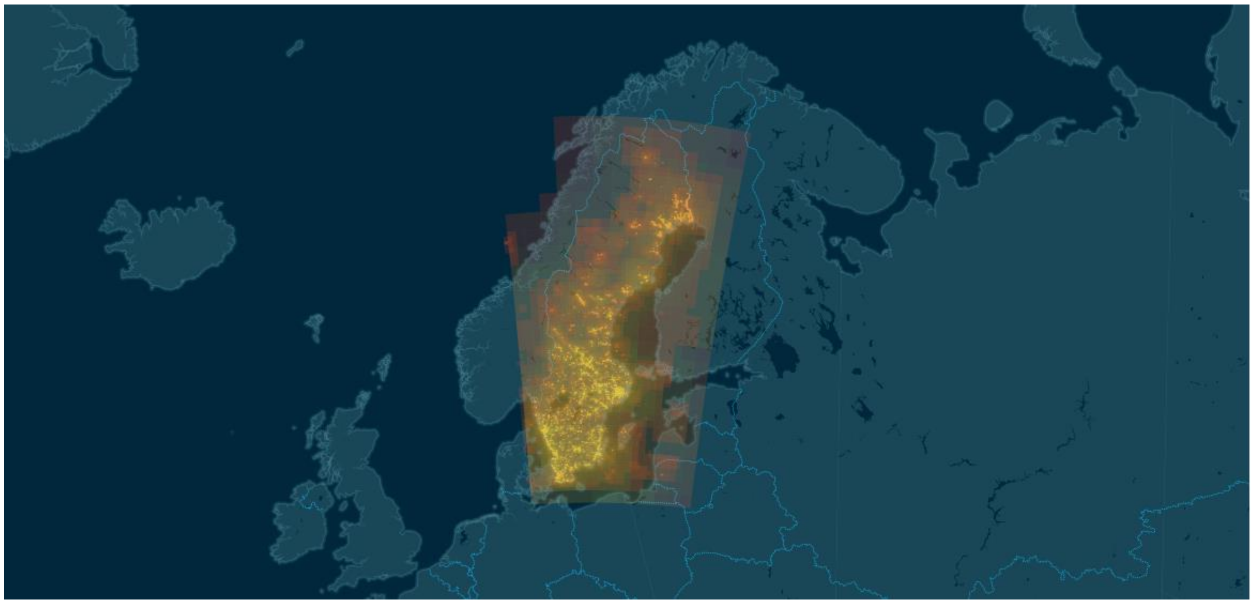


Behavior-driven Tile Caching in Web GIS Applications

Master of Science Thesis in the Programme Software Engineering

ANDERS OLOFSSON

Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, October 2013

Behavior-driven Tile Caching in Web GIS Applications

Anders Olofsson

© Anders Olofsson, October 2013.

Examiner: Matthias Tichy

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover: Heat map visualization of seven days' user accesses, see chapter 6.

Department of Computer Science and Engineering
Göteborg, Sweden, October 2013

Abstract

Tile-based Web GIS is an increasingly popular way of displaying maps online, where the map view consists of square images called tiles. In a typical setting, a majority of the tiles on a tile server remains unused over large amounts of time. The setting of the study is the company Kartena. Investigations are performed which tells if the number of cached tiles at the company Kartena can be reduced while still keeping an acceptable cache hit ratio. The goal is to create an algorithm which gives as good cache hit ratios as possible, meaning that as many accesses as possible are cached server-side. By identifying typical navigation behavior of web map users, optimizations can be made on the server. By identifying and only rendering a small subset of the total amount of tiles in advance, storage requirements as well as rendering times go down.

Two studies related to the problem are identified. Quinn and Gahegan use heuristics and heat maps to create a predictive model, and Garcia et al. use past usage statistics to predict future usage. Using the mentioned studies as well as heat maps and statistical analysis, an algorithm is created which - given a number of tile access logs and a set of domain-specific heuristics - provides a recommendation of which tiles that are suitable for caching. An experiment is performed by examining real usage of the applications and see how well the new model would perform in terms of cache-hit ratio. Depending on the amount of training data used, the experiment indicates that hit ratios of 95% and upwards are possible. The results suggest that the algorithm can be used to realize an on-demand caching solution at Kartena. The resulting algorithm can also be used to reduce storage costs and rendering times in similar settings.

Acknowledgements

First of all I would like to thank my academic supervisor Jörgen Hansson, who early on took great interest in the project and have been an invaluable support. Of equal importance is my industry supervisor Per Liedman, a big thanks for the expertise provided and the time dedicated.

A special thanks also to the team at Kartena AB – both for providing the possibility of conducting the thesis and for being great encouragement and assistance.

Finally, I would like to thank my friends and family for support. The thesis would not have been possible without them.

Contents

List of Figures

List of Tables

1 Introduction

Web Geographic Information Systems (Web GIS), *"Slippy maps"* or simply *"web maps"*, can be considered a trend in cartography (Kraak and Brown, 2002). There are many different map services available and they are slowly becoming a natural part of navigation and surveying tasks.

Web maps are typically composited of square image files called tiles, which are pre-rendered (or *cached*) and put on a tile server. Tiles are then requested from various web applications. The number of tiles on the server depends on the area covered and the number of scale levels used – commonly this number is large. *Caching* all tiles is costly, both in terms of storage required and rendering times. An approach which addresses these issues is "on-demand rendering", which renders tiles only when requested from a web map application.

The setting of our study is Kartena, a Gothenburg-based company developing and maintaining tile-based web applications. Kartena wants to move to an on-demand architecture in order to reduce the amount of pre-rendering that needs to be performed. However, since on-demand rendering leads to slightly longer loading times for the end user, the hope is that the subset containing the most popular tiles can be cached in advance. By investigating different ways of predicting future user behavior, a suitable subset can be identified which will reduce the number of on-demand renderings that have to be performed. Currently, no information of user behavior exists, which is the starting point of the study.

Several previous studies have been made on the subject. Quinn and Gahegan propose a heuristic solution, where typical user patterns are extracted by analyzing heat maps, and Garcia et al. propose a solution where access logs are mined in order to estimate future tile usage. The research introduced in this report describes a similar approach to tile prioritizing, where both heuristics and access log mining are used to prioritize tiles. By doing this, the study offers a new take on the tile caching problem.

Purpose
The aim of the study is to investigate different methods of predicting future tile accesses at the company Kartena. By doing this, the introduction of an on-demand rendering solution is made possible since a partial cache can be built based on the predictions – optimally reducing the tile loading times for the majority of users.

By examining current state of the art approaches to descriptive algorithms and heuristic models and combining the findings, the thesis also evaluates the field of Web GIS tile caching in a new way.

Goals
The primary goal of the study is to develop an algorithm which, based on previous user behavior and a set of heuristics, provides a recommendation of which tiles to cache. Output from the algorithm is a list of tile indexes in the form (x, y, scale), which ranks individual tiles from high to low caching priority. The second goal is to develop a technical solution to the problem, which automatically parses access logs in order to be able to provide a caching recommendation. Heuristics should be defined in a user-friendly way in order to ease the addition of new heuristics.

Results
The result of the study is an algorithm, which provides a cache recommendation by combining the theory presented in two previous studies. When the training data is sparse, the algorithm uses heuristical predictions to provide a cache recommendation and when the training data grows the algorithm adjusts the output according to the learned behavior. When tested on real world data, the

algorithm managed to obtain hit ratios of 95% and upwards. According to Kartena the results are more than good enough to implement an on-demand caching solution.

1.1 Scope

The study investigates the possible business gains (in terms of future cache size and hit ratios) from implementing an on-demand caching solution. The actual implementation of an on-demand caching solution is not performed – due both to time constraints and the fact that Kartena requested a purely investigative approach.

Exact calculations on saved storage space and reduced rendering times are not performed. Both, but especially rendering times, depend on the geographical information covered by each tile – exact calculations are therefore not trivial and were deemed not possible for the given time frame.

The algorithms and models presented are generalizable for all tile-based web GIS systems and are not bound to any particular technical platform or technologies. The study therefore keeps technical details to a minimum. For the interested reader, technical details on how the solution was obtained are presented in Appendix A.

1.2 Outline

The study is started with a background chapter, which aims to create an understanding of the topic and provide necessary knowledge. The related work section follows, where the two studies that are the foundation of this report are presented. A domain analysis is then performed, this is done to enable the heuristic and descriptive analysis, such as specifying the client set and investigating basic tile patterns. The domain analysis is followed by the two main sections of the study - heuristic approach and descriptive approach. The heuristic chapter is heavily based on one of the previously presented studies, where the findings of Quinn and Gahegan are investigated with relation to the domain. A heuristic algorithm is created, which uses the findings of the analysis to create a tile ranking. The descriptive chapter implements two algorithms: the model created by Garcia et al., and the trivial model. The results are evaluated in the evaluation chapter. The study is rounded off with a discussion, followed by the conclusions made along with suggestions for future work.

2 Background

This section explains the technical details of GIS mapping and tile-based GIS systems in order to provide the reader with a sufficient understanding of the problem. The company Kartena is also presented - more specifically the tile rendering pipeline used and the possible business benefits from the study.

2.1 Maps in GIS and tile-based GIS systems

In GIS systems, spatial map information is commonly expressed in vector form (Sample and Ioup, 2010). When displaying a map, the vector information is styled (according to some styling rules) and then rasterized to a visible image. Primitive web-based GIS systems commonly offer real-time server-side rendering of images which is then served to the user. The downside to this approach is the constant rendering that is required – two users viewing slightly different areas of a map would require two separate rasterizing jobs to take place.

Sample and Ioup (2010) identify an alternative way of designing web GIS applications. By dividing the geographical data into square bitmaps of predefined sizes called "tiles", a virtualized map view can be created on-the-fly by combining several tiles. This leads to faster applications as no on-line rendering has to be performed, the tiles are rendered – or *cached* – in advance and then distributed to the users when requested. This is the most common technique used in today's map services (Boulos et al., 2010).

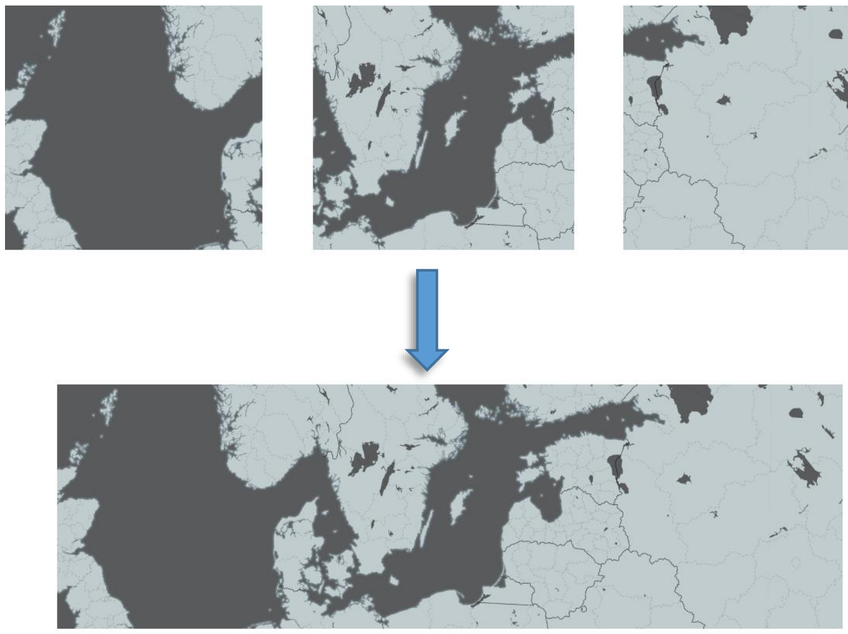


Figure 1: Combining several tiles to a single view

Tiles are commonly ordered in a pyramid-like scheme (Figure 2), where resolution – or *scale level* – increases the further down we get in the pyramid. The topmost scale level has index 0. Tiles at this scale level cover a big area. When creating tiles for scale level 1, tiles for the above scale level (in this case, 0) are typically divided into four smaller parts which in turn are rendered. Each scale level therefore consists of a grid of images, and each scale level can be mapped to a particular map scale. Tiles are referenced using a discrete addressing scheme, namely (x, y, scale) or (x, y, z). For example, (6370, -25696, 13) denotes the tile at scale level 13 where x = 6370 and y = -25696.

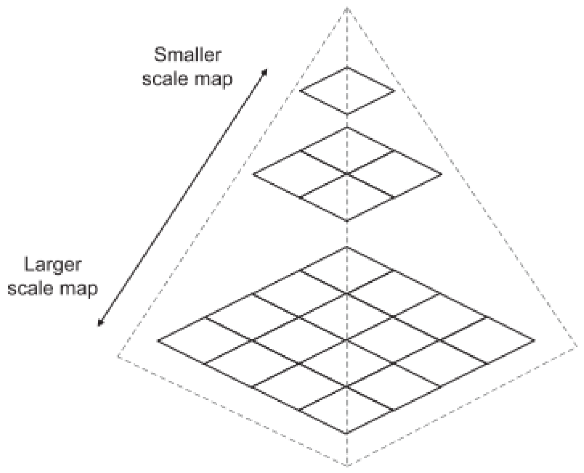


Figure 2: Pyramid tile scheme (Quinn and Gahegan, 2010)

The term "WMS" (Web Map Service) is commonly used to refer to online map applications (OGS, 2006). This study uses the same term to refer to the tile-based map applications developed by Kartena. While a

WMS doesn't have to be tile-based, the definition is broad and the same connection has been done in earlier studies, which motivates the usage of the term.

2.2 Kartena

Founded in 2000, Kartena is a Gothenburg-based company offering GIS services. The company offers route planning tools and positioning services, as well as tailor-made WMS applications for a number of clients – the latter being the focus of this study.

Map data in vector form is provided by Lantmäteriet (the Swedish official body for dealing with cartography issues). The *tile lifecycle,* describing how tiles are created, stored and served to WMS applications, is presented in Table 1.

Step	Task
1	Map data is parsed and put in a PostGIS database for easy access (PostGIS, 2013).
2	Data is imported into the software TileMill, where css-like styling rules are created and previewed (MapBox, 2013).
3	Using the styling rules from step 2, the data is rendered (scale levels 0-14) to 256x256 tiles in PNG format by using the software mapnik (Pavlenko, 2013).
4	Tiles are placed on a tile server which uses the software TileStache (TileStache, 2013), to serve the tiles.
5	Tile requests are logged and stored in plain text files consisting of 24 hours each. Log files are encoded using the Common Log Format, or CLF (W3C, 1995).

Table 1: Tile lifecycle at Kartena

An example URL for the tile (155, -782, 8) is *[serverURL]/8/155/-782.png*. When a tile is served, the tile server logs using the Common Log Format (CLF) (Apache, 2012). Logs are stored in text files consisting of 24h of consecutive log data each; a log file typically contains between 200 000 and 400 000 tile requests.

2.2.1 Kartena's Tile set

The tile set at Kartena is determined by the RT 90 coordinate system as defined by Lantmäteriet (Lantmäteriet, 2013). The geographical boundaries where the RT 90 projection is suitable are set by the rectangle (10.5700, 55.2000) and (24.1800, 69.1000). This area covers the whole of Sweden, along with parts of the Baltic Ocean. For reference, the boundaries are shown in Figure 3.

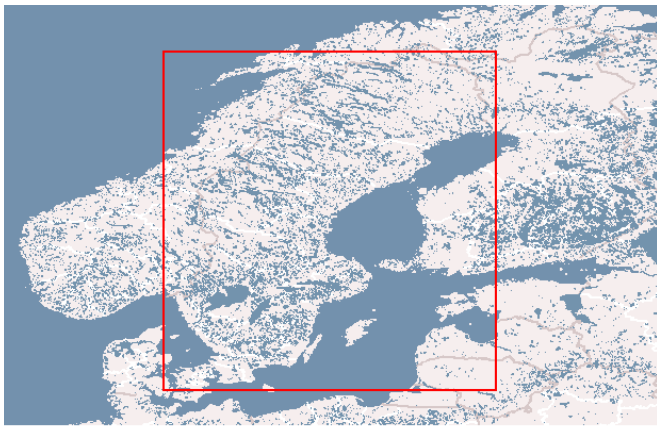


Figure 3: RT 90 boundaries (Spatial Reference, 1997)

Using the boundaries, we calculate the corner tiles for each scale level and in turn compute the total number of tiles. Table 2 shows the number of tiles at each scale level, along with the running total for the previous levels.

