Chapter 2

Web Map Tile Services for Spatial Data Infrastructures: Management and Optimization

Ricardo García, Juan Pablo de Castro, Elena Verdú,
María Jesús Verdú and Luisa María Regueras

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/46129

1. Introduction

Web mapping has become a popular way of distributing online mapping through the Internet. Multiple services, like the popular Google Maps or Microsoft Bing Maps, allow users to visualize cartography by using a simple Web browser and an Internet connection. However, geographic information is an expensive resource, and for this reason standardization is needed to promote its availability and reuse. In order to standardize this kind of map services, the Open Geospatial Consortium (OGC) developed the Web Map Service (WMS) recommendation [1]. This standard provides a simple HTTP interface for requesting geo-referenced map images from one or more distributed geospatial databases. It was designed for custom maps rendering, enabling clients to request exactly the desired map image. This way, clients can request arbitrary sized map images to the server, superposing multiple layers, covering an arbitrary geographic bounding box, in any supported coordinate reference system or even applying specific styles and background colors.

However, this flexibility reduces the potential to cache map images, because the probability of receiving two exact map requests is very low. Therefore, it forces images to be dynamically generated on the fly each time a request is received. This involves a very time-consuming and computationally-expensive process that negatively affects service scalability and users' Quality of Service (QoS).

A common approach to improve the cachability of requests is to divide the map into a discrete set of images, called tiles, and restrict user requests to that set [2]. Several specifications have been developed to address how cacheable image tiles are advertised from server-side and how a client requests cached image tiles. The Open Source Geospatial Foundation (OSGeo) developed the WMS Tile Caching (usually known as WMS-C) proposal [3]. Later, the OGC released the Web Map Tile Service Standard (WMTS) [4] inspired by the former and other similar initiatives.

Most popular commercial services, like Google Maps, Yahoo Maps or Microsoft Virtual Earth, have already shown that significant performance improvements can be achieved by adopting this methodology, using their custom tiling schemes.

The potential of tiled map services is that map image tiles can be cached at any intermediate location between the client and the server, reducing the latency associated to the image generation process. Tile caches are usually deployed server-side, serving map image tiles concurrently to multiple users. Moreover, many mapping clients, like Google Earth or Nasa World Wind, have embedded caches, which can also reduce network congestion and network delay.

This chapter deals with the algorithms that allow the optimization and management of these tile caches: population strategies (*seeding*), tile pre-fetching and cache replacement policies.

2. Tiling schemes

Maps have been known for a long time only as printed on paper. Those printed cartographic maps were static representations limited to a fixed visualization scale with a certain Level Of Detail (LOD). However, with the development of digital maps, users can enlarge or reduce the visualized area by zooming operations, and the LOD is expected to be updated accordingly.

The adaptation of map content is strongly scale-dependent: A small-scale map contains less detailed information than a large scale map of the same area. The process of reducing the amount of data and adjusting the information to the given scale is called cartographic generalization, and it is usually carried out by the web map server [5].

In order to offer a tiled web map service, the web map server renders the map across a fixed set of scales through progressive generalization. Rendered map images are then divided into tiles, describing a tile pyramid as depicted in Figure 1.

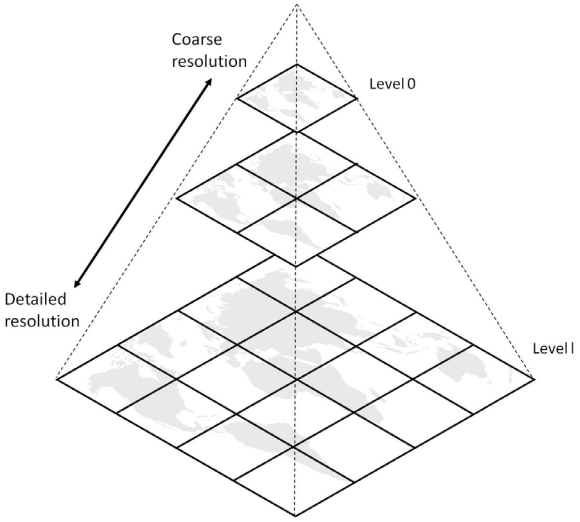


Figure 1. Tile pyramid representation.

