International Cartographic Association
Association Cartographique Internationale

An Efficient Data Structure Providing Maps of the Frequency of Public Transit Service Within User-Specified Time Windows

Annika Bonerath [a*], Yu Dong [b], Jan-Henrik Haunert [a]

[a] *University of Bonn, surname@igg.uni-bonn.de*
[b] *University of Bonn*

* Corresponding Author

Abstract: To understand the quality of a public transportation network, map-based visualizations are the first choice. Since the network quality depends on the transportation schedule, the visualization interface should incorporate time. In this work, we provide such an interface where users can filter the data for time windows. When a user queries for a certain time window, we display all network segments that are traversed by at least θ means of public transportation, e.g., buses in the time window on the map. This gives insights into the frequency of the public transit service.
Since such transportation networks can contain a high number of network segments, testing each network segment at the moment when the user specifies the time-window query is not fast enough. We investigate an approach to provide the query result in real time. Our contribution is a data structure that answers such arbitrary time-window queries from the user interface. The data structure is based on a tree structure which is augmented with further information. To evaluate our data structure, we perform experiments on real-world data. With our data structure, we answer time-window queries in at most 25 milliseconds whereas testing each network segment on-demand takes at least 60 milliseconds for the data set that has 102599 road segments.

Keywords: spatio-temporal data, interactive visualization, data structure, time window

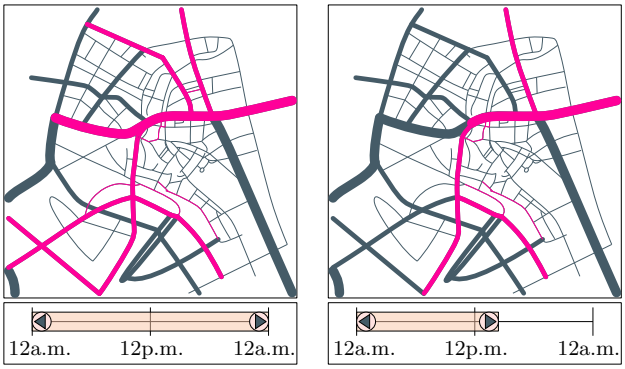
1. Introduction

For many people, the public transport network plays a major role in their daily lives. Therefore, careful planning is of utmost importance. This means that decision-makers must be able to easily inform themselves about the spatial and temporal patterns of connectivity meaning the frequency of public transit service. The most common visualizations of public transport data are (1) the schedule tables for transportation lines at each station and (2) a map with the routes of all transportation lines. Although both visualizations work well for certain applications, e.g., informing users of their next route, they do not provide a high-level overview of the spatio-temporal patterns of the entire network. We focus this work on bus networks but it can be easily extended to networks that also contain trams, etc.

In this work, we look at an interactive visualization approach that enables users to explore the spatio-temporal patterns of the public transportation network.

1. **spatial patterns:** Which parts of the city are served, i.e. , which roads are traversed by buses?
2. **temporal patterns:** In which time windows are which parts of the city served?

Driven by these two criteria, we introduce the concept of frequently served road segments, i.e., we say a road segment is *frequently served* for a time window if it is traversed by at least θ buses in the time window. We consider θ to be a given threshold for the network. When a user explores the public transportation network, we display all frequently served road segments for the queried time window. In a more enhanced variant of the visualization, one can either use more than one threshold or encode the number of buses that traversed the road for all roads that have been traversed by more than θ buses.



(a) largest time window (b) small time window

Figure 1. The public transportation network and the user interface for two time windows. The road segments that are frequently served are drawn in pink. Between (a) and (b) the users have slid the right boundary of the time window.

We note that there are other concepts for the visualization of patterns of a public transportation network, e.g., displaying all road segments at which at least θ buses stop in the queried time window instead of looking at travers-

Advances in Cartography and GIScience of the International Cartographic Association, 4, 1, 2023.
31st International Cartographic Conference (ICC 2023), 13–18 August 2023, Cape Town, South Africa. This contribution underwent double-blind peer review based on the full paper. https://doi.org/10.5194/ica-adv-4-1-2023 | © Author(s) 2023. CC BY 4.0 License

ing buses. We want to emphasize that our data structure supports also such a concept.

In the following, we introduce our user interface and how the user can interact with it. Our interface is based on a *map* and a *timeline*. The timeline consists of a time axis (e.g., showing hours of a day) as well as a time window represented by a rectangle. An example of such a time window is Monday 9 a.m. till Monday 11 a.m. To enable user interaction, we use time sliders as introduced by Andrienko and Andrienko (1999). The user can interact by (i) panning: continuous translation of the time window; (ii) left or right boundary slide: continuous change of the left or right boundary of the time window. We implement the interaction by three buttons; see Figure 1. With the time slider, the user can filter the data for a time window. For each time window, we display the frequently served roads on the map, i.e., the road segments that were traversed by at least θ buses in the time window; see Figure 1.

