starting from a cold cache the cacheability of a small amount of clusters is on par with the cacheability of the original data.

METIS and k-means yield good results in the amount of downloaded data metric, but both struggle in other tests. Clusters from METIS have a complex shape because it does not consider the stops' locations, making it harder for clients to interpret them. As a result, the query times using METIS data are consistently worse than those using other methods. A similar pattern presents itself for the merge-based method, which is also noticeably worse in the query time metric – more so than in the downloaded data metric. The k-means method on the other hand shows great results in both the query time and downloaded data metrics, but the resulting data fragments are harder to cache. Our own Hub method is the only method that performs well across all metrics. This method combines the geospatial and the graph-like features of public transit networks. The merge-based approach does this all well, but is burdened by more complex cluster shapes.

More than anything though, our results show that the difference between clustering methods is small, and that engineers can safely choose practical and simple solutions. Our own method domain-specific method yields the best re-

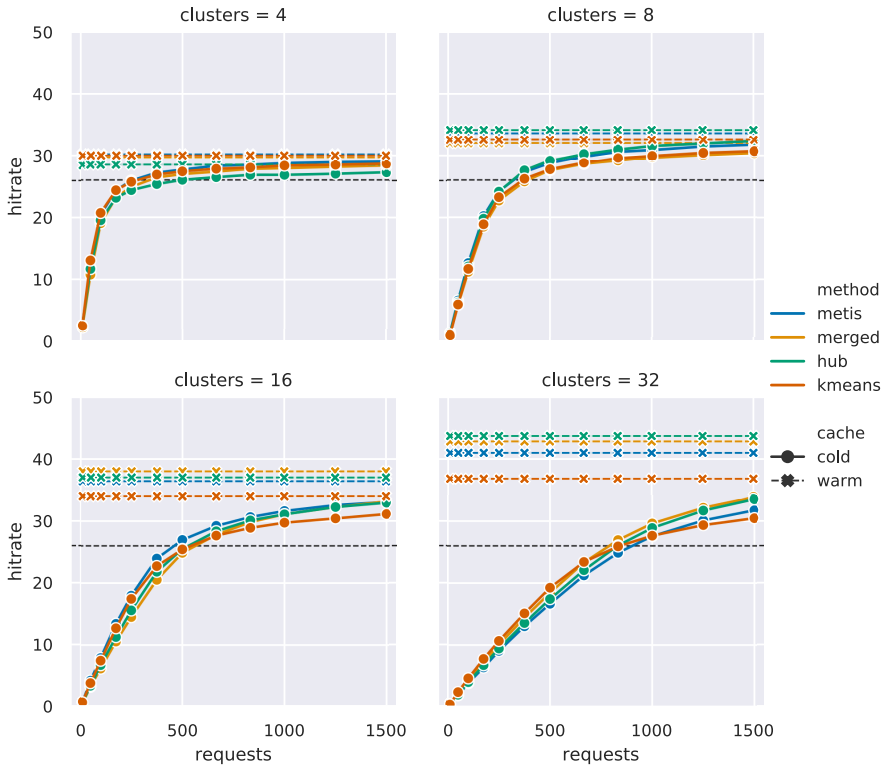

Fig. 6. Line plots of the median cache hit rates per configuration, showing that caches take longer to warm up with a fine-grained partitioning. However, any method with 4 or 8 clusters matches the hit rate of the original data on a warm cache (26%) after 350 requests.

sults, but it is so simple we do not consider this specific method to be our main contribution; it's the realization that simple methods can, and do, outperform complex methods. And it is this insight that's useful for web engineers – one does not *have* to be a domain expert to publish quality data.

6 Conclusion

In this paper we investigated what data publishers can do to make their open transit data easier to use. Based on research from the field of route planning, we explored the idea of geospatially partitioning public transit networks. We evaluated 4 different clustering methods for the use-case of client-side route planning: k-means, METIS, and two domain-specific methods of our own. The partitions were obtained using Voronoi diagrams, and were then published with the appropriate hypermedia controls that clients can use to discover clusters of public transit stops.