Scale level	Upper left tile	Lower right tile	Number of tiles	Running total
0	(0, 1, -4)	(0, 1, -3)	2	2
1	(1, 1, -7)	(1, 2, -6)	4	6
2	(2, 2, -15)	(2, 4, -12)	12	18
3	(3, 4, -29)	(3, 7, -23)	28	46
4	(4, 9, -59)	(4, 15, -47)	91	137
5	(5, 18, -117)	(5, 30, -93)	325	462
6	(6, 36, -234)	(6, 59, -187)	1 152	1 614
7	(7, 71, -468)	(7, 118, -374)	4 560	6 174
8	(8, 142, -936)	(8, 236, -747)	18 050	24 224
9	(9, 285, -1872)	(9, 473, -1494)	71 631	95 855
10	(10, 570, -3744)	(10, 945, -2988)	284 632	380 487
11	(11, 1139, -7489)	(11, 1890, -5977)	1 137 776	1 518 263
12	(12, 2279, -14978)	(12, 3780, -11953)	4 545 052	6 063 315
13	(13, 4557, -29955)	(13, 7561, -23906)	18 180 250	24 243 565
14	**(14, 9114, -59911)**	**(14, 15122, -47812)**	**72 708 900**	**96 952 465**
15	(15, 18228, -119821)	(15, 30243, -95625	290 751 152	387 703 617
16	(16, 36455, -239642)	(16, 60487, -191249	1 163 053 002	1550 756 619
17	(17, 72910, -479285)	(17, 120973, -382499	4 651 970 368	6 202 726 987
18	(18, 145820, -958570)	(18, 241947, -764998	18 607 785 344	24 810 512 331
19	(19, 291641, -1917140)	(19, 483893, -1529995)	74 429 979 938	99 240 492 269

Table 2: Total and individual number of tiles for scale levels 0-19.

As can be seen in the table, level 14 contains 96 952 465 tiles. This large number is the highest amount of tiles that can ever be cached at Kartena, and gives some insight into the numbers dealt with and the problems with an exponential pyramid structure. For example, the introduction of scale level 15 would require 387 703 617 tiles, greatly increasing storage costs and rendering times.

2.2.2 Business Benefits

As previously stated, the large number of tiles leads to long rendering times. Currently, all tiles are rendered in one rendering session, typically lasting several days. Also, bitmaps of Sweden for scale levels 0-14 occupy roughly 153 GB, which raises costs since more server storage is needed. Both initial storage costs and rendering times will be lower when decreasing the amount of pre-rendered tiles, which is the main business benefit of the study. Also, new map data is regularly provided by Lantmäteriet and with shortened rendering times new material can go live more quickly. Similarly, a reduced number of tiles might help in introducing new scale levels, which currently is too costly in terms of both storage and rendering times.

It should be noted that using an on-demand solution, storage costs are likely to increase over time since newly accessed tiles are placed in the cache once rendered. Subsequent accesses to this tile by other users will therefore not require any rendering. In previous attempts of letting certain tile servers run purely on-demand, the resulting number of cached tiles after roughly a year of usage was around 40% of the total tile body. 60% of the tiles were therefore not accessed in over a year, which is an indication that significant optimizations can be made.

The research introduced in this study mainly affects step 3 in Table 3. While changes to the rendering pipeline is outside of the study's scope, the practicalities would consist of configuring mapnik to render the tiles recommended by the algorithm.

3 Related Work

As previously mentioned a number of similar studies have been made, which should be examined to connect the study to existing research. Two existing articles are instrumental to our work: *"A Predictive Model for Frequently Viewed Tiles in a Web Map"* by Quinn and Gahegan (2010) which attempts to come up with heuristical patterns behind tile accesses, and *"A Descriptive Model Based on the Mining of Web Map Server Logs for Tile Prefetching in a Web Map Cache"* (2012) by Garcia et al. which parses access logs to derive individual tile statistics. This section presents those two studies.

3.1 A Predictive Model for Frequently Viewed Tiles in a Web Map

In this paper by Quinn and Gahegan, the authors propose a heuristic approach to the problem of determining the most frequently used tiles in a web map. Microsoft Bing heat maps were analyzed and heuristical predictions of the most important areas were created.

The result of the study is a model (Figure 4) where coast lines, cities, major roads and "points of interest" were deemed important to users. Some additional heuristics are added, such as buffering and the removal of "holes" in otherwise fully cached areas.

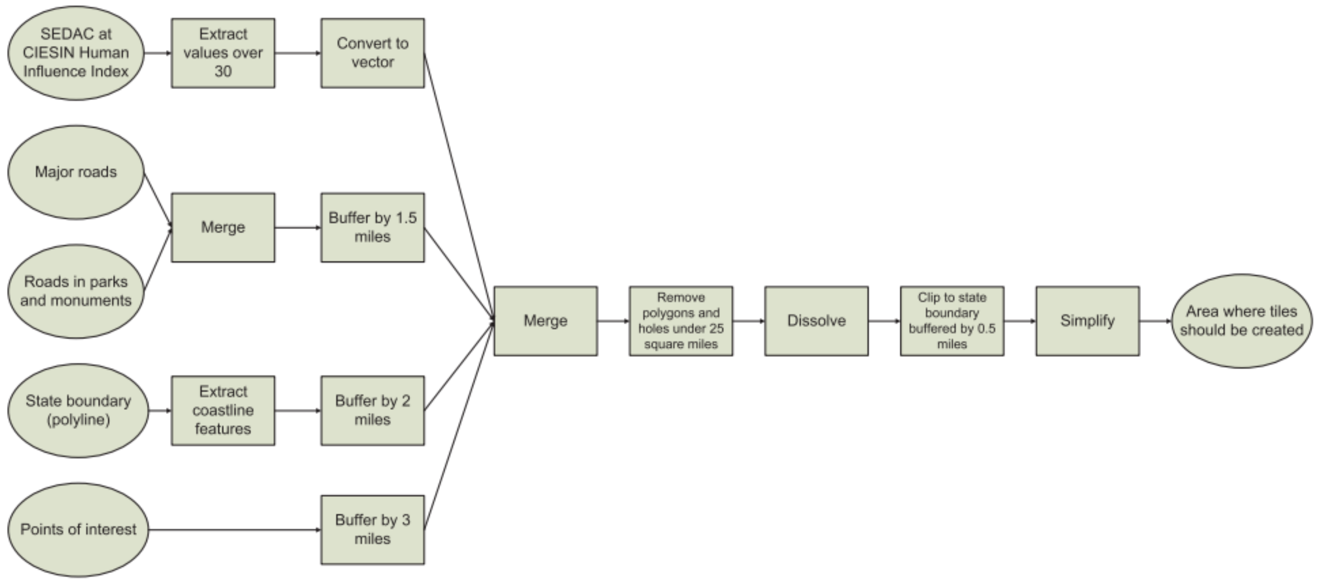

SEDAC at CIESIN Human Influence Index
Extract values over 30
Convert to vector
Major roads
Roads in parks and monuments
Merge
Buffer by 1.5 miles
State boundary (polyline)
Extract coastline features
Buffer by 2 miles
Points of interest
Buffer by 3 miles
Merge
Remove polygons and holes under 25 square miles
Dissolve
Clip to state boundary buffered by 0.5 miles
Simplify
Area where tiles should be created

Figure 4: The resulting model proposed by Quinn and Gahegan

The authors also discuss the potential gains that can be achieved by implementing and using the model. As in Kartena's case, the reduced rendering times and storage costs are the main advantages being identified. Other possible areas of usage are also found including cartography, where the cartographer can put more effort into making selected areas more usable, and aerial imagery updates where selected areas can be updated more often.

The concepts are introduced in a pedagogical way and the results are directly applicable to Kartena's domain. Because of this, the study *"A Predictive Model for Frequently Viewed Tiles in a Web Map"* is used as a starting point for the predictive research introduced in chapter 6. The aim is to evaluate and verify the findings of Quinn and Gahegan in Kartena's domain, and to possibly make use of the findings in the new algorithm.

3.2 A Descriptive Model Based on the Mining of Web Map Server Logs for Tile Prefetching in a Web Map Cache

In this study by Garcia et al. (2012), the authors offer a descriptive approach to tile caching, based on the premise that tile access patterns are slow to change. As in this report, the goal is to create a partial cache method were only a selected number of tiles are rendered. The result is the *simplified model* which is trained using access logs from Spanish nation-wide web services to predict future usage.

Since the number of tiles in most web map services is large, dealing with individual tile statistics is impractical. The services examined include zoom levels all the way to level 19, which together with the (compared to Sweden's) slightly larger area increases the tile count to the order of billions (refer to Table 2 for an estimate). The simplified model therefore uses statistics from a selected *prediction level*, which preferably has a manageable number of tiles. The simplified model then uses this level to determine the probabilities of tile accesses at other levels.

The performance and accuracy of the model is examined by comparing its output to the access data from other dates (more specifically, access logs from new dates). Multiple tests were performed by comparing different prediction levels to "target" testing levels, with the result being a matrix that shows how particular scale levels benefit from different prediction levels. The results varied greatly, with the best prediction levels resulting in hit ratios between 70-95% and the worst prediction levels resulting in hit ratios around 8-15%. According to the authors, the results indicate that the potential gains from using a similar model are high.

Even though the word "prefetching" is used in the title, the study mainly describes an application where pre-rendering and caching is used – i.e. the same as in Kartena's case. The similarities of the domains motivate further investigation of the findings of Garcia et al.; the simplified model is therefore implemented and compared in the evaluation chapter.

3.3 Additional Studies

While somewhat sparse, other studies exist which can be used as starting points. Garcia et al. published another study in the same area, *"A Cache Replacement Policy Based on Neural Networks Applied to Web Map Tile Caching"* (Garcia et al., 2011) The approach is a little different than the before mentioned study, instead of the simplified model an Artificial Neural Network (ANN) is used to provide a tile cache recommendation. Since more extensive background knowledge was needed the study was not examined closer.

The paper *"Hotmap: Looking at Geographic Attention"* by D. Fisher (2007) discusses the applications of tile request heat maps, as well as the design choices involved in and the lessons learned from creating such systems. The system of focus is "Hotmap", which visualizes the tile requests made to Microsoft Surface (currently Bing maps). Similar to Quinn and Gahegan's study, Fisher concludes that tile popularity tends to follow population. Borders of many kinds were also interesting, such as coastlines, roads and even the borders of available map data. As the study is cited in Quinn and Gahegan's work and many of their ideas are directly derived from Fisher's study, focus is put on the former. The lack of actual models to use also supports the decision.

4 Domain Analysis

An initial analysis of the domain is here performed. This sheds light of various characteristics of the domain, which helps in making the following algorithm more accurate. Data gathering methods are explained, followed by the client base. Lastly the scale levels are investigated.

4.1 Tile Statistics and Data Gathering

Before any analysis can be done, tile statistics need to be collected. For a number of reasons, tile statistics are obtained from web access logs. Access log mining is the data gathering technique used by a number of related studies, for example the mentioned study by Garcia et al. Also, in the paper *Mining E-Commerce Data: The Good, the Bad, and the Ugly* (2001), Kohavi discusses different approaches to the mining of website data in order to gain insight into customer patterns. The author describes different means of acquiring data in an e-commerce setting, and while the domain is different the methods would be similar in a Web GIS setting. Access logs are already available at Kartena, and exactly describe actual tile requests which are things that speak to their advantage. The downside is that they cannot be modified to log more things if needed.

An alternative to access logs would be to implement some sort of system which monitors user behavior at browser level. This way, detailed information could be obtained such as time spent looking at certain areas. This was deemed to be too time consuming and error prone however, not to mention the legal issues that would come into play.

4.2 Non-linear Access Patterns

In the WMS applications studied by Garcia et al., a non-linear access pattern was found. Tile accesses where found to follow the *80:20 rule* meaning that 20% of the accessed tiles received roughly 80% of the total hits. In simpler words, this means that a small portion of the accessed tiles receive a majority of the hits. Can similar patterns be found at Kartena? Figure 5 demonstrates the relationship between number of requests and number of tiles for seven days of investigated access logs.

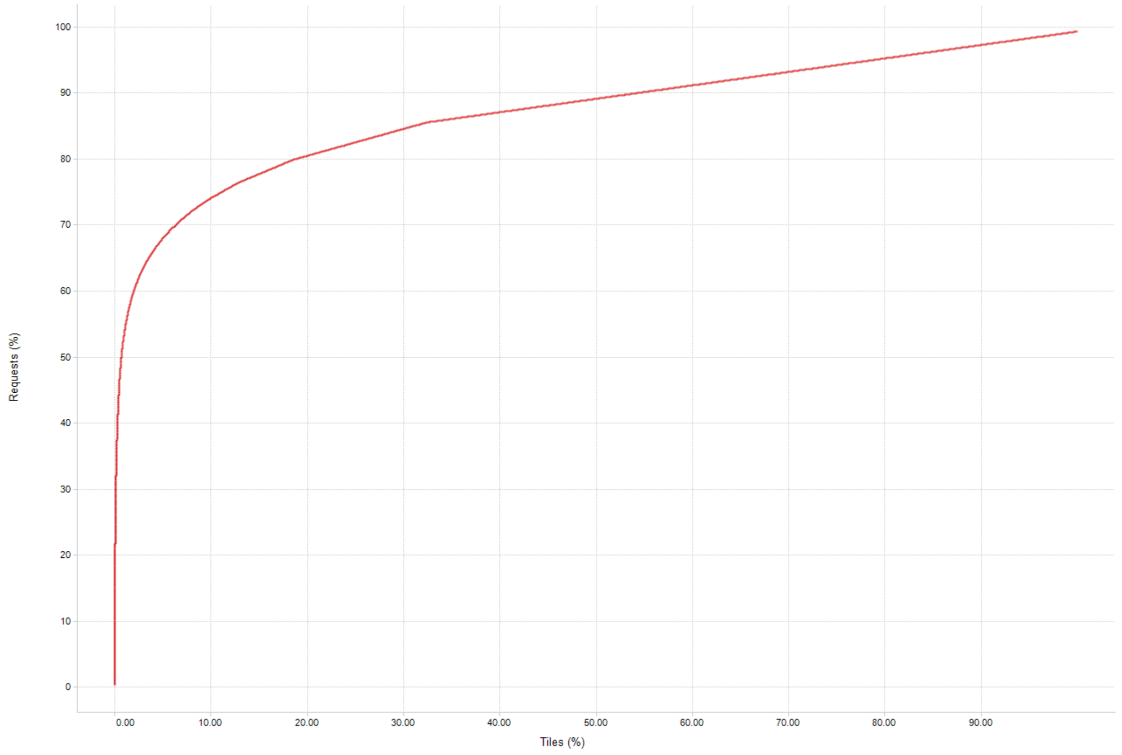


Figure 5: Percentile of requests for 7 days of access data

Tile accesses at Kartena also seem to follow the 80:20 rule. This shows that the tile accesses, although not coming from public general-purpose WMS applications, already show some characteristics of such usage (Garcia et al., 2012). This motivates the use of both Garcia et al.'s and Quinn and Gahegan's results as starting points to the study.

4.3 Clients

The client-owned WMS applications are the driving factor behind the tile accesses. Considering this, it is motivated to specify the client base in order to be able to investigate certain patterns. Table 3 displays the referrers found in seven days of access logs, ordered by number of requests made from each referrer. A brief description of each WMS is presented, along with the size in percentage of hits as well as the geographical areas of interest where this is determinable.

Client	**WMS description**	**Hits**	**Hits (%)**	**Areas of interest**
Client 1	Map application of a Swedish ice cream company, showing locations where the ice cream trucks stop.	907913	39.544	All over Sweden
Client 2	Map applications of a web developer, specializing in producing maps for community transit services.	549684	23.941	Bus stops and train stations in various Swedish cities
Client 3	Map view of a website listing parking opportunities in Stockholm.	218538	9.5184	Stockholm
Client 4	Map application of a sport streaming website, showing current and future events.	132909	5.7888	Depends on sport event covered
Client 5	Map view of a student housing agency located in Gothenburg.	132848	5.7862	Göteborg
Client 6	Map application of a Swedish betting site, showing physical locations.	129219	5.6281	Populated areas all over Sweden

Client 7	Map application offering travel directions by bus or train.	44860	1.9539	Populated areas all over Sweden
Client 8	Map view of a Swedish ferry line, showing all stops of a Swedish ferry line.	42915	1.8692	Göteborg
Client 9	Map application of a transit company located in Kronoberg county.	30654	1.3351	Mainly Växjö and Alvesta, travel destinations can be all over Sweden
Client 10	Various accesses where the referrer could not be determined (localhost, or no referrer at all).	25953	1.1304	
Client 11	Map application of a car service company showing all service stations.	14464	0.6300	Populated areas all over Sweden
Client 12	Map application of a transit company located in Kalmar.	13058	0.5687	Mainly Kalmar, travel destinations can be all over Sweden
Client 13	Map application of an amusement park located in Gothenburg.	10763	0.4688	Göteborg
Client 14	Map application of a bicycle vendor chain, showing all stores.	10299	0.4486	Stockholm
Client 15	Route planning application used by a major Swedish construction company.	6785	0.2955	All over Sweden
Client 16	Map application of a Swedish county, showing public services.	5069	0.2208	Skåne
Client 17	Map application of an energy company, showing all offices.	4256	0.1854	Various Swedish cities
Client 18	Various requests made through a demo application of a Kartena-developed extension to the software Leaflet.	2979	0.1298	All over Europe
Client 19	Web application of a private health care company, showing all clinics.	2667	0.1162	Stockholm
Client 20	Mobile app capable of displaying participants in running competitions.	2583	0.1125	Vaxjo, Göteborg
Client 21	Various administrative applications and tools at Kartena.	1671	0.0728	Sweden
Client 22	Web site of a Scandinavian shopping centre company, showing existing and planned shopping centres.	1598	0.0696	Partille, Malmö, Trollhättan, Kristianstad, Helsingborg, Borlänge, Örebro, Norrköping, Karlstad, Stockholm, Torp
Client 23	Map application of a Swedish machine leasing company, showing all store locations.	1543	0.0672	Various Swedish cities
Client 24	Web view of a Swedish concrete company, listing all office locations.	1383	0.0602	Various Swedish cities
Client 25	Map view of the local branch of a major Swedish banking organization.	520	0.0226	Borås, Bollebygd, Mark, Svenljunga

Client 26	Map view of a Swedish insurance company, listing all office locations.	436	0.0190	Major to minor Swedish cities
Client 27	Map view of a Swedish estate listing website, showing properties for sale.	235	0.0102	Populated areas all over Sweden
Client 28	Map view of a Swedish hostel organization, showing hostel locations.	98	0.0043	Populated areas all over Sweden
Client 29	Map view of a Swedish hotel, displaying the hotel location.	30	0.0013	Stockholm
Client 30	Map view of a fuel station company, listing all fuel station locations.	16	0.0007	Cities all over Sweden
Client 31	Map application of an office supply company, listing all stores.	5	0.0002	Cities all over Sweden

Table 3: Description of clients

Some interesting facts can be observed, such as the size differences between the clients. The smallest clients are almost negligible in size, while the biggest ones contribute to the most hits. Since the tile set is used only by the above clients, their areas of interest should in some way affect the resulting cache suggestion. This is further discussed in the heuristics chapter.