Using this visualization, the user can easily and quickly assess parts of a city that are not that frequently served by buses. By interacting with the time window the users can explore temporal and spatial patterns. This can enable them to get a quick overview of the network. Experts and decision-makers can use this visualization in combination with background knowledge (e.g. population density) to make better-informed decisions.

In the following, we give a formal definition of our problem.

Frequently Served Roads for Time-Window Query.
input:

1. *The road segments R of a public transportation network where each road segment is annotated with the time stamps of bus traversals.*
2. *A pre-defined threshold θ that specifies the minimum number of bus traversals such that a road segment is frequently served.*
3. *A user-defined time-window query $Q = [t, t']$.*

output:

The set of all road segments R_Q that were traversed by at least θ buses in the time window Q.

In order to allow a pleasant user experience, when exploring the frequently served road segments of a city, a user should be able to receive the visualization in real-time for a time-window query. Motivated by movies with a frame rate of 24 images per second, we want to achieve a response time of roughly 40 milliseconds.

A simple approach for computing the visualization for a time-window query is an on-demand compuation. Here, we go through all road segments and we count the number k of bus traversals in the time window Q for each road segment. Then, we report all road segments where k is larger than the given threshold. All reported road segments are displayed. Especially for large data sets, such an on-demand implementation can lead to high computation times. In our experiments with a network of 102599 road segments, such a computation approach takes at least

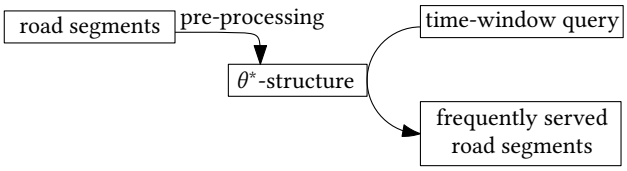


Figure 2. Pre-processing the public transportation network into our data structure.

60 milliseconds for each query. Hence, the computation time achieved by this on-demand approach is not sufficient and interaction might be very unpleasant.

Our contribution is a data structure that enables a real-time interaction with the time slider. Note that the visualization remains the same as for the on-demand approach that we described before but we speed-up the computation. We call this data structure θ^*-structure. The general idea is that we pre-process the public transportation network consisting of the road segments annotated with the bus traversal times into the θ^*-structure; see Figure 2. This θ^*-structure encodes the answers for any time-window query. This means that whenever a user queries for a particular time window, the answer can directly be obtained from the θ^*-structure. The θ^*-structure is based on a binary tree where each node of the tree is enhanced with additional information. In particular, we associate each leaf node of the tree with a road segment. We investigate which road segment is assigned to which node. Note that our θ^*-structure is a generalization of the θ-structure presented by Bonerath et al. (2020). With this work, we resolve the restriction of the θ-structure that we need an underlying grid and the visualization needs to be a grid-based density map. We note that our data structure can be easily extended to support multiple thresholds just like the θ-structure.

For the evaluation of our data structure, we perform experiments with real-world data from the public transportation network of Bonn, Germany. The benefit of our data structure is the speed-up in query time with respect to the on-demand approach. We show that the query time is smaller than 25 milliseconds and hence, we consider it to be appropriate for real-time exploration. The data structure brings an additional construction time as costs. It is performed once in a pre-processing step and in our experiments we can build the data structure in up to 32 minutes for a network with 102599 road segments. We deem to be acceptable with regard to the speed-up of the query time.

We structure our paper as follows. At first, we discuss related work in Section 2. Secondly, we present a formal model for the θ^*-structure in Section 3. Then, in Section 4 we present our experimental evaluation on real-world data. Finally, in Section 5 and Section 6, we give an overview on more general application scenarios of the θ^*-structure and an outlook for future work.

2. Related Work

In the following, we discuss several research fields and works from these fields that are closely related to our θ^*-structure.

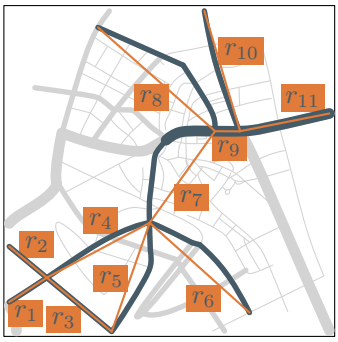


Figure 3. Thinned out (dark grey) and simplified (orange) road network.

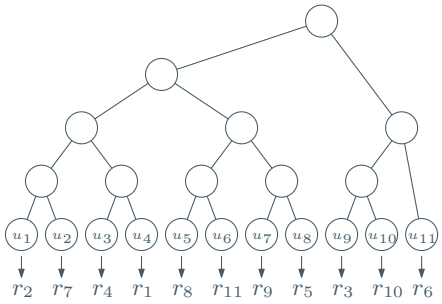


Figure 4. θ^*-structure for a random mapping between the leaf nodes and the road segments.