Our goal was to make open transit data more useful, so that more people can use it in more applications. We focused on the use case of client-side route planning, which have to be conservative in which data they download as mobile data is still expensive. And in that regard, we succeeded. A simple clustering algorithm and 8 clusters is all it takes to download 4 times less data, and to answer queries 2.4 times faster. Preliminary results show that the cacheability, and thus the scalability, of this approach is on par with the existing Linked Connections publishing scheme.

More than anything though, we have found that the difference between clustering methods is small, and that engineers can safely go for simple solutions – any geospatial fragmentation is better than no fragmentation at all. Future work can investigate if this translates to the publishing of other geospatial data such as road networks, sensor data, or points of interest. We postulate that it does, simply because the world is not uniformly populated – data from densely populated regions will be in higher demand. Additionally, our approach should be tested in the real world, comparing it to both route planning services and existing Linked Connections servers.

References

1. Antrim, A., Barbeau, S.J., et al.: The many uses of gtfs data–opening the door to transit and multimodal applications. Location-Aware Information Systems Laboratory at the University of South Florida **4** (2013)
2. Aurenhammer, F.: Voronoi diagrams—a survey of a fundamental geometric data structure. ACM Computing Surveys (CSUR) **23**(3), 345–405 (1991)
3. Bast, H., Carlsson, E., Eigenwillig, A., Geisberger, R., Harrelson, C., Raychev, V., Viger, F.: Fast routing in very large public transportation networks using transfer patterns. In: European Symposium on Algorithms. pp. 290–301. Springer (2010)
4. Bast, H., Carlsson, E., Eigenwillig, A., Geisberger, R., Harrelson, C., Raychev, V., Viger, F.: Fast routing in very large public transportation networks using transfer patterns. In: European Symposium on Algorithms. pp. 290–301. Springer (2010)

5. Bast, H., Hertel, M., Storandt, S.: Scalable transfer patterns. In: 2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX). pp. 15–29. SIAM (2016)
6. Bauer, R., Delling, D., Wagner, D.: Experimental study of speed up techniques for timetable information systems. Networks **57**(1), 38–52 (2011)
7. Berger, A., Delling, D., Gebhardt, A., Müller-Hannemann, M.: Accelerating time-dependent multi-criteria timetable information is harder than expected. In: 9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'09). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2009)
8. Berners-Lee, T.: Linked data-design issues. http://www.w3.org/DesignIssues/LinkedData.html (2006)
9. Colpaert, P., Llaves, A., Verborgh, R., Corcho, O., Mannens, E., Van de Walle, R.: Intermodal public transit routing using liked connections. In: International Semantic Web Conference: Posters and Demos. pp. 1–5 (2015)
10. Commission, E.: Mobile broadband prices in europe 2019. http://www.w3.org/DesignIssues/LinkedData.html (2019)
11. Delling, D., Dibbelt, J., Pajor, T., Zündorf, T.: Faster transit routing by hyper partitioning. In: 17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2017)
12. Delling, D., Goldberg, A.V., Razenshteyn, I., Werneck, R.F.: Graph partitioning with natural cuts. In: 2011 IEEE International Parallel & Distributed Processing Symposium. pp. 1135–1146. IEEE (2011)
13. Dibbelt, J., Pajor, T., Strasser, B., Wagner, D.: Connection scan algorithm (2017)
14. Flanders: Population: size and growth. https://www.statistiekvlaanderen.be/en/population-size-and-growth-0 (2019)
15. Geisberger, R., Sanders, P., Schultes, D., Delling, D.: Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In: International Workshop on Experimental and Efficient Algorithms. pp. 319–333. Springer (2008)
16. Hofmann, H., Kafadar, K., Wickham, H.: Letter-value plots: Boxplots for large data. Tech. rep., had.co.nz (2011)
17. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on scientific Computing **20**(1), 359–392 (1998)
18. Verborgh, R., Hartig, O., De Meester, B., Haesendonck, G., De Vocht, L., Vander Sande, M., Cyganiak, R., Colpaert, P., Mannens, E., Van de Walle, R.: Querying datasets on the web with high availability. In: International Semantic Web Conference. pp. 180–196. Springer (2014)