4.4 Scale Levels

Using the same seven days of sample data, we can look at the popularity of individual scale levels. Table 4 shows the scale levels ordered by number of hits.

Scale level	**Number of requests**	**Requests (%)**
13	295804	12.88
11	280009	12.19
12	272919	11.88
14	202802	8.831
8	199882	8.704
10	192903	8.400
9	164353	7.157
5	161306	7.024
1	143695	6.257
7	135461	5.899
4	115703	5.038
6	79116	3.445
3	27579	1.201
2	24752	1.078
0	179	0.008

Table 4: Scale levels ordered by number of requests

As evident by Table 4, higher levels are typically more popular than lower levels. Higher levels offer users a more detailed view of an area than lower levels which may be one of the reasons behind their popularity.

Higher scale levels also have a much higher tile count, which together with browser caching is one of the reasons they get more requests. For example, level 0 only contains 2 tiles, as we saw in Table 2. When a user visits scale level 0, those tiles are typically placed in the browser cache and not reloaded when he/she zooms in and out in the application. Higher scale levels contain a much larger number of tiles and "new" tile accesses are much more likely to occur.

It should be noted that scale levels probably are heavily affected by the WMS applications. For example, level 1 places quite high compared to levels 0, 2 and 3. The reason behind this could be that level 1 offers a reasonable sized overview of Sweden, offering a good starting point for several types of WMS applications. For example, a popular WMS which uses scale level 6 in its starting view makes this level place higher in the above table, as the starting view tiles are requested by all users. If the WMS is also clearly restricted to a particular area, level 6 should be prioritized high *especially* in this particular area – other areas might not need as much focus on scale level 6. No WMS-specific analysis of scale level popularity has been performed however, this is left as a possible starting point of further studies.

5 Heuristics

By manually analyzing previous behavior, is it possible to create an understanding of common user patterns and "guess" where future users will look? This section presents the heuristic part of the algorithm.

Initially, a heat map analysis is performed in order to get an overview of the domain. This is followed by a review of the study done by Quinn and Gahegan - particularly how the findings apply to Kartena's context. Finally, the resulting heuristic algorithm is presented.

5.1 Heat Map Analysis

Heat maps have been previously used to analyze user behavior in WMS applications, for example Fisher (2007). By visualizing user accesses, user patterns and other characteristics can be estimated. To a large extent, the heuristic research is based on heat maps.

In the following heat maps, tiles are represented as a colored tetragons. Tiles are colored from red to yellow, corresponding to an increasing number of hits. As tile accesses frequencies are skewed (recall the 20:80 rule and Figure 5), the coloring has to reflect this to provide a linear visualization. By default, a linear coloring scheme is used. Using a linear coloring scheme, very few tiles are colored yellow as popular tiles are very rare compared to un-popular tiles. Fisher (2007) proposes that a logarithmic color scaling is used instead, which was used as a starting point for coloring. It should be noted that tiles with no popularity (i.e. received no hits) are not rendered at all.

Using an opacity value of 40%, the tiles are superimposed on a background layer - in essence a geographical map. Fisher states that a low-contrast greyscale coloring should be used for the backgrounds as the colored tiles then are easy to distinguish from the backgrounds. However, attempts made found that greyscale images conflicted with the low opacity tiles – a darker background proved to be more suitable and was therefore chosen.

When constructing heat maps, it has to be decided how much data should be displayed. Early investigations found that the number of accesses was typically smaller during weekends. Since the areas of interest and in turn user behavior also might change, the data used should contain at least 7 days of access data in order to ensure that the changed behavior is covered. On the other hand, the time required to render increases with larger data sets. Rendering a single heat map image typically takes a couple of minutes – a trade-off is therefore made between the data size and the heat map's usability in terms of rendering times. Also, heat maps get cluttered and hard to interpret with more data points. It was decided that seven days' worth of access logs were enough to perform an initial heat map analysis.

In order to provide the reader with an understanding of the tile sizes, Figures 6-20 present a selection of the available scale levels along with an example snapshot of tiles on this level. The heat map was created from 7 days of access logs, consisting of 2 311 954 tile requests.

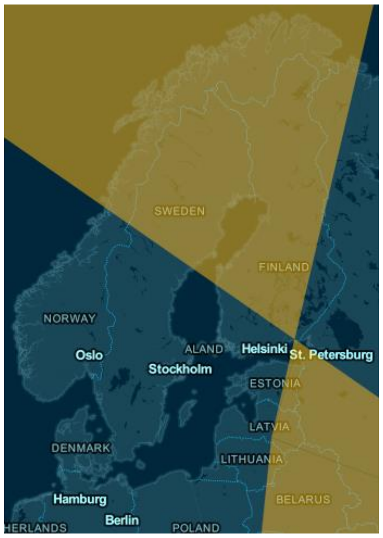


Figure 6: Scale level 0

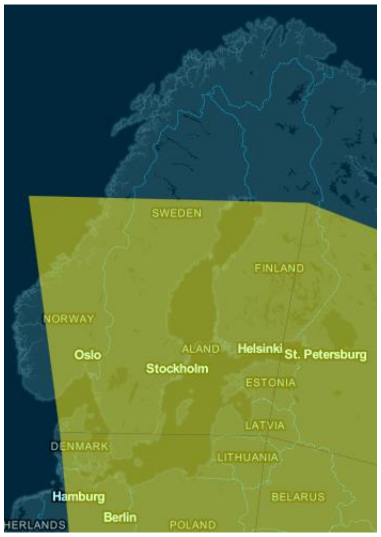


Figure 7: Scale level 1

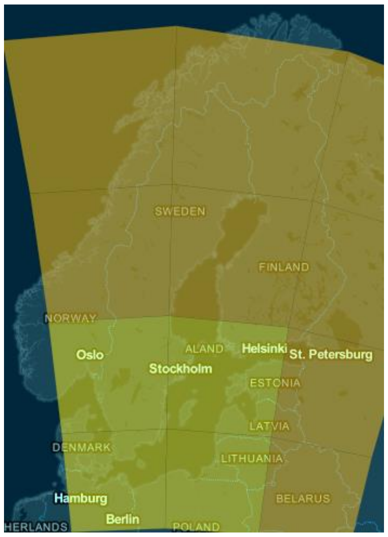


Figure 8: Scale level 2

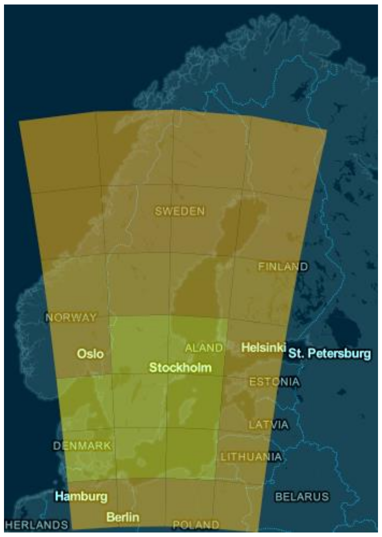


Figure 9: Scale level 3

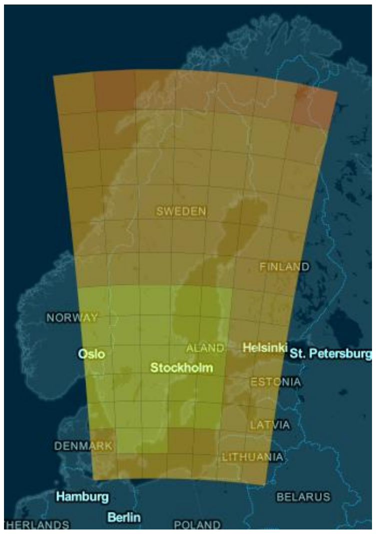


Figure 10: Scale level 4

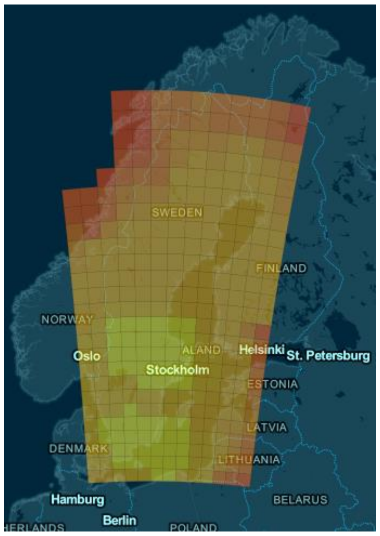


Figure 11: Scale level 5

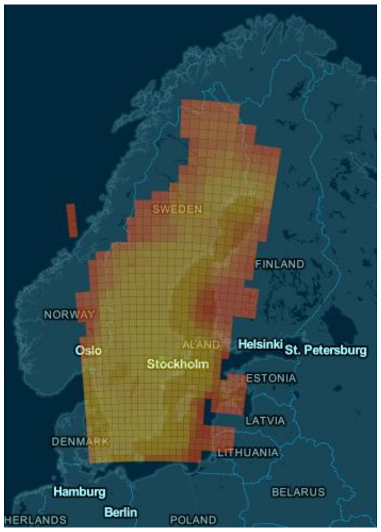


Figure 12: Scale level 6

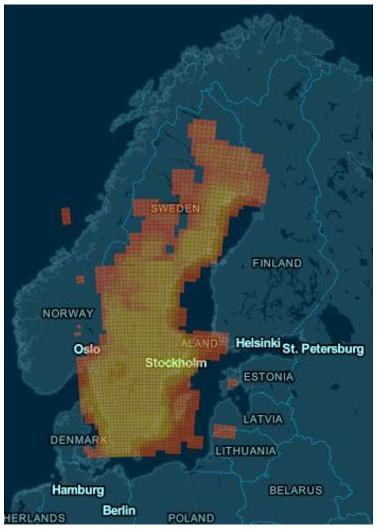


Figure 13: Scale level 7

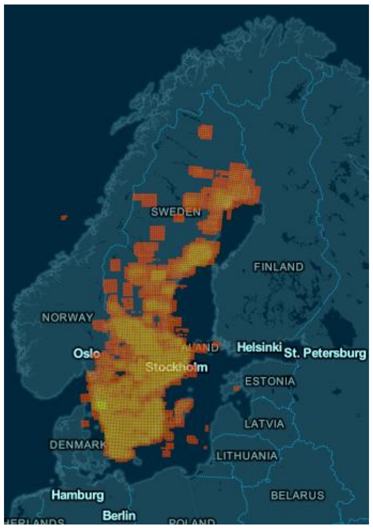


Figure 14: Scale level 8

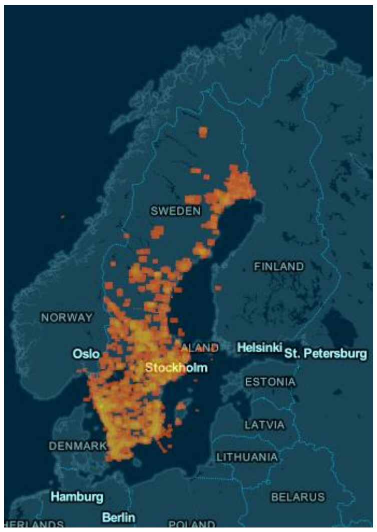


Figure 15: Scale level 9

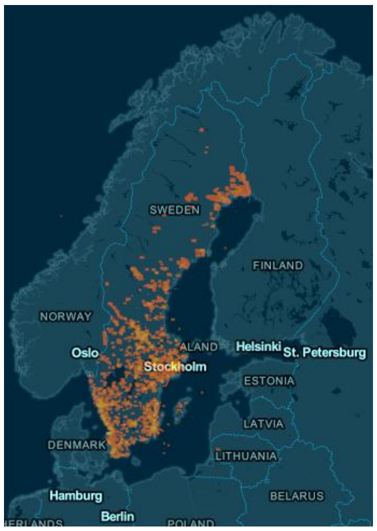


Figure 16: Scale level 10

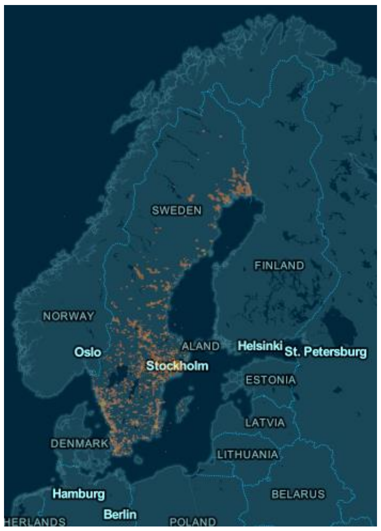


Figure 17: Scale level 11

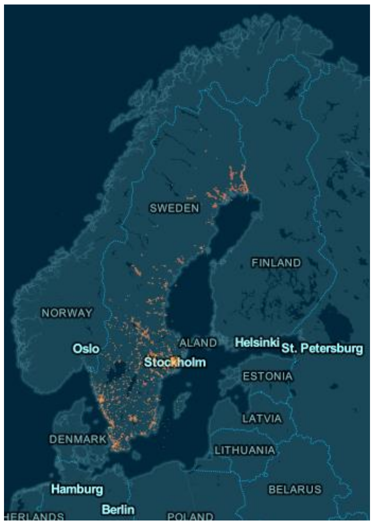


Figure 18: Scale level 12

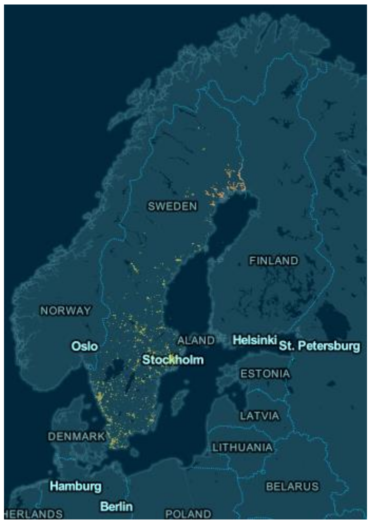


Figure 19: Scale level 13

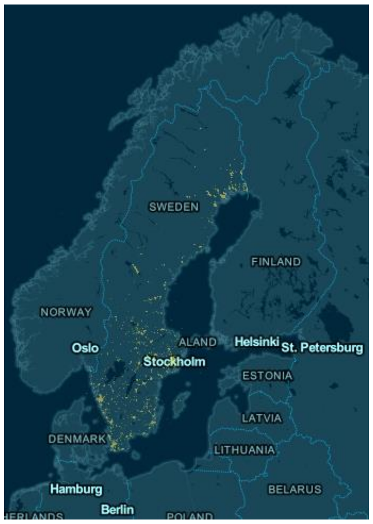


Figure 20: Scale level 14

Figures 6-20 show, especially on higher scale levels, that user interest is focused on a few, quite specific, areas. Lower scale levels are less specific, but still show some interesting patterns. The north of Sweden is clearly of less interest than the southern parts. Major cities are clearly more popular than other areas, e.g. Gothenburg in scale level 8.

The square shape of the tiles are clearly visible. Since Kartena and the heat map visualization software TileMill use different *map projections*, some distortions are present. The "curved" characteristics of the tiles is an effect of the conversion from Kartena's RT 90-projection to *WGS 84* (Spatial Reference, 2007), which is the projection used by TileMill. The distorted look of level 0 is another consequence. This behavior is typical and will not affect the study in a major way – besides level 0 the area covered by the tiles are the same in both projections. Generally, lower scale levels are left out in the following heat map renderings as those levels tend to hide too much of the tiles beneath.

When overlaying tiles, user patterns become clearer. Figure 21 shows a compilation of zoom levels 4-14, with a lowered tile opacity in order for several scale levels to be visible at once. Generally, lower scale levels are left out in the following heat map renderings as they tend to hide too much of the tiles beneath.