Spatio-Temporal Data Visualization In cartography, the visualization of spatio-temporal data is an important branch (Williams et al. (2013)). But also other research fields, e.g., in visual analytics (Keim et al. (2008), Andrienko et al. (2010), Sun et al. (2013)), and geovisualization (Kraak (2003)) contributed to this topic. In particular, the visualization of public transportation data is an important scenario. Amongst others, Forsch et al. (2021) considered isochrones for travel times given a fixed starting point, Zeng et al. (2014) developed a visualization system that combines different views, and Scheepens et al. (2011) looked at density maps for trajectories that do not necessarily lie on a network.

Dynamic Query Interfaces Kapler and Wright (2005) introduced *dynamic query interfaces* as an interaction technique for the exploration of various kinds of data. Often, it is implemented for temporal filtering and, then, it is called *time slider* (Kapler and Wright (2005), Ahlberg and Shneiderman (2003), Hochheiser and Shneiderman (2004), Robinson et al. (2017), Andrienko and Andrienko (1999), Burigat and Chittaro (2005)). The user can define a query using sliders and the application gives a real-time response. Due to the required real-time response, the problem of efficiency (Tanin et al. (1996)) arises. Overall, dynamic query interfaces are well studied for user performance and they show a good performance in comparison to a paper printout, a text search, and form-fill-in interfaces (Kapler and Wright (2005), Ahlberg et al. (1992)).

Time-Windowed Data Structures Data structures that can be queried efficiently for time windows were introduced by (Bannister et al. (2013)) and are called *time-windowed data structures*. They exist for reporting properties of *relational event graphs* (Bannister et al. (2013), Chanchary and Maheshwari (2019), Chanchary et al. (2019)), for problems from computational geometry (Bannister et al. (2014), Bokal et al. (2015), Chan and Pratt (2015, 2016), Chanchary et al. (2018)), and for event visualizations based on α-shapes (Bonerath et al. (2019)) and density maps (Bonerath et al. (2020)). From a technical point of view, our data structure is based on the θ-structure. Hence, we will discuss this structure in detail and highlight the differences to our approach.

θ-Structure For the θ-structure, Bonerath et al. (2020) have the scenario that they are given a set of events as input where each event is a pair of a point in space and a time stamp. An example of such event data is bird observations. Here, the aim is to visualize grid-based density maps for time-window queries. To achieve a real-time response the θ-structure is introduced. It corresponds to a quad-tree where the leaf node level is induced by the spatial grid. Further, they introduce for each node of the quad-tree a time function that improves the query time.

In contrast to the θ-structure, our θ^*-structure is a binary tree. We have adopted the time function for each node. The main contrast to the θ-structure is that there is no straightforward mapping between the road segments and the leaf nodes of the θ^*-structure. We will discuss several versions and (experimentally) evaluate these.

3. θ^*-Structure

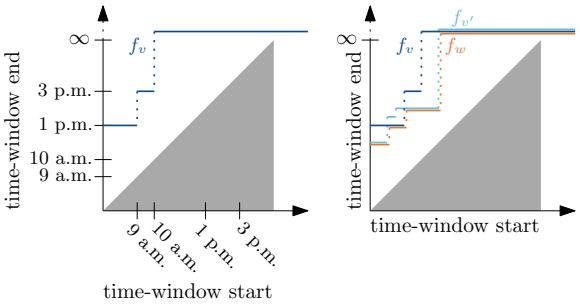
In the following, we introduce our θ^*-structure which allows efficient querying for frequently-served road segments in time windows. Let R be the set of n road segments where each road segment $r \in R$ is annotated with a set of time stamps of the bus traversal times. Let $\mathscr{B}$ be a balanced binary tree with n leaf nodes. In our data structure, we map each road segment from R to one leaf node of $\mathscr{B}$. Further, we introduce the *time function* for each node of $\mathscr{B}$ that speeds up the query time.

We structure the description of the θ^*-structure as follows. At first, we introduce the time function that we compute for each node of $\mathscr{B}$ under the assumption that we are given a mapping between the road segments from R to the leaf nodes from $\mathscr{B}$. Secondly, we show how to query the θ^*-structure for a particular time window. And thirdly, we discuss different options for mapping the road segments to the leaf nodes.

3.1 Build a θ^*-Structure

In this section, we assume that we are given a mapping between the leaf nodes $U = \{u_1, \ldots, u_n\}$ of the binary tree $\mathscr{B}$ and the road segments R. In a very simple variant, one could use a random order of the road segments. In order to simplify the notation and without loss of generality, we assume that road segment r_i is mapped to leaf node u_i for $1 \leq i \leq n$.