For example, Microsoft Bing Maps uses a tiling scheme where the first level allows representing the whole world in four tiles (2x2) of 256x256 pixels. The next level represents the whole world in 16 tiles (4x4) of 256x256 pixels and so on in powers of 4. A comprehensive study on tiling schemes can be found in [2].

2.1. Simplified model

Given the exponential nature of the scale pyramid, the resource consumption to store map tiles results often prohibitive for many providers when the cartography covers a wide geographic area for multiple scales. Consider for example that Google's BigTable, which contains the high-resolution satellite imagery of the world's surface as shown in Google Maps and Google Earth, contained approximately 70 terabytes of data in 2006 [6].

Besides the storage of map tiles, many caching systems also maintain metadata associated to each individual tile, such as the time when it was introduced into the cache, the last access to that object, or the number of times it has been requested. This information can then be used to improve the cache management; for example, when the cache is out of space, the LRU (*Least Recently Used*) replacement policy uses the last access time to discard the least recently used items first.

However, the space required to store the metadata associated to a given tile may only differ by two or three orders of magnitude to the one necessary to store the actual map image object. Therefore, it is not usually feasible to work with the statistics of individual tiles. To alleviate this problem, a simplified model has been proposed by different researchers. This model groups the statistics of adjacent tiles into a single object [7]. A grid is defined so all objects inside the same grid section are combined into a single one. The pyramidal structure of scales is therefore transformed in some way in a prism-like structure with the same number of items in all the scales.

3. Web Map Server workload

In order to deal with this complexity some cache management algorithms have been created. However, the efficiency of the designed algorithms usually depends on the service's workload. Because of this, prior to diving into the details of the cache management policies, a workload characterization of the WMS services need to be shown. Lets take some real-life examples for such characterization: trace files from two different tiled web map services, Cartociudad[1] and IDEE-Base[2], provided by the National Geographic Institute (IGN)[3] of Spain, are presented in this chapter.

Cartociudad is the official cartographic database of the Spanish cities and villages with their streets and roads networks topologically structured, while IDEE-Base allows viewing the Numeric Cartographic Base 1:25,000 and 1:200,000 of the IGN.

Available trace files were filtered to contain only valid web map requests according to the WMS-C recommendation. Traces from Cartociudad comprise a total of 2.369.555 requests

[1] http://www.cartociudad.es
[2] http://www.idee.es
[3] http://www.ign.es/ign/main/index.do?locale=en

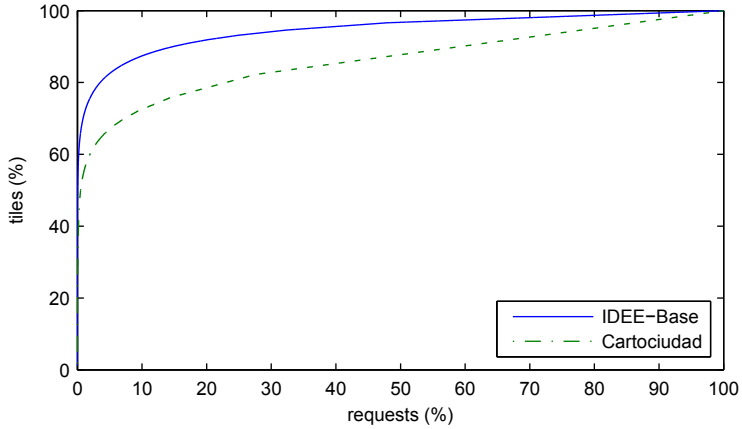


Figure 2. Percentile of requests for the analyzed services.

received from the 9^{th} December of 2009 to 13^{th} May in 2010. IDEE-Base logs reflect a total of 16.891.616 requests received between 15^{th} March and 17^{th} June in 2010.