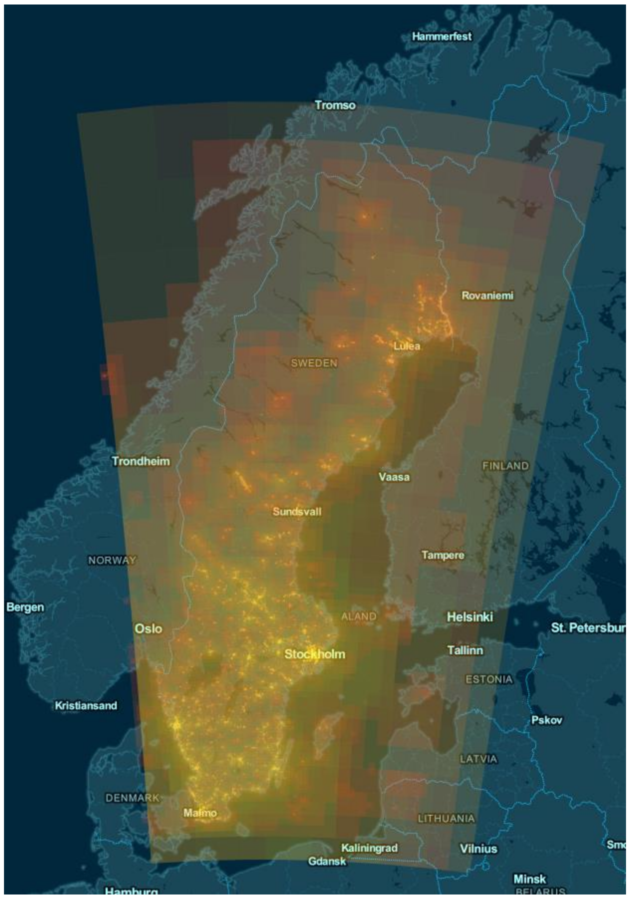


Figure 21: Low-opacity compilation of scale levels 4-14

Some interesting observations can be made. Land areas are clearly more popular than ocean areas, which is reasonable. It also appears that a high portions of the hits are based around and on cities. Some sporadic patterns can be found, such as “lines” in the images which indicates user panning along some route, possibly a road. This is one of the areas which Quinn and Gahegan deemed important, a more thorough analysis is made in the literature review section.

5.2 Comparison to Existing Studies

Quinn and Gahegan (2010) propose four different areas which benefits a predictive model. The areas are presented below, along with reasoning of how they apply to the investigated data. Do the findings translate well to Kartena’s domain?

5.2.1 Populated Places

The authors identify populated areas as typically being interesting to users. The assumption is largely based on Fisher's study (2007), which found that users tend to look where they live. Also, users not living in a populated area will still have an interest populated regions, as they might work or plan to go there.

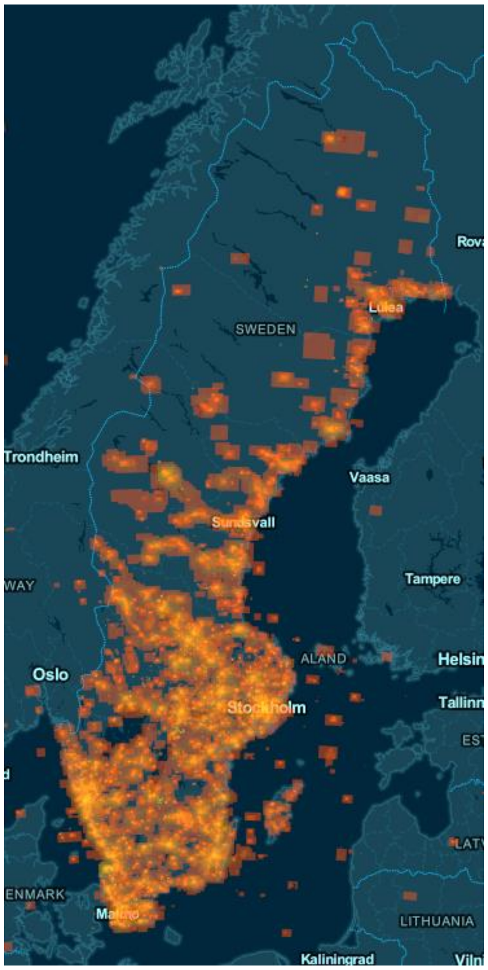


Figure 22: Populated areas for scale levels larger than 8

When closely examined at different zoom levels, Figure 22 shows that yellow tiles are frequently located in areas that are more populated. In unpopulated areas, tiles tend to be red or transparent, indicating few or no hits. As mentioned in the heat map analysis chapter, tile requests in the north of Sweden decrease to the point of being non-existent, probably because there is not much focus by the clients in this region. A conclusion is that cities are typically more interesting than areas in between.

While populated areas are interesting to users, we cannot assume that tile popularity follows city size. Table 5 shows the 20 biggest cities in Sweden ordered by population, along with their corresponding number of tile requests based on the geographic position shown in Figure 23. Tile requests with scale level larger than 6 were used, lower levels would make the tiles too unspecific for city interest conclusions to be made.

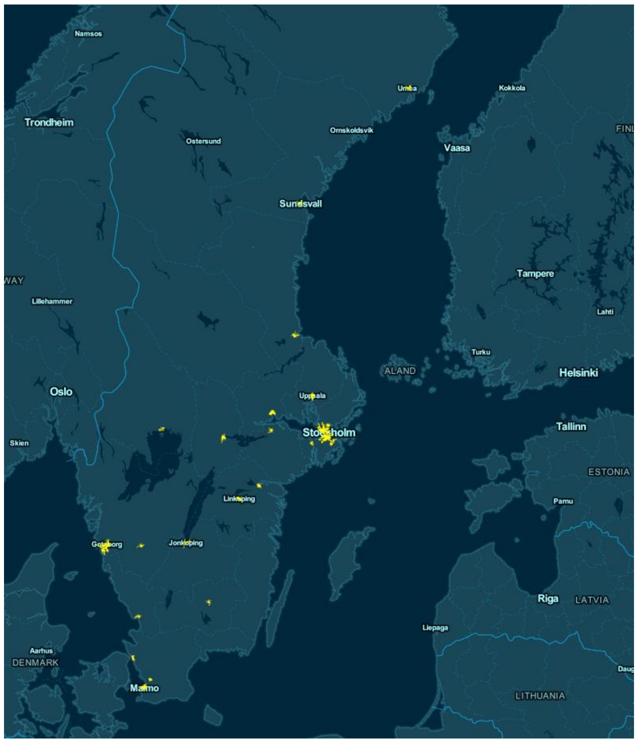


Figure 23: Locations of the 20 biggest cities in Sweden

City	**Population**	**Hits (scale >= 6)**
Stockholm	1372565	204139
Göteborg	549839	139080
Malmö	280415	12500
Uppsala	140454	6633
Västerås	110877	22660
Örebro	107038	27205
Linköping	104232	3417
Helsingborg	97122	6746
Jönköping	89396	3029
Norrköping	87247	2740
Lund	82800	6898
Umeå	79594	2453
Gävle	71033	3528
Borås	66273	1937
Eskilstuna	64679	12877
Södertälje	64619	4742
Karlstad	61685	1746
Täby	61272	6024
Växjö	60887	54420
Halmstad	58577	14104

Table 5: The 20 biggest Swedish cities, ordered by population.

City	**Population**	**Hits (scale >= 6)**
Stockholm	1372565	204139
Göteborg	549839	139080
Växjö	60887	54420
Örebro	107038	27205
Västerås	110877	22660
Halmstad	58577	14104
Eskilstuna	64679	12877
Malmö	280415	12500
Lund	82800	6898
Helsingborg	97122	6746
Uppsala	140454	6633
Täby	61272	6024
Södertälje	64619	4742
Gävle	71033	3528
Linköping	104232	3417
Jönköping	89396	3029
Norrköping	87247	2740
Umeå	79594	2453
Borås	66273	1937
Karlstad	61685	1746

Table 6: The 20 biggest Swedish cities, ordered by number of hits.

It is evident that there is no clear correlation between city population and the number of hits. Stockholm and Gothenburg place high in the hit ranking, which could be an effect of both many clients being based in this region as well as the high population of those cities. However, Malmö receives relatively few hits considering it is the third biggest city in the country. Also, Växjö places third in the hit ranking despite having a relatively small population. This is probably an effect of Client 9 offering services in the region leading to an increased number of tile accesses. Växjö's high ranking indicates that client areas of interest clearly affect the request numbers – if Client 9 would be removed the area would most likely receive much a less number of hits. This should somehow be reflected in the heuristical model.

As seen in Table 3, the bigger clients (Client 1 and 2 for example) together stand for more than half of the total accesses. These clients do not have specific regions of interest, which makes it difficult to estimate the populated areas that are interesting to them. Under the assumption that WMS users look at areas close to their home and that user locations are related to city size, it is reasonable to also let the size of populated areas be a driving factor.

Data of populated areas in Sweden (essentially what is seen in Figure 23, containing every Swedish city and town) was obtained from Lantmäteriet. The data was reasonably easy to parse which made it suitable for use in the heuristic algorithm.

The conclusions are:

- Populated areas are interesting to users and should be cached.
- Client areas of interest, where this can be determined, should be the driving factor which decides tile priority. Following this, cities should be ranked by population.

5.2.2 Major Roads

Quinn and Gahegan identify major road networks to be an important part in WMS applications. In their study, roads where clearly popular to users – a lot of user interest was based around roads despite not being located in other areas of interest such as cities.

While not as obvious as for Quinn and Gahegan's research, some evidence that roads are interesting can be found. In Figure 24, a single user of Client 15's WMS (the route-planning WMS) did a search between the cities Bjärred and Hässleholm and presumably panned along the proposed route.

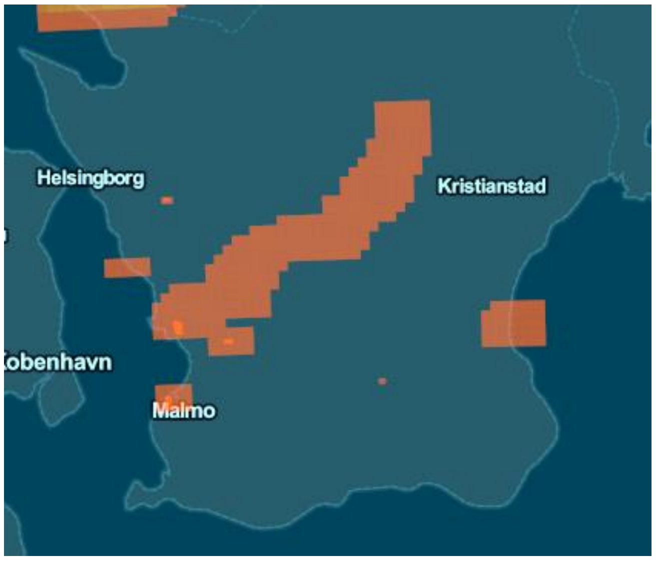


Figure 24: Example of a road panning event

In the analyzed data, only three occurrences of road panning could be found. All occurrences came from the WMS of Client 15, and used scale levels 9, 10 and 11. While not enough data to draw definite conclusions, the absence of higher scale levels seems relevant - panning hundreds of kilometers using the highest zoom level would probably take too long.

Road data was found to be hard to parse and rank, which makes it difficult to build a reasonable heuristic conclusion. Unlike city data which can be easily ranked on population, road data did not contain any meta-information which could be used for ranking. While a road cache most certainly would increase the cache hit ratio it is uncertain how efficient the increase would be. More tiles leads to more cache hits, but for the model to be efficient more evidence should exist in order for a road cache to be motivated. A large increase in cached tiles does not motivate a small increase in cache hit ratio.

The conclusions are:

- If the number of route planning applications is high, the amount of road caching should increase in priority.
- If roads are cached separately, this should be done on scale levels 9, 10 and 11.

5.2.3 Coastlines

The paper states that coastlines are accessed more frequently than other areas. The authors mention that this may be an effect of many roads being situated along coast lines which may explain this behavior. Still, they decide to independently cache all coastlines independent of nearby roads.

No real evidence of coastline interest can be found in the gathered data. The high interest in the western coast of Sweden (as evident in Figure 22) is mainly due to the many applications being based here. The northern Baltic ocean coastline has no apparent hit patterns close to coastlines besides cities. The conclusion is that coast lines are not particularly interesting to users.

5.2.4 Points of Interest

The authors deem certain "points of interest" as typically being popular to users. Points of interest includes national parks, mountain peaks as well as universities and historic buildings.

No real evidence of similar user behavior can be found. Since the WMS analyzed by Quinn and Gahegan is a public general purpose one, the findings may not be applicable. When extending the time range analyzed, some examples of "random" tile accesses can be found. For example, a user of the WMS provided by Client 3 investigated a wilderness area in northern Sweden (Figure 25). The findings were too few and infrequent however, making it hard to make a heuristic conclusion.

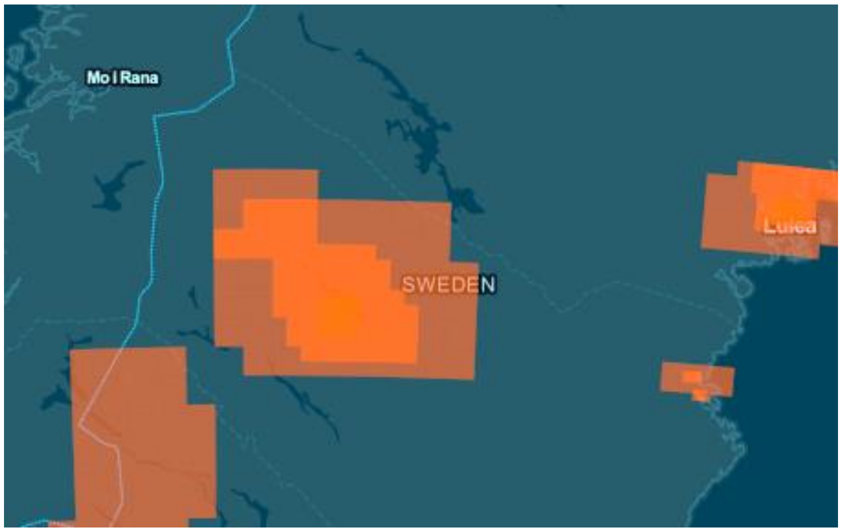


Figure 25: Example of a "random" tile access pattern

In Kartena's domain a "point of interest" could possibly be translated to the clients' areas of interest. For instance, if a popular WMS provides many tile requests to a specific area other than cities, this area should be cached. Again, the client's areas of interest should be a driving factor behind the heuristical model. Other than this, "points of interest" as defined by Quinn and Gahegan is not applicable to the current client set.

5.2.5 Review Conclusion

Some of the findings of Quinn and Gahegan translated well to this context.

- Populated areas should be cached, using both city population and WMS interest. This means that if a city both has a high population and a high WMS interest, this should be reflected in the output.
- Road networks are not cached, both because the reward is not clear and the data availability is limited.
- Neither coast lines nor points of interest as defined by Quinn and Gahegan are suitable caching candidates.

The differences between the general WMS application investigated by the authors (Microsoft Bing) and more client-driven WMS applications, such as Kartena's, are evident. Only populated areas translated reasonably well to Kartena's setting, and road networks, coast lines and points of interest did not translate well. While Kartena's domain was similar to general WMS applications in terms of access patterns (recall the 80:20 rule), it is clear that user behavior does not have to be similar.

5.3 Filling and Buffering

By default, the city data only contains the actual *populated* areas for each city. In some cases, this results in "holes" in the data at the location of parks, bodies of water and the like. An example of this can be seen in Figure 26. A scrolling user might find it annoying to encounter non-rendered tiles in the middle of a city, as those tiles would take longer to render. Figure 27 shows the city data with holes removed.

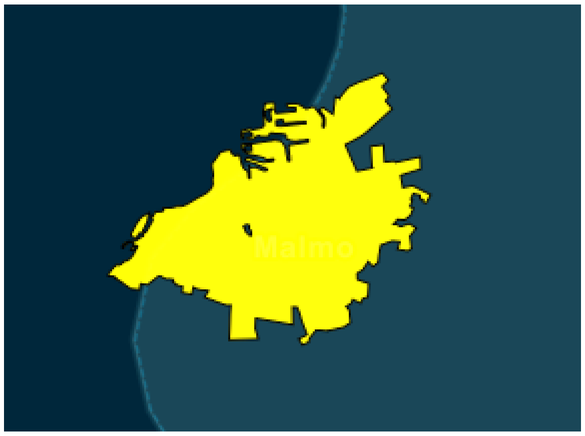


Figure 26: Malmö with holes

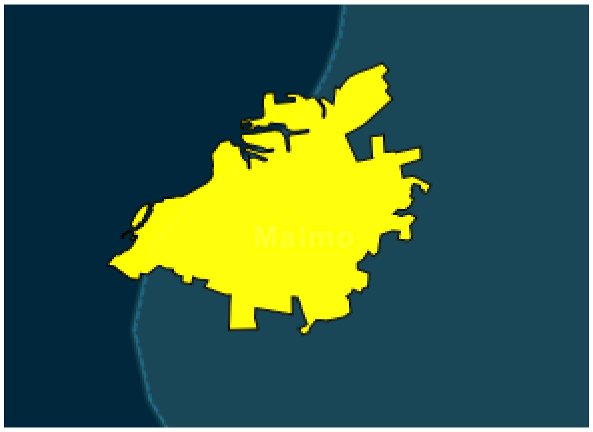


Figure 27: Malmö with holes removed

As with parks and fields, rivers and other geographical delimiters like forces the city to be divided in multiple polygons. By *buffering* the city boundaries, a more coherent area can be created. Buffering extends the borders of an area, hopefully eliminating any unwanted space. Figure 28 and Figure 29 show the difference before and after buffering.