The *time-function* $f_i(t)$ of a leaf node u_i reports for any time-window start time the earliest time-window end time



(a) time function of Equation 1 (b) time function of Equation 2

Figure 5. Time function. (a) The time function for one leaf node of the θ^*-structure. (b) The time functions of two sibling nodes f_v and $f_{v'}$ and their parent node f_w.

such that the road r_i is frequently served, i.e., it is traversed by θ buses. Take a road segment with four-time stamps $T_i = \{9\text{ a.m.}, 10\text{ a.m.}, 1\text{ p.m.}, 3\text{ p.m.}\}$ and a threshold $\theta = 3$ as example. Equation 1 and Figure 5a give the time functions for v.

$$f_i(t) = \begin{cases} 1\text{ p.m.} & \text{if } t \leq 9\text{ a.m.} \\ 3\text{ p.m.} & \text{if } 9\text{ a.m.} < t \leq 10\text{ a.m.} \\ \infty & \text{if } 10\text{ a.m.} < t. \end{cases} \quad (1)$$

Now, we introduce the time-function f_w of an internal node w of $\mathscr{B}$. Let $u_i, \ldots, u_j$ be the leaf nodes that are descendants of w. We want the time function f_w to report for any time-window start time the earliest time-window end time such that at least one road segment $r \in \{r_i, \ldots, r_j\}$ is frequently served.

We compute the time functions from the bottom of $\mathscr{B}$ to the top. For the leaf nodes, we can compute the time function as described above. For an internal node w, let v and v' be its children for which we have already computed the time-functions f_v and $f_{v'}$, respectively. Then, the time-function f_w is the lower boundary of the time-functions f_v and $f_{v'}$; see Figure 5b.

$$f_w(t) = \min\{f_v(t), f_{v'}(t)\} \quad (2)$$

Overall, we can compute the θ^*-structure in $O(nm \log n)$ time where n is the number of road segments and m is the overall number of time stamps. The θ^*-structure has asymptotic size of $O(nm \log n)$.

3.2 Query a θ^*-Structure

In the following, we describe how we query a θ^*-structure for a time window $[t_{\text{start}}, t_{\text{end}}]$. For a node v of the binary tree, we say that v *reports* `true` *for* Q if $f_v(t_{\text{start}}) \leq t_{\text{end}}$ and otherwise v *reports* `false` *for* Q. For v being an internal node reporting `true` means that there is at least one descendant leaf node of v that reports `true`. For v being a leaf node reporting `true` means that the road segment that corresponds to v is frequently served for Q. The query approach is straightforward: we start at the root node of the tree and traverse it downwards as long as we either reach a leaf node or a node reports `false`; see Algorithm 1.

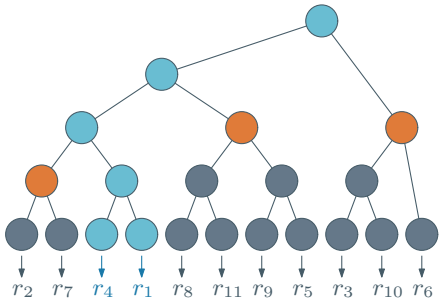


Figure 6. The query procedure for a time-window query Q. Blue nodes report `true`, orange nodes report `false`, and grey nodes are not queried. The road segments of the two blue leaf nodes correspond to the frequently served road segments for Q.

Algorithm 1: QUERY

Data: θ^*-structure $\mathscr{B}$, time-window query $[t_{\text{start}}, t_{\text{end}}]$
Result: frequently served road segments R_Q for $[t_{\text{start}}, t_{\text{end}}]$

```
let v be the root node of B;
if v reports true for Q then
    if v is leaf node of B then
        add corresponding road segment of v to R_Q;
    else
        let B' and B'' be the subtrees of B that are
          created when we remove v from B;
        add result of QUERY(B', [t_start, t_end]) to R_Q;
        add result of QUERY(B'', [t_start, t_end]) to R_Q;
    end
end
```

The running time of checking whether a node reports `true` or `false` for a time-window query, depends on the time needed to evaluate $f_v(t_{\text{start}})$. Since f_v is a staircase function, we can store it as an array F_v where each row corresponds to the leftmost point of a horizontal segment of f_v. Equation 3 is the array for the time function given in Equation 1.

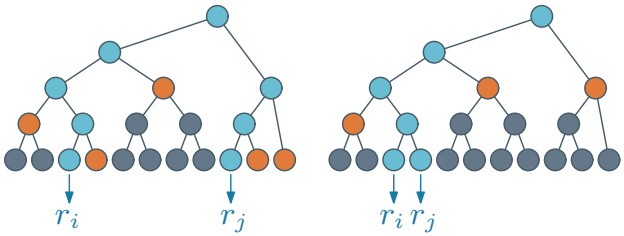
$$F_v = \begin{bmatrix} -\infty & 1\text{ p.m.} \\ 9\text{ a.m.} & 3\text{ p.m.} \\ 10\text{ a.m.} & \infty \end{bmatrix} \quad (3)$$

For evaluating $f_v(t_{\text{start}})$, we perform a binary search on the first column of F_v for the first value that is larger than t_{start}.

For a data set of n road segments with m time stamps over all road segments, and k road segments that need to be reported for a query Q, the asymptotic query time of Algorithm 1 is $O(k \log n \log m)$.