It must be noted that the performance gain achieved by the use of a tile cache will vary depending on how the tile requests are distributed over the tiling space. If those were uniformly distributed, the cache gain would be proportional to the cache size. However, lucky for us, it has been found that tile requests usually follow a heavy-tailed Pareto distribution, as shown in Figure 2. In our example, tile requests to the Cartociudad map service follow the 20:80 rule, which means that the 20% of tiles receive the 80% of the total number of requests. In the case of IDEE-Base, this behaviour is even more prominent, where the 10% of tiles receive almost a 90% of total requests. Services that show Pareto distributions are well-suited for caching, because high cache hit ratios can be found by caching a reduced fraction of the total tiles.

Figure 3 and Figure 4 show the distribution of tile requests to each resolution level of the tile pyramid for the analyzed services. The maximum number of requests is received at resolution level 4 for both services. This peak is due to the fact that this is the default resolution on the initial rendering of the popular clients in use with this cartography, as it allows the visualization of the whole country on a single screen. As can be observed, the density of requests (requests/tile) is higher at low resolution levels than at higher ones. Because of this, a common practice consists in pregenerating the tiles belonging to the lowest resolution levels, and leave the rest of tiles to be cached on demand when they are first requested.

4. Tile cache implementations

With the standardization of tiled web map services, multiple tile cache implementations have appeared. Between them, the main existent implementations are: TileCache, GeoWebCache and MapProxy. A comparison between these implementations is summarized in Table 1.

As can be seen, TileCache and MapProxy are both implemented in Python (interpreted language), while GeoWebCache is implemented in Java (compiled language). These three

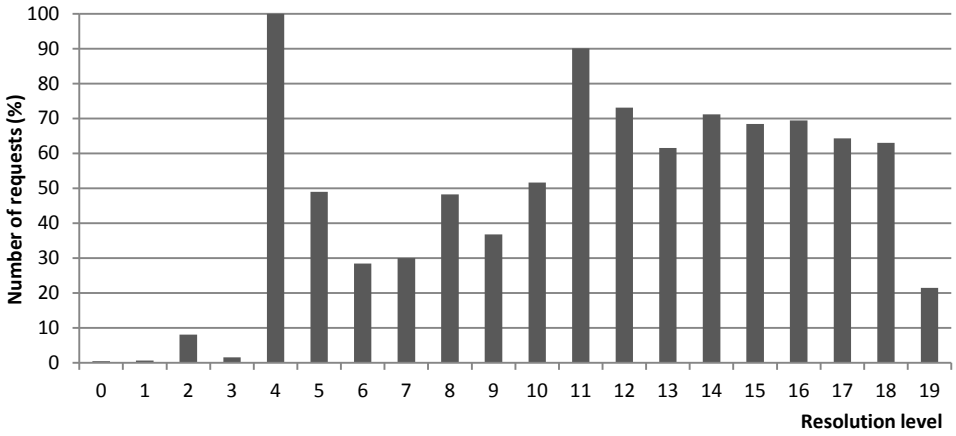


Figure 3. Distribution of requests along the different resolution levels for Cartociudad service.