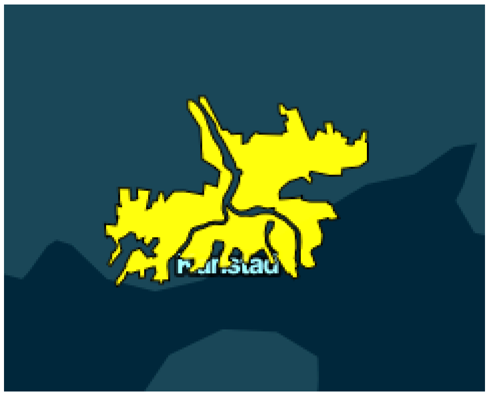


Figure 28: Karlstad with no buffering

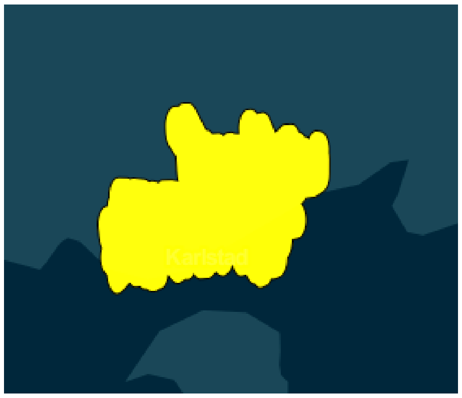


Figure 29: Karlstad with buffering

The operation inevitably leads to a larger area than necessary being covered – some forest parts outside the city boundaries as well as parts of the water are now part of the “city”. It was deemed that this side effect is inevitable and the gains in usability are more important than the losses in exactness of the caching.

Buffering was done by Quinn and Gahegan – several different buffer values were used depending on the type of object being buffered, as could be seen in Figure 4. In this study, the buffer range is constant since only one type of object (city) is being buffered. The buffer range was chosen so that waterways and other delimiters within city bounds are well and fully covered, removing unwanted spaces from the cache output.

5.4 Scale Ranking

So far, the heuristic analysis has only dealt with the two-dimensional attributes of the tiles – in other words the geographical positions on a map. Each position may have different popularities at different scale levels however, which needs to be reflected in the ranking.

Tile rankings for each zoom level are derived the data presented in Table 4: *Scale levels ordered by number of requests*, as presented in chapter 4. Since the populated analysis was done on levels 6 – 14, only those levels are used. The normalized ranking values for the heuristical model are presented in Table 7.

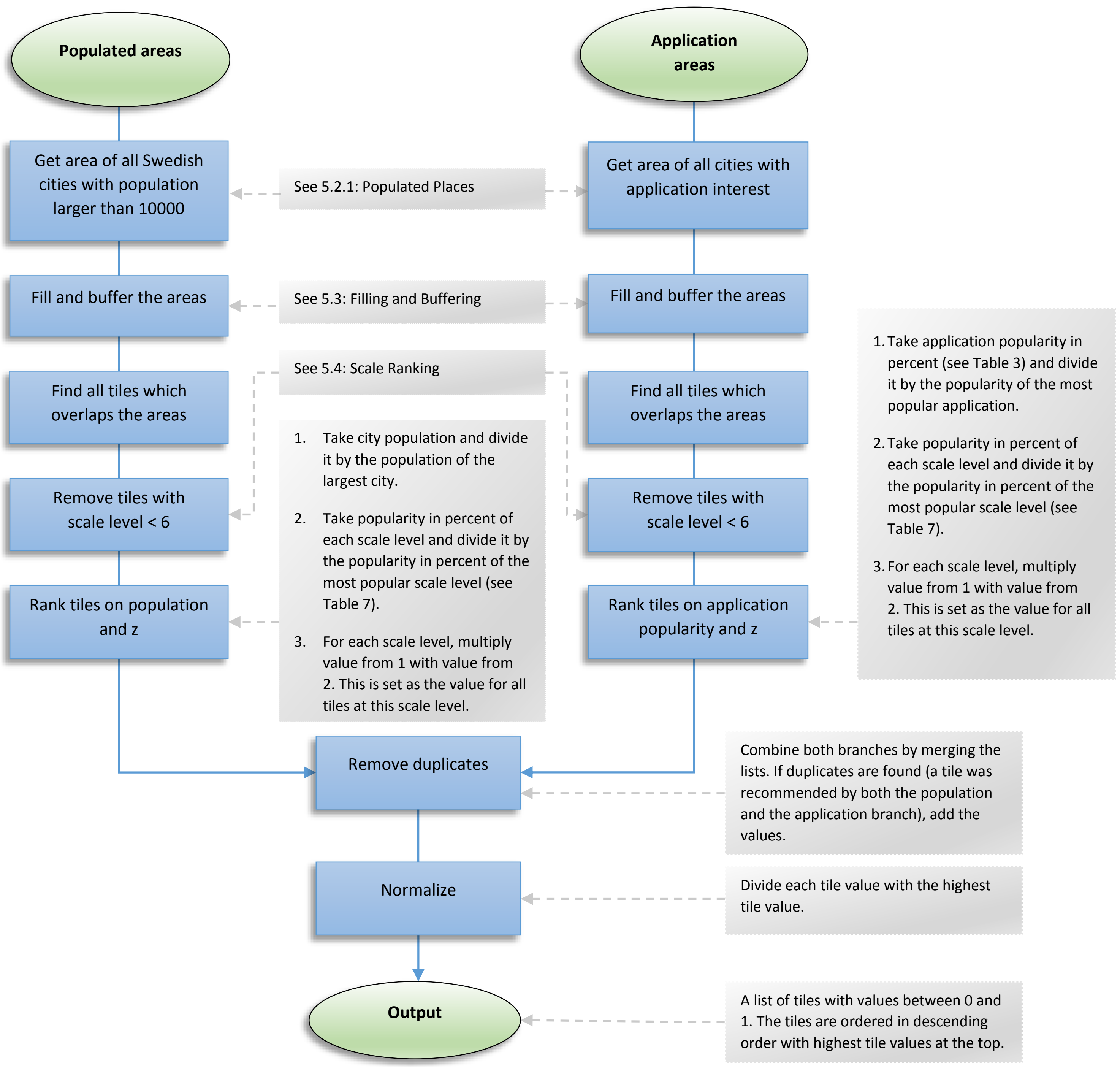
Scale level	Requests	Normalized ranking value
13	295804	1.000
11	280009	0.947
12	272919	0.923
14	202802	0.686
8	199882	0.676
10	192903	0.652
9	164353	0.556
7	135461	0.458
6	79116	0.268

Table 7: Normalized scale rankings for the heuristical model

In order to ease future computations, normalization to the interval 0-1 is performed. Normalization is done by dividing all values with the highest value, resulting in all values being located between 0 and 1 and the relative proportions kept intact.

5.5 Resulting Heuristic Model

The final heuristic model is presented in Figure 30.



Populated areas
Application areas
Get area of all Swedish cities with population larger than 10000
See 5.2.1: Populated Places
Get area of all cities with application interest
Fill and buffer the areas
See 5.3: Filling and Buffering
Fill and buffer the areas
See 5.4: Scale Ranking
Find all tiles which overlaps the areas
Find all tiles which overlaps the areas
1. Take city population and divide it by the population of the largest city.
2. Take popularity in percent of each scale level and divide it by the popularity in percent of the most popular scale level (see Table 7).
3. For each scale level, multiply value from 1 with value from 2. This is set as the value for all tiles at this scale level.
1. Take application popularity in percent (see Table 3) and divide it by the popularity of the most popular application.
2. Take popularity in percent of each scale level and divide it by the popularity in percent of the most popular scale level (see Table 7).
3. For each scale level, multiply value from 1 with value from 2. This is set as the value for all tiles at this scale level.
Remove tiles with scale level < 6
Remove tiles with scale level < 6
Rank tiles on population and z
Rank tiles on application popularity and z
Remove duplicates
Combine both branches by merging the lists. If duplicates are found (a tile was recommended by both the population and the application branch), add the values.
Normalize
Divide each tile value with the highest tile value.
Output
A list of tiles with values between 0 and 1. The tiles are ordered in descending order with highest tile values at the top.

Figure 30: The resulting heuristic model

It should be noted that the output produced by the model assumes that further tile accesses will follow the same patterns. If the access patterns are suspected to have changed, the model should be run again with updated domain information and provide a new cache recommendation.

As can be seen in the “Remove duplicates” step, merging of the different branches is done by simply adding the values. The addition is followed by a normalization to the interval 0-1 - the same operation that was used on the scale levels. Those two steps assures that both branches are given equal importance, and also that the relative values between tiles are kept intact.

As described in the figure, the output of the model is an ordered list which keeps the most interesting tiles at the top. When deciding a tile cache, the *cache limit* (number of tiles to cache) is first decided. This number of tiles are then picked from the list, starting from the top. It should be noted that both the type of output and the cache selection process is the same for all following models – all of them output an ordered list tile ranking with values between 0 and 1 from where the number of tiles equal to the cache limit is selected.

Example output

Figure 31 shows an example output from the heuristical model. Population areas and numbers data were obtained from Lantmäteriet, and the application areas are derived from Table 3. Using this as input, the total number of tiles in the output was 469179. For more tiles in the cache output, a lower population limit should be chosen or more application areas specified.

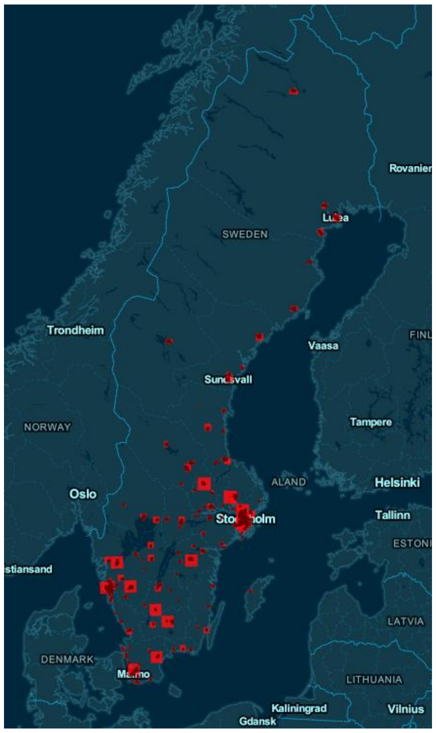


Figure 31: Sample output from the heuristical model

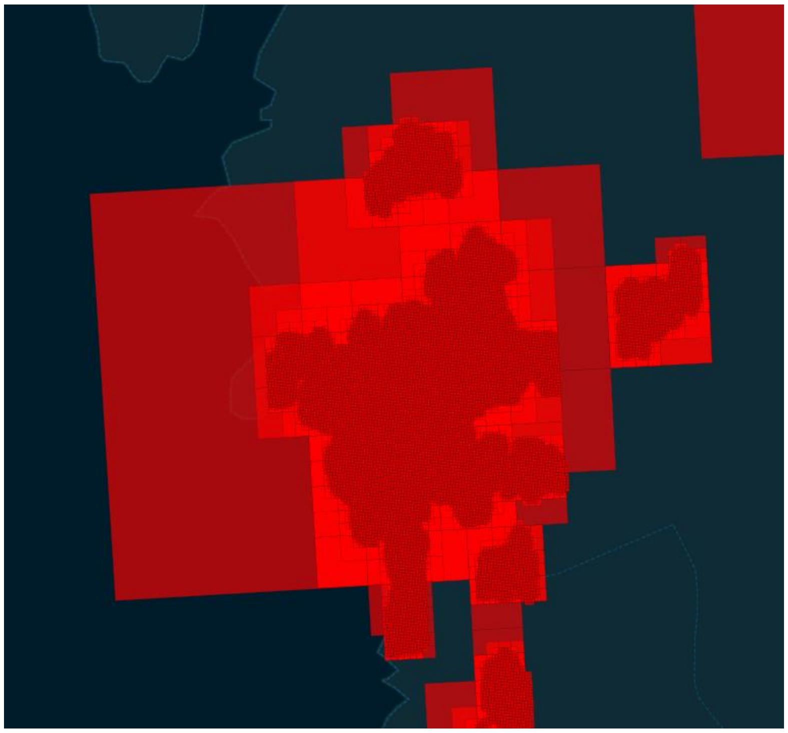
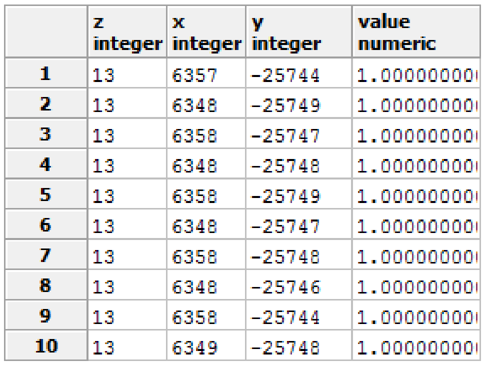


Figure 32: Close-up view of the cached tiles covering Gothenburg and surrounding areas

Also, Figure 32 shows a closer view of the tile cache over Gothenburg. It is noticeable how the precision increases with an increased scale level. For reference, the 10 most popular tiles in the output are shown in Figure 33. As expected, all tiles are from scale level 13. They all have a value of 1.0 meaning that they were from the predicted most popular area, which in this case is Stockholm.



	z integer	x integer	y integer	value numeric
1	13	6357	-25744	1.000000000
2	13	6348	-25749	1.000000000
3	13	6358	-25747	1.000000000
4	13	6348	-25748	1.000000000
5	13	6358	-25749	1.000000000
6	13	6348	-25747	1.000000000
7	13	6358	-25748	1.000000000
8	13	6348	-25746	1.000000000
9	13	6358	-25744	1.000000000
10	13	6349	-25748	1.000000000

Figure 33: The ten most popular tiles using the heuristical model

6 Descriptive Analysis

In this section, two descriptive algorithms are implemented and compared: The trivial model, where past request frequency is used directly as a way of predicting future accesses, and the simplified model proposed by Garcia et al.

6.1 Trivial Model

Early in the study, the idea that letting the request frequency directly decide a tile ranking emerged. Such a solution would simulate an on-demand tile caching server that has been running for a while, instead of waiting for data we simply inject the accesses from a previous time period.

The *trivial model* originates from the assumption that user patterns are consistent over time. Assuming this, the trivial solution to the problem is to simply order tiles based on past popularity. This is done by examining the training data and counting all the tile accesses made to each tile. The values are then normalized to the interval 0-1, where the most popular tile has the value 1 and tiles with no hits get the value 0.

Using the same 7-day data set as in the previous sections, an example cache recommendation can be made. The training data contained roughly 900000 unique requests, which is the length of the cache recommendation. A visualization of the cache recommendation is shown in Figure 34. Each cached tile is shown in a low-opacity red, leading to solid colors where user interest is high:

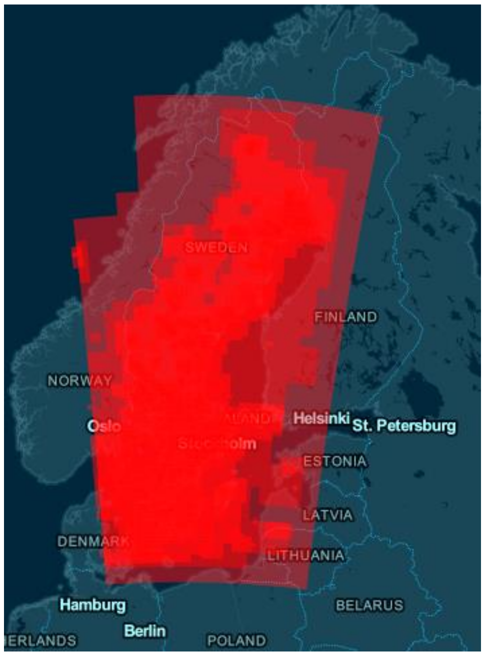
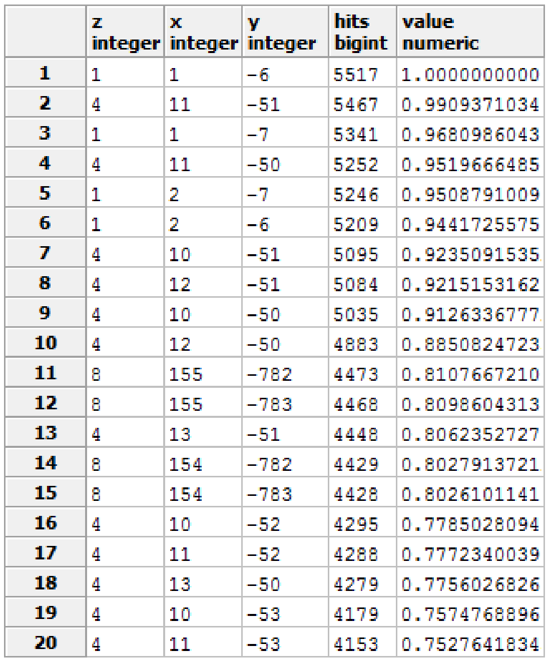


Figure 34:Example output from the trivial model, scale level > 5



	z integer	x integer	y integer	hits bigint	value numeric
1	1	1	-6	5517	1.0000000000
2	4	11	-51	5467	0.9909371034
3	1	1	-7	5341	0.9680986043
4	4	11	-50	5252	0.9519666485
5	1	2	-7	5246	0.9508791009
6	1	2	-6	5209	0.9441725575
7	4	10	-51	5095	0.9235091535
8	4	12	-51	5084	0.9215153162
9	4	10	-50	5035	0.9126336777
10	4	12	-50	4883	0.8850824723
11	8	155	-782	4473	0.8107667210
12	8	155	-783	4468	0.8098604313
13	4	13	-51	4448	0.8062352727
14	8	154	-782	4429	0.8027913721
15	8	154	-783	4428	0.8026101141
16	4	10	-52	4295	0.7785028094
17	4	11	-52	4288	0.7772340039
18	4	13	-50	4279	0.7756026826
19	4	10	-53	4179	0.7574768896
20	4	11	-53	4153	0.7527641834

Figure 35:The 20 most popular tiles using the trivial model

As an example of the tile recommendation, the 20 most popular tiles are shown in Figure 35. It is noticeable how a very small portion of the tiles receive many hits – the values in the column "value" decrease quite quickly. For an example, consider the 20th most popular tile is about 75% as popular as the most popular tile.