3.3 Build a Good θ^*-Structure

In the following, we discuss how to improve the θ^*-structure such that the query time is reduced. First, we discuss what is a good mapping between the road segments and the leaf nodes. This does not improve the asymptotic query time but in the real-world experiments we show its real-world relevance. Secondly, we show how to incorporate the technique fractional cascading (Chazelle and



(a) 13 nodes queried (b) 10 nodes queried

Figure 7. Query where road segments r_i and r_j are frequently served. (a) r_i and r_j are mapped to neighboring nodes. (b) r_i and r_j are mapped to nodes that are far apart.

Guibas, 1986a,b) which improves the asymptotic and experimentally evaluated query time.

Mapping In the following, we want to analyze which order of the leaf nodes of the θ^*-structure $\mathscr{B}$ is advantageous for the query time. In Section 3.2, we described how $\mathscr{B}$ answers a time-window query. Figure 7 shows the query procedure for two different mappings between the road segments and the leaf nodes. It shows that a time-window query Q can be answered more efficiently if all frequently served road segments for Q lie closely together. Then, we can exclude large parts of the θ^*-structure in the query procedure. Intuitively, it would be good to find a mapping between the road segments and the leaf nodes of the θ^*-structure such that neighboring leaf nodes correspond to road segments that are frequently served for the same time-window queries. In the following, we provide an approach that we call `max-order` to generate such a mapping. In the experimental evaluation, we compare this to a random order.

For the `max-order` we compute the maximal vertical distance between pairs of time functions f_v and $f_{v'}$. Let $[t_1,t_2]$ be the temporal range where the time functions f_v and $f_{v'}$ are larger than $-\infty$ and smaller than ∞. Then, we define the maximal vertical distance `max-dist` of the time function f_v and $f_{v'}$ as follows

$$\texttt{max-dist}(f_v, f_{v'}) = \max_{t\in[t_1,t_2]} |f_v(t) - f_{v'}(t)| \quad (4)$$

For the mapping between the road segments and the leaf nodes, we start with a randomly selected road segment and associate it to the first node v. Then, we compute the maximal vertical distance of f_v and the time functions of all other road segments. We associate the road segment with the smallest maximal vertical distance to the second leaf node w. We repeat this procedure, now computing the maximal vertical distances to the time function f_w of node w.

Fractional Cascading To further improve the query time, we use the method fractional cascading (Chazelle and Guibas, 1986a,b, de Berg et al., 2008). Without fractional cascading, we need to evaluate each time-function independently. With fractional cascading, we store pointers at the parent node time-function to the child node time-function. Hence, we need to do just one time a binary search in the time-function instead of searching the time function of each node that we query. This improves the asymptotic query time to $O(k \log n + \log m)$. For more details, we want to refer to the θ-structure by Bonerath et al. (2020).

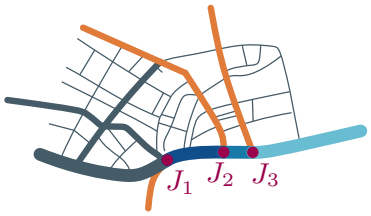


Figure 8. Pre-processing of the road network. All grey roads are never traversed by a bus and are neglected. The brown roads are each one road segment. The road that consists of the tree blue road segments is split at the junctions J_1, J_2, and J_3. At each junction either a bus line is added to the road segment or leaves the road. Hence, we receive the three blue road segments.

4. Experiments

In the following, we describe our experiments on real-world data from the public transportation network of Bonn. First, we give an overview of the pre-processing steps that need to be done to transform the data into our θ^*-structure. Secondly, we evaluate the construction and query times obtained with the different versions.

Data Pre-Processing For our experiments, we used the public transportation network of the city of Bonn, Germany obtained by the VRS GmbH[1] under the data license Deutschland Zero Version 2.0. The network contains (i) 6880 stations, (ii) 567 bus, tram, and train routes which are a sequence of stations, and (iii) 121866 trips which consists of the sequence of stations defined by the routes enriched with stop times. Based on the geometries provided by the routes, we processed the data into a graph. To receive the road segments annotated with the bus-traversal time stamps, we performed several pre-processing steps, i.e.,

1. computing the traversal times for the road segments in between stations by linear interpolation
2. merging time stamps of trips where they traverse the same road segment
3. partitioning road segments where routes come in or leave the street; see Figure 8.

After the pre-processing phase, we have 29159 road segments with at least one, on average 24, and at maximum 2354 bus traversal time stamps. We call this data set `Bonn`.

Although the `Bonn` transportation network is large, there are of course much larger networks. In order to simulate these, we have extended the `Bonn` dataset. In more detail: we added the road segments of Bonn, which are not used in the real-world transportation network and provided them with one to 200 artificial bus traversal times. Thus we end up with 102599 road segments. We call this larger dataset `Bonn-Extended`.