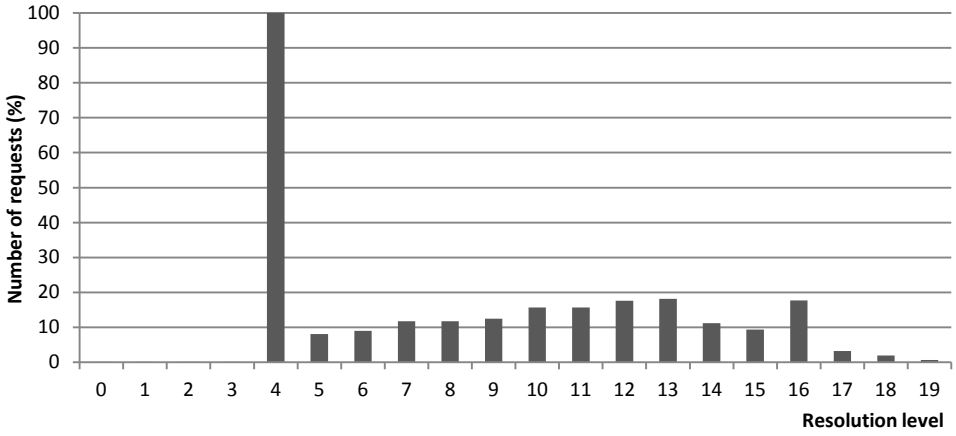


Figure 4. Distribution of requests along the different resolution levels for IDEE-Base service.

services implement the WMS-C, TMS and KML service interfaces. GeoWebCache and MapProxy also offer the WMTS service from OGC. In addition, GeoWebCache can recombine and resample tiles to answer arbitrary WMS requests, and can also be used to serve maps to Google Maps and Microsoft Bing Maps.

All these services offer the possibility of storing map image tiles directly in the file system. TileCache and GeoWebCache also support the MBTiles speficication[4] for storing tiled map data in a SQLite database for immediate use and for transfer. MapProxy supports the Apache CouchDB[5], a document-oriented database that can be queried and indexed in a MapReduce

[4] http://mapbox.com/mbtiles-spec/
[5] http://couchdb.apache.org/

	TileCache	*GeoWebCache*	*MapProxy*
Company	Metacarta Labs	OSGeo	Omniscale
Implementation	Python	Java	Python
Supported services	WMS-C, TMS, KML	WMS, WMS-C, TMS WMTS, KML, Google Maps, Bing Maps	WMS, WMS-C, TMS WMTS, KML
Tile storage	Disk, GoogleDisk, Memcached, Amazon S3, MBTiles	Disk	Disk, MBTiles CouchDB
Tile metadata storage	No	Yes	Yes
Replacement policies	LRU	LRU, LFU	None
Seeding regions	bounding box center and radius	bounding box	bounding box WKT polygons any OGR source
Supports Meta Tiles	Yes	Yes	Yes
Supports Meta Buffer	Yes	Yes	Yes
Reprojection on-the-fly	No	Yes (with Geoserver)	Yes (native)

Table 1. Comparison of features between different open-source tile cache implementations: TileCache, GeoWebCache and MapProxy.

fashion, as backend to store tiles. Moreover, TileCache can store map tiles in the cloud through Amazon S3[6] or to maintain them in memory using Memcached[7].

GeoWebCache maintain tile metadata, such as the last access time or the number of times that each tile has been requested. By using this metadata, it supports the LRU and LFU replacement policies. TileCache supports LRU by using the operating system's time of last access.

These services allow to specify a geographic region for automatically seeding tiles. For example, TileCache can be configured to seed a particular regions defined by a rectangular bounding box or a circle by specifying its center and radius. GeoWebCache supports only the

[6] http://aws.amazon.com/es/s3/
[7] http://memcached.org/

former. MapProxy offers three different ways to describe the extent of a seeding or cleanup task: a simple rectangular bounding box, a text file with one or more polygons in WKT format, or polygons from any data source readable with OGR (e.g. Shapefile, PostGIS).

These three services support both metatiling and meta-buffer methods. The meta-buffer adds extra space at the edges of the requested area.

When a request of a tile in an unsupported coordinate reference system (CRS) is received, both GeoWebCache and MapProxy supports the reprojection on the fly from one of the available CRSs to the specified one. The former achieves this using GeoServer, while the latter offers it natively.

5. Cache management algorithms

Significant improvements can be achieved by using a cache of map tiles, like the ones discussed above. However, adequate cache management policies are needed, especially in local SDIs with lack of resources. In this section, our contributions to the main cache strategies are presented: cache population (or *seeding*), cache replacement and tile prefetching.