The detail and accuracy of the model is dependent on the size of the training data. We cannot rank tiles we do not have any information on. For example, in order for the model to recommend a cache of one million tiles, at least one million unique tile hits would have to be present in the training data. This is one of the drawbacks of this model – large caches demand large amounts of training data as no other assumptions are made regarding tile priority. Here, a combination with the heuristic predictive model previously created might come in handy, which is further examined in the evaluation chapter.

6.2 Simplified Model

The simplified model, as defined by Garcia et al., is a way of estimating the popularity of individual tiles by only using tile data from one scale level. Tile popularity for all tiles are then calculated from this level, which is shown below.

In order to come up with a cache recommendation a *prediction scale level* has to be chosen. This is essentially one of the scale levels available in the domain (in Kartena's case between 0 and 14), and determines the level from which the statistics for all levels are obtained. Popularity values for all scale levels can then be created using the following logic:

Let *s* denote the prediction scale level. Ranking values for all tiles are then be obtained by using the following logic:

- Tiles with scale *z* = *s* get the same ranking value as the corresponding tile value in the trivial solution.
- Tiles with scale *z* > *s* get the same ranking value as the tile in *s* which covers the tile in *z*'s area.
- Tiles with scale *z* < *s* get a ranking value by calculating the average values of the four tiles in scale level *(z + 1)* covering the same area.

An example of the behavior is shown in Figure 36. The prediction level's values are derived directly from the trivial model. Tiles below the prediction level (larger scale value) gets the same value as the tile above, and for each additional level the procedure is repeated. Tiles above (smaller scale value) are averaged from the values below, and again the process is repeated for each additional level.

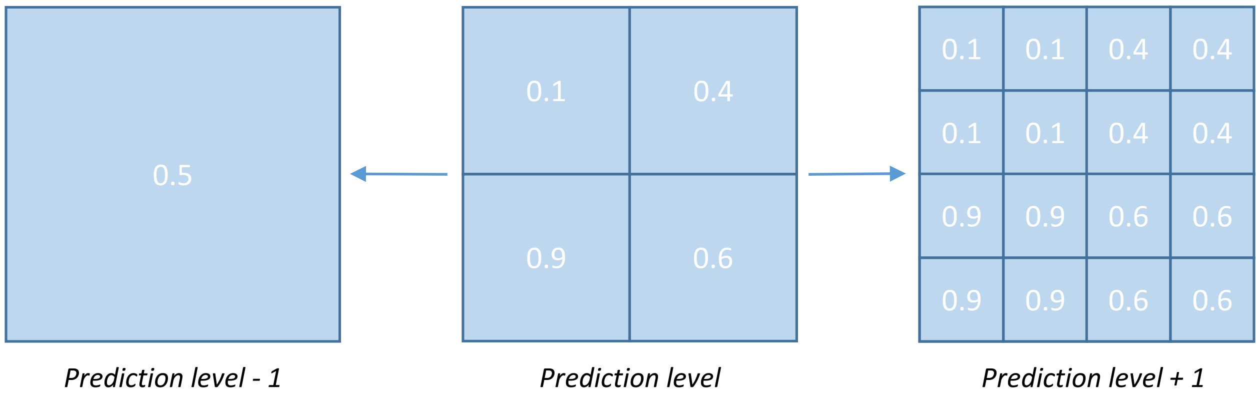


Figure 36: Simplified model, example behavior

Using this behavior, the algorithm is able to "guess" the probability of all tiles. As for the trivial model, the output is an ordered list where the most popular tiles are located at the top. The main downside of the algorithm is that many tiles will share the same ranking value – because of the many propagations of the prediction level's tile values to higher levels.

According to the authors, the simplifications allow for a less resource-intensive solution as only data from the prediction level is used – all other levels can be ignored. However, the number of calculations gets higher with larger prediction levels, as more averaging-operations are needed. While a speed comparison is outside the aim of the study, this is still a factor which affects the study (as evident in the evaluation chapter).

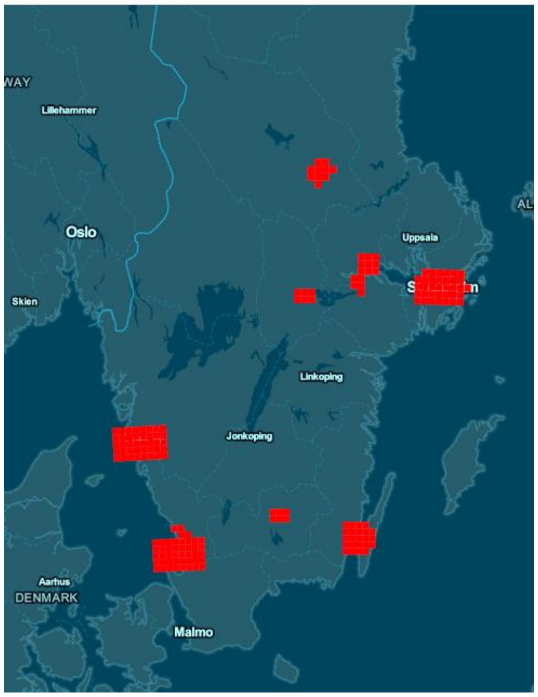
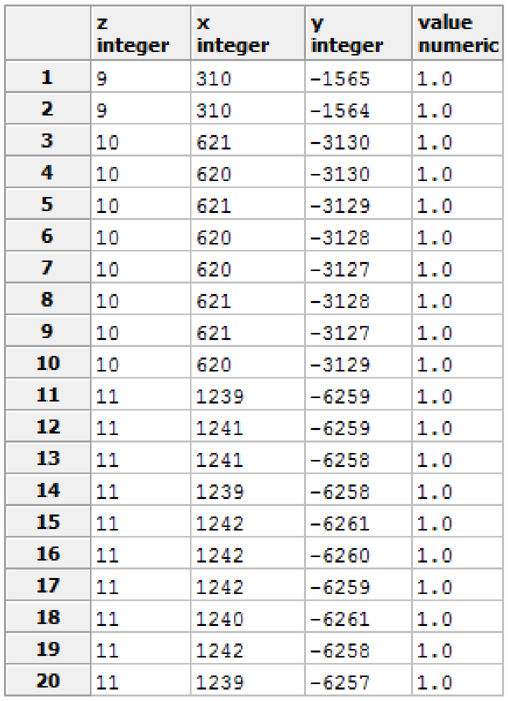


Figure 37: Example of simplified model cache recommendation using scale level 9 as the prediction source



	z integer	x integer	y integer	value numeric
1	9	310	-1565	1.0
2	9	310	-1564	1.0
3	10	621	-3130	1.0
4	10	620	-3130	1.0
5	10	621	-3129	1.0
6	10	620	-3128	1.0
7	10	620	-3127	1.0
8	10	621	-3128	1.0
9	10	621	-3127	1.0
10	10	620	-3129	1.0
11	11	1239	-6259	1.0
12	11	1241	-6259	1.0
13	11	1241	-6258	1.0
14	11	1239	-6258	1.0
15	11	1242	-6261	1.0
16	11	1242	-6260	1.0
17	11	1242	-6259	1.0
18	11	1240	-6261	1.0
19	11	1242	-6258	1.0
20	11	1239	-6257	1.0

Figure 38: The 20 most popular tiles using the simplified model

Figure 37 shows an example of the simplified model's output. It is noticeable that the model results in a quite few number of areas being cached deeply, compared to the trivial model where the cached tiles were more scattered. The result looks more like the output from the heuristic model with very few areas being cached. Also, Figure 38 shows an example of the tile ranking. In this particular training data, the two most popular tiles in level 9 got the same number of hits leading to both having the tile value 1.0. All tiles beneath get the same tile value, as evident in the eight tiles of level 10 and the 32 tiles of level 11 (where only 10 are showed due to the limited size of the table).

Because of the estimating nature of the algorithm, quite small amounts of training data is required to construct a cache compared to the trivial model. While the ranking values in the output may be inexact and not as fine grained as for the trivial model, the hope is that the simplified model provides a suitable alternative when only limited amounts of training data is available.

7 Evaluation

When evaluating the models, a metric reflecting the performance has to be decided on. In existing literature, *hit ratio* is commonly used to evaluate the performance of tile caching methods. In this context, hit ratio is defined as the percentage of accesses in a test set which lead to cached tiles. While there are other metrics that may be important, achieving a good hit ratio is the purpose of the study and the area that have the most impact on business benefits at Kartena.

In order for the evaluation to carry weight, the results must be set in relation to other values. More specifically, what are "good" hit ratios? Garcia et al., when using the mean of normalized access frequencies as threshold, obtained hit ratios of up to 96%. The conclusion was that this is a high number, and that accurate predictions are possible. During discussions, Kartena stated that a 95% hit ratio is a good starting point for a lower hit ratio limit. This number might be adjusted at later stages, but for initial attempts it was deemed to be sufficient. While a high hit ratio as possible is wanted, 95% is selected as the lower limit for the introduction of an on-demand solution at Kartena.

The following sections evaluate the models. First, individual experiments on the three models are performed, which are followed by a combined comparison.

7.1 Individual Comparison

The individual comparison is a first step in determining the performance in a real setting. The models are tested separately against 7 days of access logs, which contain roughly two million tile accesses. The trivial model and the simplified model require training data, which was taken from a different set also containing 7 days. As with the heat map analysis, 7 days was considered a suitable length – the data sets should compensate for the fact that the user behavior might be different on the weekends.

As a fair comparison has to be made, all cache recommendations should contain the same number of tiles. Using the current training data set, the trivial model provides the shortest cache recommendation (471313 tiles) which makes it the limiting factor. For the following three tests, the output limit of 471313 tiles, or 0. 48612% of all possible tiles is therefore chosen.

7.1.1 Heuristical Model

The predictions, which as mentioned originates mainly from populated areas and WMS areas of interest, should ideally be able to produce decent results on its own. The results are shown in Table 8:

Cached tiles (%)	Hits	Hits (%)
0. 486%	1228612	67.96

Table 8: Heuristical model - test results

It is clear that the model captures the majority of user requests. However, the results are far from the goal of 95% - it is not enough to build a cache by using only the heuristical model.

It should be noted that the tile cache of 0.486% is pretty small. By buffering larger areas, lowering the population threshold and allowing for more zoom levels the tile count would increase. This is further discussed in the discussion chapter.

7.1.2 Trivial Model

If user behavior is consistent over time, the trivial model is suspected to provide accurate cache predictions. No evaluations of similar models were found in existing literature.

Cached tiles (%)	Hits	Hits (%)
0.486%	2186333	94.59

Table 9: Trivial model - test results

As evident by the table, the trivial model provides good results. A more detailed comparison is done in the combined comparison later in this chapter. Perhaps the model can consistently provide results over 95% when more training data is used.

7.1.3 Garcia et al.'s Simplified Model

When evaluating the simplified model, it is necessary to decide on a prediction scale which yields the most accurate results. A subset of different scale levels and their performance is listed in Table 10.

Prediction scale	Hits	Hits (%)
0	591831	25.61

1	576679	24.95
2	614455	26.58
3	491410	21.26
4	14577	0.631
5	97078	4.200
6	489730	21.19
7	535955	23.19
8	732840	31.71
9	907913	39.28

Table 10: Simplified model test results, 0.49% cached tiles

During experiments, calculating the levels got increasingly more time consuming for higher prediction scale. The reason is that for higher levels, more averaging-operations have to be performed as the number of tiles in lower scale levels also increase. Level 9 alone took several hours to complete, and more levels would mean exponentially longer calculating times. The combined experiment would simply take too long to complete. This is the reason why only levels 0-9 were tested.

Scale level 9 has the best performance, even if the results clearly are not as good as for the trivial model. The results vary a lot across prediction scale levels however, the low performance of levels 4 and 5 is somewhat surprising. When allowing for a higher cache output (1% instead of 0.49%), more coherent results appear:

Prediction scale	**Hits**	**Hits (%)**
0	1043612	45.15
1	962661	41.65
2	1076666	46.58
3	875424	37.88
4	600849	26.00
5	579040	25.05
6	665946	28.81
7	591808	25.60
8	951752	41.18
9	1131956	48.97

Table 11: Simplified model test results, 1% cached tiles

Following the results of Table 10 and Table 11, level 9 should be used as the prediction source. Higher hit ratios might be possible if higher scale levels where used.

The above results are *roughly* in the same area as the tests performed by Garcia et al. While their upper hit ratios was up in the 95% range, the number of tiles was bigger. When the authors used a cache of 1%, the results were in the area of 5% - 60%. No test on all scale levels was performed, which is unfortunate as this would allow for a more fair comparison.

7.2 Combined Comparison

As stated in the purpose description, the solution is based on a combination of heuristical predictions and descriptive statistics. From the three individual models, two combined algorithms are obtained.

Algorithm A consists of the heuristic model plus the trivial model. Given the good performance of the trivial model and the relatively bad performance of the heuristical model, the latter placed "after" the trivial model in the ranking. This means that the trivial model caches as many tiles at it can and if the cache limit is not reached, the heuristical model fills in the rest. This way, the trivial model is given more importance and the heuristical model kicks in when needed. Duplicates are removed by adding the tile values, followed by a normalization to the interval 0-1 (the same procedure as for the heuristical model, recall chapter 5 and Figure 30 where the two branches are added followed by a normalization).

The extent of which the two models contribute to the cache recommendation is dependant on the training data. Currently, the trivial model needs about 1-2 weeks of data to completely cover the cache limit of 0.486%. Using one day of training data, roughly 1/4th of the cache limit is covered, the rest is left to the heuristical model.

The tile value v_A for the tile $T\{x, y, z\}$ before normalization is expressed as:

$$v_A\{T(x, y, z)\} = v_{trivial}(x, y, z) + v_{heuristical}(x, y, z) \times C, \qquad 0 \leq C \leq 1$$

Where $v_{trivial}(x, y, z)$ is its value in the trivial model $v_{heuristical}(x, y, z)$ its value in the heuristical model and C the value of the least popular tile in the trivial model. As C is always between 0 and 1, multiplying with C ensures that all heuristical values are smaller than the trivial model's, placing them after the trivial model in the final cache recommendation.

Algorithm B consists of the heuristic model plus the simplified model using level 9 as the prediction source. Since the algorithms were roughly the same in terms of performance, both algorithms are given the same importance when combining them. Also, as they both completely cover the cache limit of 0.486% on their own, placing either one after the other will completely hide the other. As with Algorithm A, tiles are combined by adding the output values from both models and normalizing the results.

For Algorithm B, the tile value v_B for the tile $T\{x, y, z\}$ before normalization is given by:

$$v_B\{T(x, y, z)\} = v_{simplified}(x, y, z) + v_{heuristical}(x, y, z)$$

Where $v_{simplified}(x, y, z)$ is its value in the simplified model and $v_{heuristical}(x, y, z)$ its value in the heuristical model. In contrast to Algorithm A, both models are given the same importance which makes further adjustments unnecessary.