[1] https://www.vrs.de/

Baselines and Versions of the θ-Structure In the following, we compare the response time for time-window queries for frequently served roads. In particular, we evaluate the different versions of the θ^*-structure that we discussed throughout the paper

- `theta*-structure`: Here, we assigned the road segments in a random order to the leaf nodes of the θ^*-structure. We build and query the θ^*-structure as described in Section 3.1 and Section 3.2.
- `theta*-structure-max`: In this version, we assigned the road segments to the leaf nodes according to `max-order` as described in Section 3.3. In order to speed up the construction time, we did only compute the maximal vertical distance to 1000 randomly chosen road segments and chose the one with the smallest distance for computing the mapping between road segments and leaf nodes.

We compare these versions of the θ^*-structure to two simple baselines that not use the θ^*-structure:

- `on-demand-simple`: Here, we count for each road segment the number of time stamps that are contained in the time window and then, report all road segments that are frequently served.
- `on-demand-timefunction`: Here, we pre-compute the time function for all road segments. For a time-window query, we evaluate the time function of all road segments and report all frequently served road segments on-demand.

Note again that the visualization for a time-window query is the same for all four variants θ^*`-structure`, θ^*`-structure-max`, `on-demand-simple`, and `on-demand-timefunction`.

Construction Time The construction of the θ^*-structure is done in a pre-processing step. Hence, the computation time is not critical for the application. Nevertheless, a short construction time is more pleasant. The construction of `theta*-structure` took 2 seconds for `Bonn` and 20 seconds for `Bonn-Extension`. The construction of `theta*-structure-max` took 232 seconds for `Bonn` and 31 minutes for `Bonn-Extension`.

Query Times We performed the experiments with 100 synthetically generated time-window queries. Figure 9 shows the query time evaluation for `Bonn` and Figure 10 for `Bonn-Extension`. The experiments show that `theta*-structure` and `theta*-structure-max` outperform the two baseline approaches `on-demand-simple` and `on-demand-timefunction`. Especially, for time windows that report smaller numbers of road segments our data structures are more efficient. This is reasonable since the theoretical running time for a query of the θ^*`-structure` and θ^*`-structure-max` depend on the number of reported road segments, while the theoretical running time of the `on-demand-simple` and `on-demand-timefunction`. For the smaller dataset `Bonn`, the query times of the data structures close get close to the query times of the on-demand approaches for an increasing number of reported road segments. For the larger dataset `Bonn-Extension`, our data structure still shows query times below 25 milliseconds while the two baseline approaches are consistently above 35 and above 60 milliseconds, respectively. Thus, our data structure can allow exploration in real-time (as described in the introduction 40 milliseconds corresponds to 24 frames per second) while this is critical or not possible with the baseline approaches. The evaluation of the query times shows outliers for both data sets. We explain this with background processes that are performed in the java implementation.

5. Generalization

In the following, we give two ideas for generalizing the θ^*-structure.

5.1 More than One Color

Often it is desired to visualize more information by color encoding the number of bus traversals of road segments. One approach is to use multiple fixed thresholds. We want to emphasize that we can tweak the θ^*-structure to support this visualization. It can be implemented straight-forward as it was discussed for the θ-structure and we want to refer to Bonerath et al. (2020) for more details.

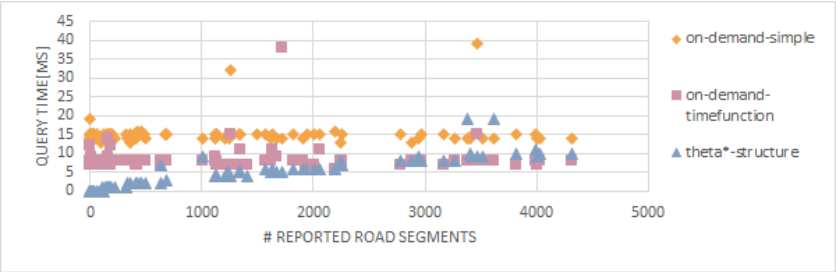
5.2 Application Scenarios

We want to emphasize that our θ^*-structure can also be applied to other scenarios. As mentioned in the introduction, one can, e.g., display all road segments at which at least θ buses stop instead of the road segments that were traversed by at least θ bus.

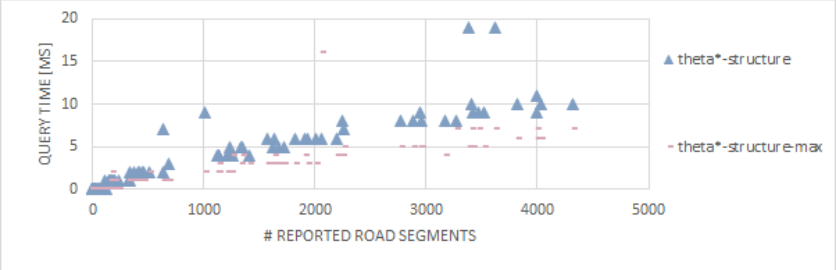
In the following, we provide two examples where we also have different underlying geometry types (polygons and points).