5.1. Cache population

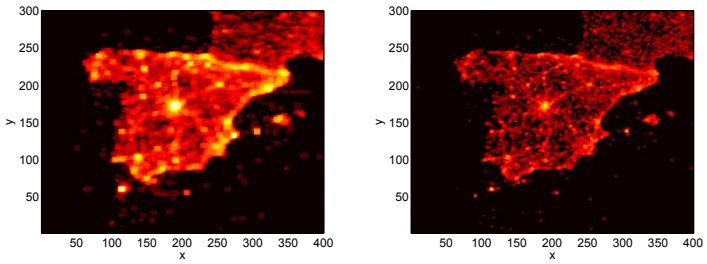
Anticipating the content that users will demand can guide server administrators to know which tiles to pregenerate and to include in their server-side caches of map tiles. With this objective in mind, a predictive model that uses variables known to be of interest to Web map users, such as populated places, major roads, coastlines, and tourist attractions, is presented in [8].

In contrast, we propose a descriptive model based on the mining of the service's past history [7]. Past history can be easily extracted, for example, from server logs. The advantage of this model is that it is able to determine in advance which areas are likely to be requested in the future based exclusively on past accesses, and it is therefore very simple.

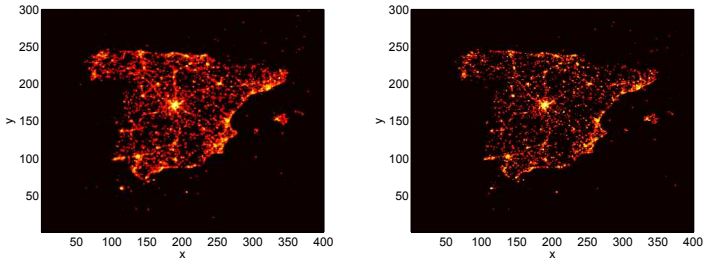
In order to experiment with the proposed model, real-world logs from the IDEE-Base nation-wide public web map service have been used. Request logs were divided in two time ranges of the same duration. The first one was used as source to make predictions and the second one was used to prove the predictions created previously. Due to the difficulty of working with the statistics of individual tiles, the simplified model presented in Section 2 has been used. Concretely, the experiment was conducted with the simplified model to the grid cell defined by the level of resolution 12.

Figure5 shows the heatmaps of requests extracted from the web server logs of IDEE-Base service, propagated to level 12 through the proposed model. These figures demonstrate that some entities such as coast lines, cities and major roads are highly requested. These elements could be used as entities for a predictive model to identify priority objects, as explained in [8].

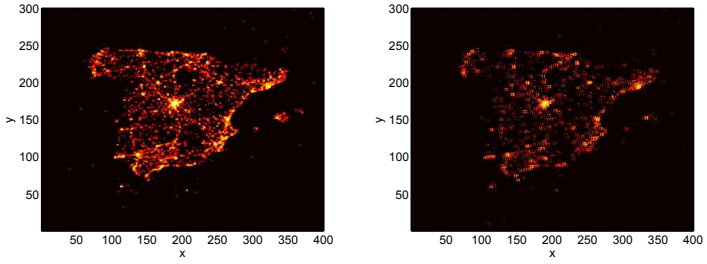
These figures show that near levels are more related than distant ones, but all of them share certain similarity. This relationships between resolution levels encourages the use of statistics collected in a level to predict the map usage patterns in another level with detailer resolution. For example, as shown in Figure5(c) and Figure5(e), resolution levels 14 and 16 are very