7.3 Experiment Design

The experiment should be able to decide which, if any, of the algorithms that is most suitable to use in an on-demand setting. As before, the target hit ratio of 95% is used, ideally the algorithms should be able to consistently provide hit ratios above this target. The outline of the experiment is presented below:

1. Four different sets are used to train the algorithms, covering 1, 7, 14 and 28 days respectively. Both algorithms are trained using each training set, and each algorithm now provide a cache recommendation.
2. Another 28 days of access logs are used to test the algorithms. As with the training data, the testing data is divided in four sets, consisting of 1, 7, 14 and 28 days each.
3. For each algorithm and all combinations training sets - testing sets, the number of tiles in the test set also found in the cache recommendation is divided by the total number of requests made in the test set. A cache hit ratio is now obtained, which is a value between 0 (no cache hits) and 1 (only cache hits).

Since all combinations are tested, 16 different cache hit ratios are obtained for each algorithm. This way, some variation in the data sets is obtained: shorter periods of training data is tested against both shorter and longer periods and vice versa. Because of limitations in data availability, 28 days was chosen as the maximum number of access logs available for both training and testing data.

Baseline
The baseline of the experiment is the "cache all" approach, meaning that 100% of the tiles in the RT 90 region are cached. This always gives a cache hit ratio of 1 since all tile requests lead to already cached tiles. Similar hit ratios are not realistic, but the optimal goal.

7.3.1 Data Sets and Output Limits

The data sets and their dates are shown in Table 12 and Table 13:

Data set	**Dates**
Training data 1	2013-03-06
Training data 2	2013-03-07 to 2013-03-13
Training data 3	2013-03-07 to 2013-03-20
Training data 4	2013-03-07 to 2013-04-03

Table 12: Training data sets

Data set	**Dates**
Test data 1	2013-05-04
Test data 2	2013-04-06 to 2013-04-12
Test data 3	2013-04-06 to 2013-04-19
Test data 4	2013-04-06 to 2013-05-03

Table 13: Test data sets

An effort was put into having both training and test sets of different length. This is why training and test set 1 only consist of one day of training data, in contrast to training and test sets 4 which cover four weeks. During the course of the study, only a couple of months' worth of access logs where available which is the reason that the dates of sets 2, 3 and 4 overlap. Ideally, no overlaps should be present – "fresh" data sets should be used for each ratio calculated.

When comparing Algorithm A and Algorithm B, it is important that the same amount of tiles is used in both comparisons, in other words an "output limit" must exist. In the individual comparisons, the trivial model was the limiting model as it provided the shortest output given 7 days of sample data. In order to be able to connect the combined experiment to the previous experiments, the same limit is chosen. The limit is therefore set to 469179 tiles, or 0.48% of all tiles. In other words, all cache recommendations are cut off so that only 0.48% of all tiles are used.

7.3.2 Results

The hit-ratio results of the experiments are shown in Table 14 and Table 15:

Algorithm A	Test data 1	Test data 2	Test data 3	Test data 4
Training data 1	0.9637	0.9564	0.9566	0.9590
Training data 2	0.9852	0.9835	0.9822	0.9829
Training data 3	0.9912	0.9907	0.9904	0.9904
Training data 4	0.9929	0.9909	0.9908	0.9914

Table 14: Results of Algorithm A

Algorithm B	Test data 1	Test data 2	Test data 3	Test data 4
Training data 1	0.5798	0.5498	0.5530	0.7490
Training data 2	0.5601	0.5544	0.5592	0.5597
Training data 3	0.5723	0.5628	0.5697	0.5691
Training data 4	0.6094	0.5985	0.6059	0.6050

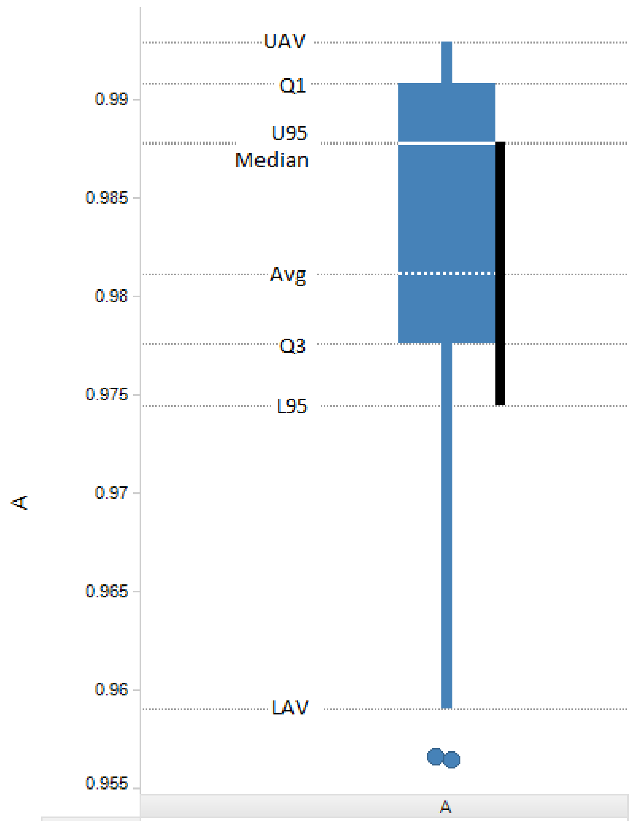
Table 15: Results of Algorithm B

It is evident that sufficiently high hit ratios can be obtained. Algorithm A, which consisted of the trivial model and the heuristical model, is a suitable candidate for creating a tile cache recommendation. It consistently provides hit ratios in the area of 98-99%, all while caching a very small portion of the total tiles (less than 0.5%). The combined approach clearly provides better values than the individual algorithms.

Algorithm B does not perform very well. Hit ratios are typically in the range 55-60%. Which is better than for the simplified model alone but actually worse than for the heuristical model. However, unusually high hit ratios were found using training set 1 and test set 4. Despite repeated investigations, no cause of the behavior could be found. The same combination of training set and test set in algorithm A did not yield any unusual results, which would be expected if the sets were unusual in any way. With the exception of this behavior, hit ratios consistently increase with longer training data sets. Therefore, the conclusion that longer training data sets are better can be drawn.

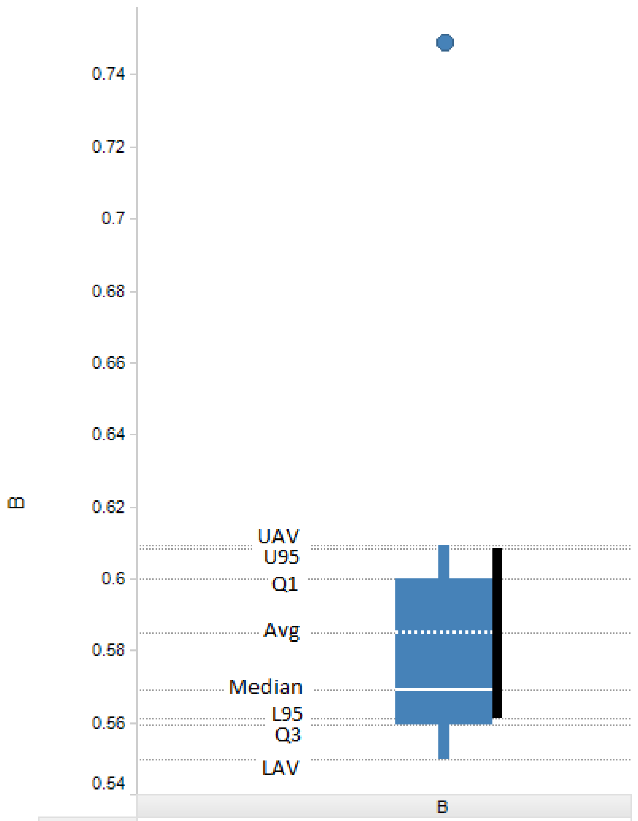
The *box plot* is a way of visualizing groups of numerical data using various statistical concepts such as mean, median and quartiles. Using box plots, it is possible to visualize many characteristics of data sets while using a relatively small amount of space. Also, it is possible to visualize several data sets without knowing the underlying distribution, which is the case of the experiment (Liu, 2008).

Figure 39 and Figure 40 show schematic box plots for algorithm A and algorithm B respectively, and Figure 41 shows both plots in a single view. Various statistical properties are shown in the graph- outliers are shown as dots and the black bar represents the range of the 95% C.I. The quartiles, mean, median and adjacent values are also shown.



	A
Avg	0.981147
Median	0.987785
Q1	0.977562
Q3	0.990833
UAV	0.992926
LAV	0.959014
U95	0.987877
L95	0.974416

Figure 39: Box plot (Algorithm A)



	B
Avg	0.584849
Median	0.569416
Q1	0.559597
Q3	0.600126
UAV	0.609347
LAV	0.549809
U95	0.608449
L95	0.561249

Figure 40: Box plot (Algorithm B)

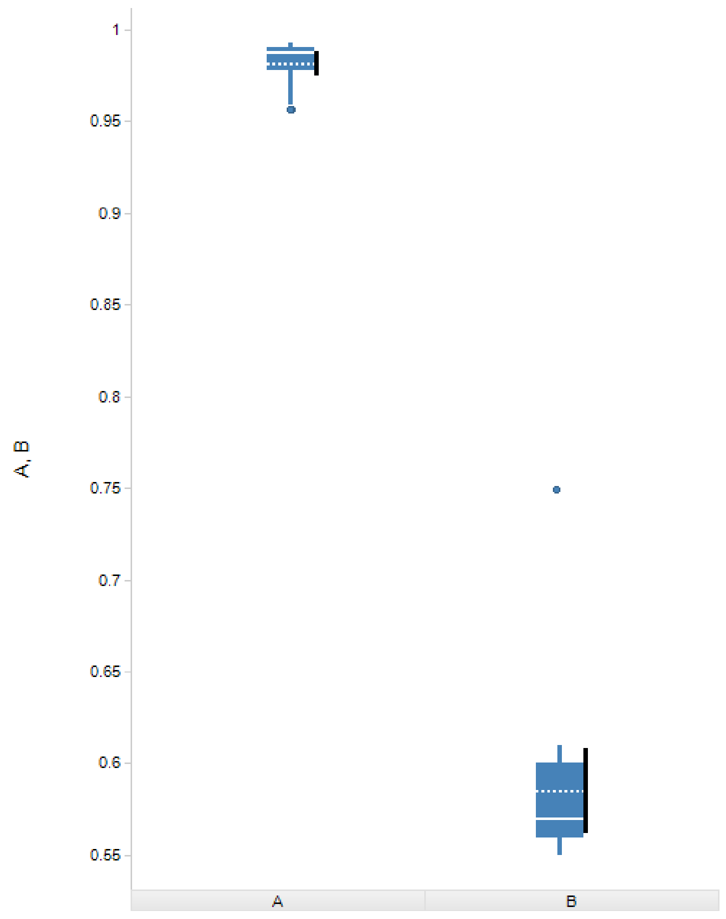


Figure 41: Box plot (Algorithm A and Algorithm B)

The performance differences between the two algorithms are clear - Algorithm A consistently provides values larger than 95%. The box plot of algorithm B is also slightly less dense and fluctuates a bit more, maybe indicating less reliable results. The relatively poor performance of training set 1 combined with the quick improvements due to the larger training sets resulted in Algorithm A:s plot being slightly left-skewed.

For algorithm A, the lower limit of the 95% confidence interval is 97.44%. It is therefore likely that further samples would perform well above the 95% limit previously set.

8 Discussion

This section presents an extended discussion of the experiment results, the possible applications of the results and the validity threats.

8.1 Evaluation of the Combined Experiment

Algorithm A consistently provides better results than Algorithm B. The trivial model's superiority to the simplified model is clearly visible in the result. It is also evident that the variance of B is high compared to A.

Neither of the individual models provided adequate hit ratios on their own. The trivial model's advantage over the simplified model is evident in the combined experiment – overall the simplified model had a pretty bad performance. It is noticeable how the combined algorithms (trivial and heuristics, and simplified and heuristics) perform better, which motivates the combination of several models.

All models could benefit from a higher cache limit. For example consider the trivial model - while good results (just over 94.5%) were obtained using a low cache limit (0.48%), results over 95% are probably possible if the cache limit is increased. In a similar way, the heuristical model would probably benefit from a higher population threshold. At the same time the "quality" of a tile ranking is something that could be investigated further, especially when increasing the cache limit. When selecting tiles to cache, there is a trade-off between cached tiles and the hit ratio they contribute. Adding a few hundred thousand tiles to the cache output will likely increase the hit ratio, the question is if the increase is enough to motivate the extra tiles.

8.2 Application of Result

Provided that some modifications to Kartena's rendering procedures are made, the tile set recommended by Algorithm A can be used as the initial cache in an on-demand architecture. The experiment indicates that great optimizations can be made in terms of storage and rendering times, while still achieving a good hit-ratio. The exact numbers of saved time and space when rendering are not known as tiles all have different rendering times and take up different amounts of space.

There are solid indications that more training data provides better hit ratios, which should be considered prior to building the cache. The algorithm performs well under varying amounts of training data, with heuristical predictions kicking in when the training data is not sufficient. Also, Algorithm A assumes that the patterns are the same as before. In the event of changes to the domain such as a changed client set, it may be wise to update the training recommendation in order for the cache to be relevant.

At some point, the number of cached tiles should be decided. In the experiment, a cache of 0.48% was used which was enough to obtain good hit ratios. When using a higher number, the hit ratio will most certainly be higher. However, the cache is limited by the amount of unique tiles in the training data. Tiles with no hits will never be recommended by the trivial model – and the heuristical model is limited by the city data.

The result also extends the current state of the research area. For Quinn and Gahegan, no quantitative user data was available, instead ready-made heat maps were utilized. This is a drawback of their study. For example, evaluation is performed by overlaying the cache recommendation on heat maps, which is meant to provide some indications on the model's accuracy. Such tests are probably prone to errors, which is something that can and should be addressed. This report presents a more exact evaluation using real-world quantitative access data, which could be further built upon.

8.3 Threats to Validity

This section presents some of the drawbacks that have been identified, along with how they were handled.

Simplified model

Due to computational limitations, all levels in the simplified model were not tested. However, it should be noted that the results were in the same area as the results of Garcia et al. It is therefore reasonable to expect similar results for the other scale levels, which would not affect the result in a major way.

Heuristic research
While the 67% hit ratio indicates that decent estimations are made, it is possible that this can be made better. The population threshold of 10000 could and should be further examined and perhaps lowered. However, the increase of cached tiles always comes at a cost, as discussed in the Future work part.

Also, the order of the filling and buffering operations should be reversed. Currently, the buffering will create new "holes" in the cache due to previously separated areas being combined. If the filling operation is performed after instead of before the buffering, this problem is likely to disappear. This drawback was realized after the experiments were done, and while the results in terms of hit ratio would probably be negligible (if the tiles were important, the trivial model would catch them) it should still be considered.

Non-independent data sets
The training sets were not independent, as test set 2 ⊂ test set 3 ⊂ $\text{test set } 4$ and training set 2 ⊂ $\text{training set } 3 \subset \text{training set } 4$. This may affect the result – in test set 4, any unusual activity in the last seven days affecting the result may be "hidden" by the two previous data sets. Longer test sequences were deemed to be more important however, which is the reason the experiment was set up the way it was. The limitations in available data was also a contributing factor – the available access logs covered roughly nine weeks which made the trade-off necessary. Also, all values in Table 14 and Table 15 are examples of possible real-world combinations of training and testing data and are therefore valid.

9 Conclusion

The study investigated the possible gains of implementing of an on-demand caching solution at the company Kartena. This section presents the conclusions made, followed by suggestions for future work.

Background
When developing web map applications, a couple of challenges exist when it comes to serving map information to the user. A typical solution is the tile-based approach, which serves portions of the map data in separate files. The large number of tiles in a typical applications puts constraints on storage and rendering times, as evident at the company Kartena where a rendering session can take several days. Since the majority of tiles are rarely accessed, the goal is to move to a semi-on-demand rendering approach where a subset of the most popular tiles are rendered in advance. Additional tiles are then rendered only when accessed by a user. Typically, this solution leads to less storage space needed and up-front rendering to be done – if a suitable tile subset can be identified.

By creating an algorithm for predicting the most popular tiles, the study set out to address the issues in a suitable way. The goal was to use a combination of heuristic usage patterns and predictive statistics as this had not been done before.

Solution
Two related studies were identified, which offered different takes on similar problems. Quinn and Gahegan analyzed heat maps and identified (to users) interesting geographical areas, in order to predict future tile usage, which. They identified major roads, coast lines, cities and "points of interest" to be interesting to users. Garcia et al. created a model which used tiles access logs to come up with a tile prediction, and created "the simplified model" which offered some optimizations to manage the large number of tiles.

The findings of Quinn and Gahegan were found to not translate particularly well to the domain. Only populated areas translated somewhat well. A heuristical model was created, which addresses the issues and also added things like filling and buffering. The simplified model was implemented, as well as a third model - the trivial model - where past tile statistics directly affected the tile recommendation.

Evaluation
The proposed algorithms were evaluated in a series of experiments, where the algorithms were compared to real world access data. A good algorithm would have as many of the accessed tiles in its recommendation, therefore obtaining a good *hit ratio*.

- The heuristical model did not perform particularly well, with hit ratios of 67.96%. It was not possible to accurately predict user behavior using the heuristical model alone.
- The simplified model as defined by Garcia et al. did not provide satisfactory results, the highest hit ratio found was 48.97%. The performance was about the same as in the study by Garcia et al.
- The trivial model showed great potential and provided the most accurate results, offering hit ratios of 94.59%.