Covid-19 One example is the visualization of Covid-19 infections. As underlying geometries, we use administrative boundaries which corresponds to a set of polygons. Each polygon is annotated with all the time stamps when a Covid-19 infection was reported. With the time slider the user can explore the spatio-temporal pattern, i.e., we display all countries where the number of infections exceeds a given threshold for the queried time window. Our data structure can easily support this scenario. In detail, each leaf node now represents a polygonal border instead of a road segment. The concept of the time function can easily be copied.

Bird Sightings Another example is bird sighting stations. Here, the underlying geometry is the station locations, i.e., a set of points. Each station is annotated with a set of time stamps where each time stamp corresponds to a sighting of a bird. Then, the user can explore the spatio-temporal patterns of bird sightings with the time slider. In this application scenario, the leaf nodes of the θ^*-structure correspond to the point location of a station. The rest of the θ^*-structure remains as previously described.



(a) query times for the θ-structure and baseline approaches



(b) query times for θ^*-structure with and without `max-order`

Figure 9. Query Time Experiments for dataset `Bonn`.

6. Conclusion and Outlook

In this work, we presented the θ^*-structure which is a data structure that supports time-window queries for visualizations of the frequency of public transit service. With respect to an on-demand computation, our data structure gives a query speed-up such that users can interact in real-time while receiving the same visualization. We discussed several speed-up variants of the θ^*-structure. In our experiments with the public transportation network of the city of Bonn, Germany, we achieved query times under 25 milliseconds.

6.1 Acknowledgements

Partially funded by German Research Foundation under Germany's Excellence Strategy, EXC-2070 - 390732324 - PhenoRob.

References

Ahlberg, C. and Shneiderman, B., 2003. Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In: *The Craft of Information Visualization*, Interactive Technologies, Morgan Kaufmann, pp. 7–13.

Ahlberg, C., Williamson, C. and Shneiderman, B., 1992. Dynamic queries for information exploration: An implementation and evaluation. In: *Proceedings of Conference on Human Factors in Computing Systems (CHI'92)*, ACM, pp. 619–626.

Andrienko, G. L. and Andrienko, N. V., 1999. Interactive maps for visual data exploration. *International Journal of Geographical Information Science* 13(4), pp. 355–374.

Andrienko, G. L., Andrienko, N. V., Demsar, U., Dransch, D., Dykes, J., Fabrikant, S. I., Jern, M., Kraak, M., Schumann, H. and Tominski, C., 2010. Space, time and visual analytics. *International Journal of Geographical Information Science* 24(10), pp. 1577–1600.

Bannister, M. J., Devanny, W. E., Goodrich, M. T., Simons, J. A. and Trott, L., 2014. Windows into geometric events: Data structures for time-windowed querying of temporal point sets. In: *Canadian Conf. on Comput. Geom. (CCCG'14)*.

Bannister, M. J., DuBois, C., Eppstein, D. and Smyth, P., 2013. Windows into relational events: Data structures for contiguous subsequences of edges. In: *Symp. on Discrete Algorithms (SODA'13)*, SIAM, pp. 856–864.

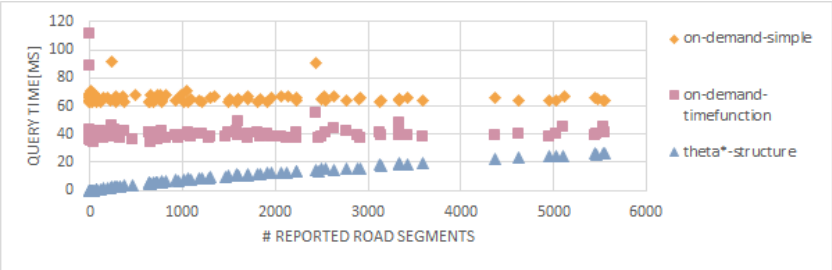
Bokal, D., Cabello, S. and Eppstein, D., 2015. Finding All Maximal Subsequences with Hereditary Properties. In: *Symp. on Comput. Geom. (SoCG'15)*, LIPIcs, Vol. 34, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 240–254.

Bonerath, A., Niedermann, B. and Haunert, J.-H., 2019. Retrieving alpha-Shapes and Schematic Polygonal Approximations for Sets of Points within Queried Temporal Ranges. In: *Advances in Geographic Information Systems (ACM SIGSPATIAL'19)*, ACM, pp. 249–258.