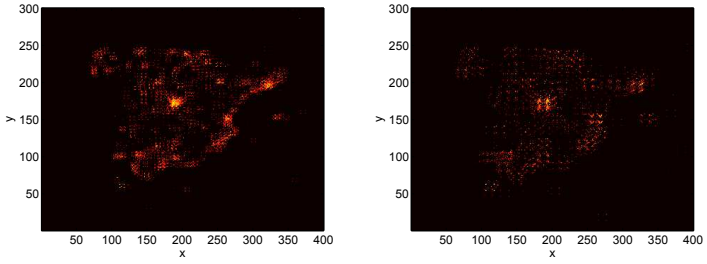
(a) Scale 12 (b) Scale 13



(c) Scale 14 (d) Scale 15



(e) Scale 16 (f) Scale 17



(g) Scale 18 (h) Scale 19

Figure 5. Heatmap of the requests to the *IDEE-BASE* service propagated from levels 12-19 to level 12.

correlated. It is easier to work with the statistics of level 14 than with those of level 16 which has much more elements.

Table 2 represents the hit percentage achieved by using this model for the IDEE-Base service. This table shows the percentage of hits obtained for the level identified by the column index from the statistics collected in the level identified by the row index. Last column shows the resources consumption, as a percentage of cached tiles. Last row collects the results of combining the statistics of all levels to make predictions over every level. Shadowed cell in Table 2 indicates that using retrieved statistics of level 13 as the prediction source, a hit rate of 92.1573% is obtained for predictions made in the level 18, being necessary the storage of a 25.8049% of the tiles in cache.

	12	13	14	15	16	17	18	19	resources
12	98.6417	98.9362	99.3573	99.4737	99.6901	99.2637	99.2993	94.7561	40.2172
13	87.8163	93.5760	95.8939	96.4146	97.4372	95.2686	92.1573	75.5073	25.8049
14	53.0529	61.5825	86.7783	88.2709	91.1807	81.6460	63.4527	43.9129	9.3302
15	37.1553	47.9419	77.9136	84.0861	83.7095	69.9489	57.0746	33.3348	5.2354
16	46.9387	57.5640	84.3110	86.7747	91.8272	78.7670	64.3781	41.8433	7.7686
17	30.2021	37.6348	57.0138	60.1330	62.2834	69.5106	55.5134	23.4647	3.2676
18	23.5791	25.5913	41.7535	46.1559	45.8693	41.4502	61.9763	33.3799	2.3291
19	8.8690	8.6848	12.4556	13.1338	14.3302	12.2756	13.6932	44.1113	1.2295
prop	98.9340	99.3080	99.6074	99.6321	99.7763	99.4244	99.4308	97.2315	41.3647

Table 2. Percentage (%) of cache hits through the simplified model obtained from *IDEE-BASE* logs, using the mean of the normalized frequencies as the probability threshold.

Nevertheless, it must be noted that the main benefit of using a partial cache is not the reduction in the number of cached tiles. The main benefits are the savings in storage space and generation time. As explained in [8], the amount of saved tiles is bigger than the storage saving. It reveals that the most interesting tiles come at a bigger cost. Mainly, popular areas are more complex, and it is necessary more disk space to store them.

Figure6 and Figure7 represent the cache hit ratios obtained by the simplified model for the *IDEE-BASE* service. This model bases its operation on the knowledge of past accesses, assuming a certain stationarity of requests; it assumes that map regions that have been popular in the past will maintain its popularity in the future. However, from a certain percentage of cached objects, identified by the continuous vertical line, the simplified model is not able to make predictions. Tiles situated at the right of this line correspond to objects that have never been requested so are not collected in server logs. To complete the model, these never-requested tiles have been randomly selected for caching, yielding a linear curve for this interval.

Results demonstrate that the simplified model obtains better results for predicting user behavior from near resolution levels. For low-resolution levels high cache hit ratios are achieved by using a reduced subset of the total tiles. However, descending in the scale pyramid, the requested objects percentage decreases, so the model prediction range and its ability to make predictions decrease too. For future work, instead of randomly selecting objects for caching in this interval, interesting features could be identified and used to define priority objects.