Two combined comparisons were also made, by pairing the heuristical model with the two descriptive models respectively. The heuristical model combined with the trivial model was called Algorithm A and the heuristical model combined with the simplified model was called Algorithm B.

- Algorithm A performed very well, obtaining hit ratios of 95% and upwards.
- Algorithm B performed better than the simplified model individually – offering hit ratios between 70 and 83%.

According to Kartena, the performance of Algorithm A was more than enough to implement an on-demand caching solution. The results were achieved when caching a small (0.48) percent of the total number of tiles, which has the business benefit of shorter rendering times and reduced required storage space. If needed, the cache can be modified – with more cached tiles the hit ratios almost certainly go up. Also, there are strong indications that the algorithm performs better when using longer training sets.

9.1 Future Work

A couple of future areas of improvement have been identified. In this section, suggestions of future areas of improvement are presented.

The number of accesses was shown to be smaller over weekends. Are they also different in nature? It is entirely possible that certain WMS applications offer services that only are used on weekdays. Overall, a more application-specific analysis could be performed. For example, performing individual heat map analyses for the biggest WMS applications might provide new information on how the heuristical cache should be constructed. Also, the default view (not always level 1) of all applications should be identified and prioritized higher in the heuristical model. The trivial solution probably takes care of a lot of this however.

The study did a fair comparison with the different models. However, further work could be done in the evaluation by tailoring the input data for each model: as mentioned in the discussion chapter, increased cache limits could be investigated further. Further evaluations could be made, for example investigating different population thresholds with relation to hit ratio and tiles cache. Also, the heuristical study could be extended with experiments measuring the popularity of all the important geographical areas found by Quinn and Gahegan.

References

Apache, 2012. *Apache HTTP Server Version 1.3*. [online] Available at: <http://httpd.apache.org/docs/1.3/logs.html> [Accessed 05 June 2013].

Boulos, M. N. K., Warren, J., Gong, J., Yue, P., 2010. Web GIS in practice VIII: HTML5 and the canvas element for interactive online mapping. *International Journal of Health Geographics,* 9(14), pp.1-13.

ESRI, 2013. *What is map caching?.* [online] Available at: <http://webhelp.esri.com/arcgisserver/9.3/java/index.htm#what_is_map_caching.htm> [Accessed 08 February 2013].

Fisher, D., 2007. Hotmap: Looking at Geographic Attention. *IEEE transactions on Visualization and Computer Graphics,* 13(6), pp.1184-1191

Garcia, R., De Castro, J.P., Verdu, M.J., Verdu, E., Regueras, L.M., Lopez, P., 2012. *A Descriptive Model for Predicting Popular Areas in a Web Map.* [online] Available at: <http://www.wseas.us/e-library/conferences/2011/Cambridge/AIKED/AIKED-66.pdf> [Accessed 08 February 2013].

Garcia, R., De Castro, J.P., Verdu, M.J., Verdu, E., Regueras, L.M., Lopez, P., 2011. *A Cache Replacement Policy Based on Neural Networks Applied to Web Map Tile Caching*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), pp. 1-7.

Kraak, M. J., 2004. The role of the map in a Web-GIS environment. *Journal of Geographical Information Systems*, 6, p.83–93.

Kraak, M. J., Brown, A., 2002. *Trends in cartography*. [online] Available at: <http://kartoweb.itc.nl/webcartography/webbook/ch02/ch02.htm> [Accessed 08 January 2013].

Lantmäteriet, 2013. *RT 90*. [online] Available at: <http://www.lantmateriet.se/Kartor-och-geografisk-information/GPS-och-geodetisk-matning/Referenssystem/Tvadimensionella-system/RT-90/> [Accessed 05 June 5, 2013].

Liu, 2008. Box plots: use and interpretation. *Transfusion*, 45, p.2279.

Manlai, Y., Chen, C., Liu, H., Lin, H., 2007. A Usability of Web Map Zoom and Pan Functions. *International Journal of Design, 1(1), 15-25.*

MapBox, 2013. *TileMill*. [online] Available at: <http://www.mapbox.com/tilemill/> [Accessed 27 May 2013].

OpenGeo, 2013. *Introduction to PostGIS*. [online] Available at: <http://workshops.opengeo.org/postgis-intro/geometries.html> [Accessed 27 May 2013].

OGS, 2006. *OpenGIS Web Map Server Implementation Specification*. [online] Available at: <http://portal.opengeospatial.org/files/?artifact_id=14416> [Accessed 05 June 2013].

Pavlenko, 2013. *What is mapnik?*. [online] Available at: <http://mapnik.org/faq/> [Accessed 11 June 2013].

PostGIS, 2013. *About PostGIS.* [online] Available at: <http://postgis.net> [Accessed 23 May 2013].

PostgreSQL, 2013. *About.* [online] Available at: <http://www.postgresql.org/about/> [Accessed 23 May 2013].

Python Software Foundation, 2013. *About Python.* [online] Available at: <http://www.python.org/about/> [Accessed 23 May 2013].

PostGIS, 2013. *PostGIS Special Functions Index*. [online] Available at: http://postgis.net/docs/manual-2.0/PostGIS_Special_Functions_Index.html#PostGIS_Aggregate_Functions [Accessed 27 May 2013].

pgAdmin, 2013. *Introduction*. [online] Available at: <http://www.pgadmin.org> [Accessed 27 May 2013].

Quinn, S., Gahegan, M., 2010. A Predictive Model for Frequently Viewed Tiles in a Web Map. *Transactions in GIS,* 14(2), pp.193-216.

Sample, J.T. and Ioup, E., 2010. *Tile-Based Geospatial Information Systems.* New York: Springer.

Soonthornsutee, R., Luenam, P., 2012. Web Log Mining for Improvement of Caching Performance. in: IMECS 2012, *International MultiConference of Engineers and Computer Scientists*. Hong Kong, People's Republic of China 14-16 March 2012.

Spatial Nodes, 2010. *Tileseeder*. [online] Available at: <http://blog.minst.net/2010/09/01/tileseeder> [Accessed 08 February 2013].

Spatial Reference, 1997. *EPSG: 2400*. [online] Available at: <http://spatialreference.org/ref/epsg/2400/> [Accessed 05 June 2013].

Spatial Reference, 2007. *EPSG: 4326*. [online] Available at: <http://spatialreference.org/ref/epsg/wgs-84> [Accessed 10 October 2013].

TileStache, 2013. *TileStache*. [online] Available at: <http://tilestache.org/> [Accessed 27 May 2013].

TileStache, 2013. *Proj4Projection*. [online] Available at: <http://tilestache.org/doc/TileStache.Goodies.Proj4Projection.html> [Accessed 27 May 2013].

Varrazzo, 2010. *psycopg.* [online] Available at: <http://initd.org/psycopg/> [Accessed 23 May 2013].

W3C, 1995. *Logging Control In W3C httpd.* [online] Available at: <http://www.w3.org/Daemon/User/Config/Logging.html> [Accessed 27 May 2013].

W3C, 1999. *Status Code Definitions*. [online] Available at: <http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html> [Accessed 28 May 28, 2013].

w3schools, 2013. *Introduction to XML*. [online] Avilable at: <http://www.w3schools.com/xml/xml_whatis.asp> [Accessed 27 May 2013].

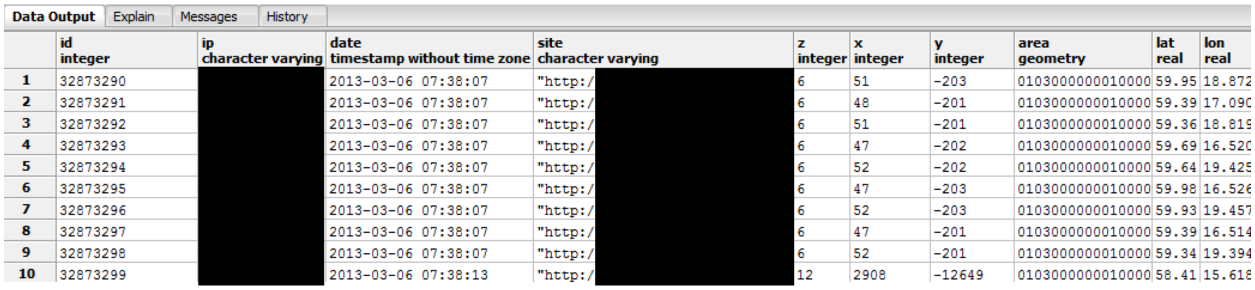
Appendix A: Technical Solution

As stated in the goal description, part of the study is to develop a technical solution to the problem. This chapter describes the different technical decisions made, the technical components used and their motivating factors.

Database

The implementations of all the algorithms rely heavily on PostGIS. PostGIS is a spatial extension of the PostgreSQL relational database system, and adds new spatial data types as well as functions. Using PostGIS, GIS-operations are made available which greatly simplifies certain operations. PostGIS-specific geographical data types are used to represent city data as well as individual tiles. Built-in functions greatly simplifies intersection queries such as "does this tile and this area overlap" and the buffering of areas (PostgreSQL, 2013) (PostGIS, 2013).

Access logs are stored in the database. Figure 42 shows some sample rows from the *"accesslog"*-table (columns *"ip"* and *"site"* censored).



	id integer	ip character varying	date timestamp without time zone	site character varying	z integer	x integer	y integer	area geometry	lat real	lon real
1	32873290	[redacted]	2013-03-06 07:38:07	"http:/ [redacted]	6	51	-203	0103000000010000	59.95	18.872
2	32873291	[redacted]	2013-03-06 07:38:07	"http:/ [redacted]	6	48	-201	0103000000010000	59.39	17.09
3	32873292	[redacted]	2013-03-06 07:38:07	"http:/ [redacted]	6	51	-201	0103000000010000	59.36	18.81
4	32873293	[redacted]	2013-03-06 07:38:07	"http:/ [redacted]	6	47	-202	0103000000010000	59.69	16.52
5	32873294	[redacted]	2013-03-06 07:38:07	"http:/ [redacted]	6	52	-202	0103000000010000	59.64	19.42
6	32873295	[redacted]	2013-03-06 07:38:07	"http:/ [redacted]	6	47	-203	0103000000010000	59.98	16.52
7	32873296	[redacted]	2013-03-06 07:38:07	"http:/ [redacted]	6	52	-203	0103000000010000	59.93	19.457
8	32873297	[redacted]	2013-03-06 07:38:07	"http:/ [redacted]	6	47	-201	0103000000010000	59.39	16.514
9	32873298	[redacted]	2013-03-06 07:38:07	"http:/ [redacted]	6	52	-201	0103000000010000	59.34	19.394
10	32873299	[redacted]	2013-03-06 07:38:13	"http:/ [redacted]	12	2908	-12649	0103000000010000	58.41	15.61

Figure 42: Example rows in the access log table

Using various queries and views, information can be obtained from the *accesslog* table. Certain other tables also exist, such as the "cached"-table where the output of the model(s) are created.

Cities are also stored in the PostGIS database, with the columns name, population and area. Using various PostGIS functions such as ST_Intersects intersections with tiles and the like can be found. Holes are removed using a combination of ST_ExteriorRing (gets the bounds of a polygon) and ST_BuildArea (constructs an area given a polygon), and buffering is done using ST_Buffer (PostGIS, 2013).

Kartena has experience with PostGIS which speaks to its favor. Alternative database systems were available, such as MySQL which has some built-in spatial support. However, the experience argument was deemed very important and is the main reason PostGIS was chosen

Python scripts

By using the python programming language, automated and context-dependable queries are achievable. The database is mainly controlled through a number of python scripts, through the PostgreSQL library psycopg (Varrazzo, 2010). For example, *"logparser.py"* reads accesslogs and fills the database with tile information. The python scripts can then be invoked through Windows batch files.

Even if only shallow knowledge of Python existed beforehand, the language proved to be quick to learn and adapt to. Kartena has developed an extension to TileStache called Proj4Projection (TileStache, 2013), which greatly simplifies the task of transforming from tile coordinates (e.g. 12, 2864, -12838) to geographical coordinates (e.g. lat = 59.28, lon = 15.22). Examples and instructions on how to use this was provided by Kartena, which is the main reason that Python along with Proj4Projection was used in the study.

The Python scripts both read and write to the database, which is symbolized by the double-pointed arrow in Figure 43. Typically, reads are performed in order to obtain some data – the data is then modified and written to a different table.

The scripts read and parse the following types of files:

Access logs
Text files containing information in CLF form. The files are parsed manually by reading each row. Only valid accesses are accepted, i.e. those which have an http status code equal to 200 (W3C, 1999). Also, requests to tiles located outside the RT-90 bounds are ignored.

Heuristics file
Heuristics are expressed in an xml file (w3schools, 2013). Python has built-in support for XML parsing, which makes the format easy to use.

Population cities are denoted using the *"popCity"*-tag. Including a city in the xml document simply means that it will be added to the population ranking. Population and ranking data is then obtained from the database – for debugging purposes only the name is specified in the heuristics file. An example use of the population tag is:

```
<popCity name = "Nynäshamn"></popCity>
```

Client areas are city-based, and are specified using the *"appCity"* tag. This tag has an "app"-attribute specifying which client the tag refers to, a *"name"*-tag specifying the city. The text field contains the value the WMS application's value. It is likely that several client areas share the same city, therefore the values are added when parsed. An example use of the application tag is:

```
<appCity app = "Client 1" name = "Stockholm">0.095184087</appCity>
```

Scale level weights are specified using the *"scale"*-tag. The attribute *"level"* contains the scale, and the text field contains the weighted value. An example use of the scale tag is:

```
<scale level="8">0.6757244662005923</scale>
```

City data
City data is expressed in a separate file using a semicolon-separated unspecified format. Each row contains a city name, population count and a string denoting a polygon in the PostGIS multipolygon-format (OpenGeo, 2013). An example row in the city file is:

```
"Karlskrona";"";35212;"MULTIPOLYGON([polygon data])"
```

9.2 Manual queries

Manual queries are used to both read and write from the database. SQL queries can be performed directly from the pgAdmin application (pgAdmin, 2013). Manual queries were mostly used for debugging and testing different approaches – most of the steps where later automated using Python scripts. Manual queries are shown as a double-pointed arrow in Figure 43, as both inserts and reads are done.

9.3 TileMill

Heat maps, city graphics and other map views are created using TileMill. The program connects directly to the database and displays any geometries found in the specified table. Styling rules are specified which changes how the result looks.

An example schema of the technical components is shown in Figure 43:

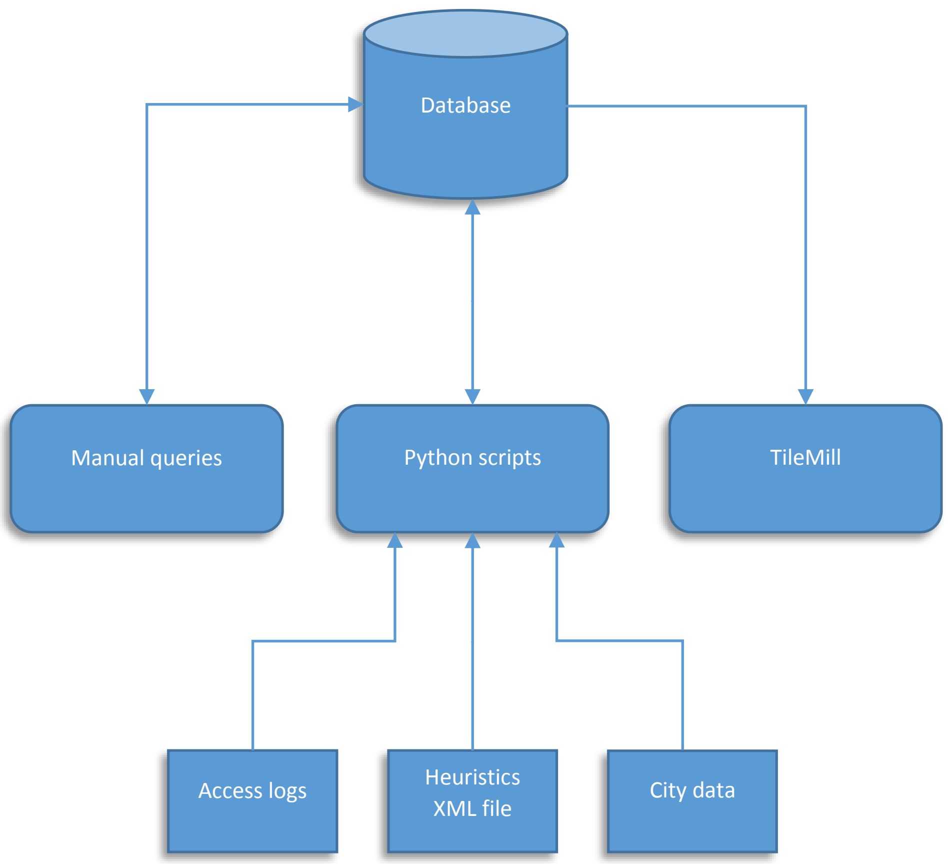


Figure 43: Technical components and their interactions