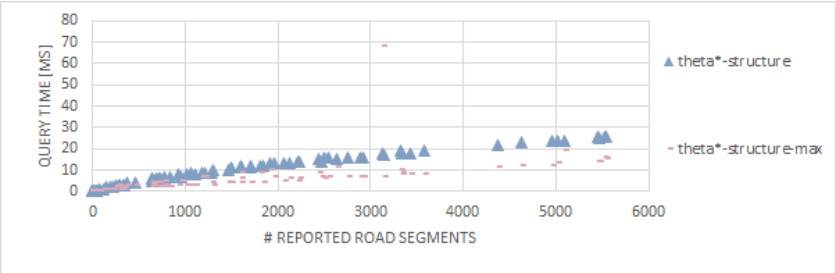
Bonerath, A., Niedermann, B., Diederich, J., Orgeig, Y., Oehrlein, J. and Haunert, J., 2020. A time-windowed data structure for spatial density maps. In: *Proceedings of 28th International Conference on Advances in Geographic Information Systems (SIGSPATIAL'20)*, ACM, pp. 15–24.

Burigat, S. and Chittaro, L., 2005. Visualizing the results of interactive queries for geographic data on mobile devices. In: *Proceedings of 13th ACM International Workshop on Geographic Information Systems (ACM-GIS'05)*, ACM, pp. 277–284.

Chan, T. M. and Pratt, S., 2015. Time-windowed closest pair. In: *Canadian Conf. on Comput. Geom. (CCCG'15)*.



(a) query times for the θ-structure and baseline approaches



(b) query times for θ^*-structure with and without `max-order`

Figure 10. Query Time Experiments for dataset `Bonn-Extended`.

Chan, T. M. and Pratt, S., 2016. Two approaches to building time-windowed geometric data structures. In: *Symp. on Comput. Geom. (SoCG'16)*, LIPIcs, Vol. 51, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 28:1–28:15.

Chanchary, F. and Maheshwari, A., 2019. Time windowed data structures for graphs. *J. of Graph Algorithms and Applications* 23(2), pp. 191–226.

Chanchary, F., Maheshwari, A. and Smid, M., 2018. Window queries for problems on intersecting objects and maximal points*. In: B. Panda and P. P. Goswami (eds), *Algorithms and Discrete Applied Mathematics*, Springer, Cham, pp. 199–213.

Chanchary, F., Maheshwari, A. and Smid, M., 2019. Querying relational event graphs using colored range searching data structures. *Discrete Applied Mathematics*.

Chazelle, B. and Guibas, L. J., 1986a. Fractional cascading: I. A data structuring technique. *Algorithmica* 1(1-4), pp. 133–162.

Chazelle, B. and Guibas, L. J., 1986b. Fractional cascading: II. applications. *Algorithmica* 1(1-4), pp. 163–191.

de Berg, M., Cheong, O., van Kreveld, M. and Overmars, M., 2008. *Computational Geometry: Algorithms and Applications*. 3rd ed. edn, Springer.

Forsch, A., Dehbi, Y., Niedermann, B., Oehrlein, J., Rottmann, P. and Haunert, J.-H., 2021. Multimodal travel-time maps with formally correct and schematic isochrones. *Transactions in GIS* 25(6), pp. 3233–3256.

Hochheiser, H. and Shneiderman, B., 2004. Dynamic query tools for time series data sets: Timebox widgets for interactive exploration. *Information Sciences* 3(1), pp. 1–18.

Kapler, T. and Wright, W., 2005. Geotime information visualization. *Information Visualization* 4(2), pp. 136–146.

Keim, D. A., Andrienko, G. L., Fekete, J., Görg, C., Kohlhammer, J. and Melançon, G., 2008. Visual analytics: Definition, process, and challenges. In: *Information Visualization - Human-Centered Issues and Perspectives*, Lecture Notes in Computer Science, Vol. 4950, Springer, pp. 154–175.

Kraak, M.-J., 2003. Geovisualization illustrated. *ISPRS Journal of Photogrammetry and Remote Sensing* 57(5-6), pp. 390–399.

Robinson, A. C., Peuquet, D. J., Pezanowski, S., Hardisty, F. A. and Swedberg, B., 2017. Design and evaluation of a geovisual analytics system for uncovering patterns in spatio-temporal event data. *Cartography and Geographic Information Science* 44(3), pp. 216–228.

Scheepens, R., Willems, N., van de Wetering, H., Andrienko, G., Andrienko, N. and van Wijk, J. J., 2011. Composite density maps for multivariate trajectories. *IEEE Trans. Vis. Comput. Graph.* 17(12), pp. 2518–2527.

Sun, G.-D., Wu, Y.-C., Liang, R.-H. and Liu, S.-X., 2013. A survey of visual analytics techniques and applications: State-of-the-art research and future challenges. *J. of Computer Science and Technology* 28, pp. 852–867.

Tanin, E., Beigel, R. and Shneiderman, B., 1996. Incremental data structures and algorithms for dynamic query interfaces. *SIGMOD Rec.* 25(4), pp. 21–24.

Williams, M., Kuhn, W. and Painho, M., 2013. Interactive maps: What we know and what we need to know. *Journal of Spatial Information Science* 6(1), pp. 59–115.

Zeng, W., Fu, C.-W., Arisona, S. M., Erath, A. and Qu, H., 2014. Visualizing mobility of public transportation system. *IEEE transactions on visualization and computer graphics* 20(12), pp. 1833–1842.