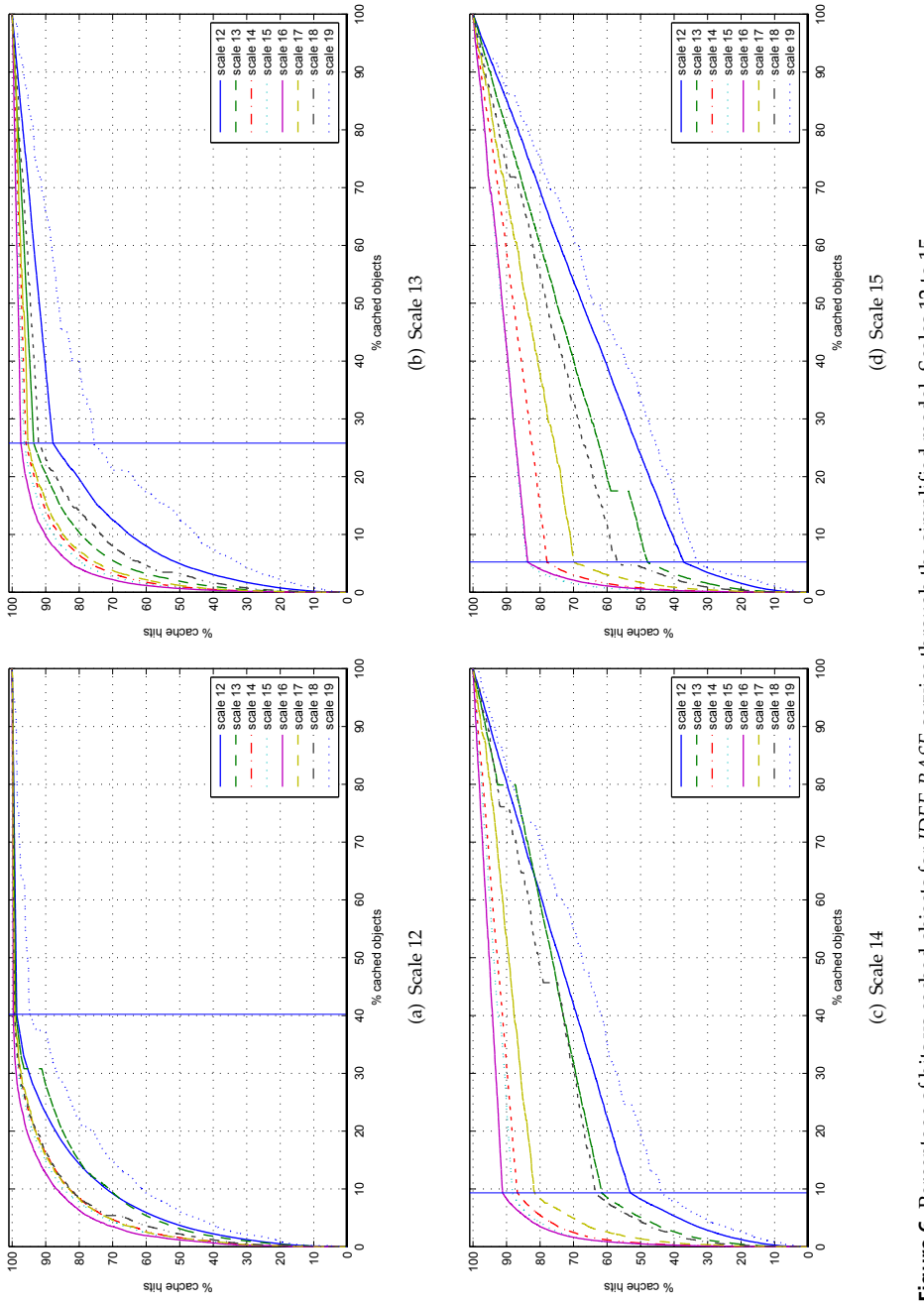


Figure 6. Percentage of hits vs cached objects for *IDEE-BASE* service through the simplified model. Scales 12 to 15.

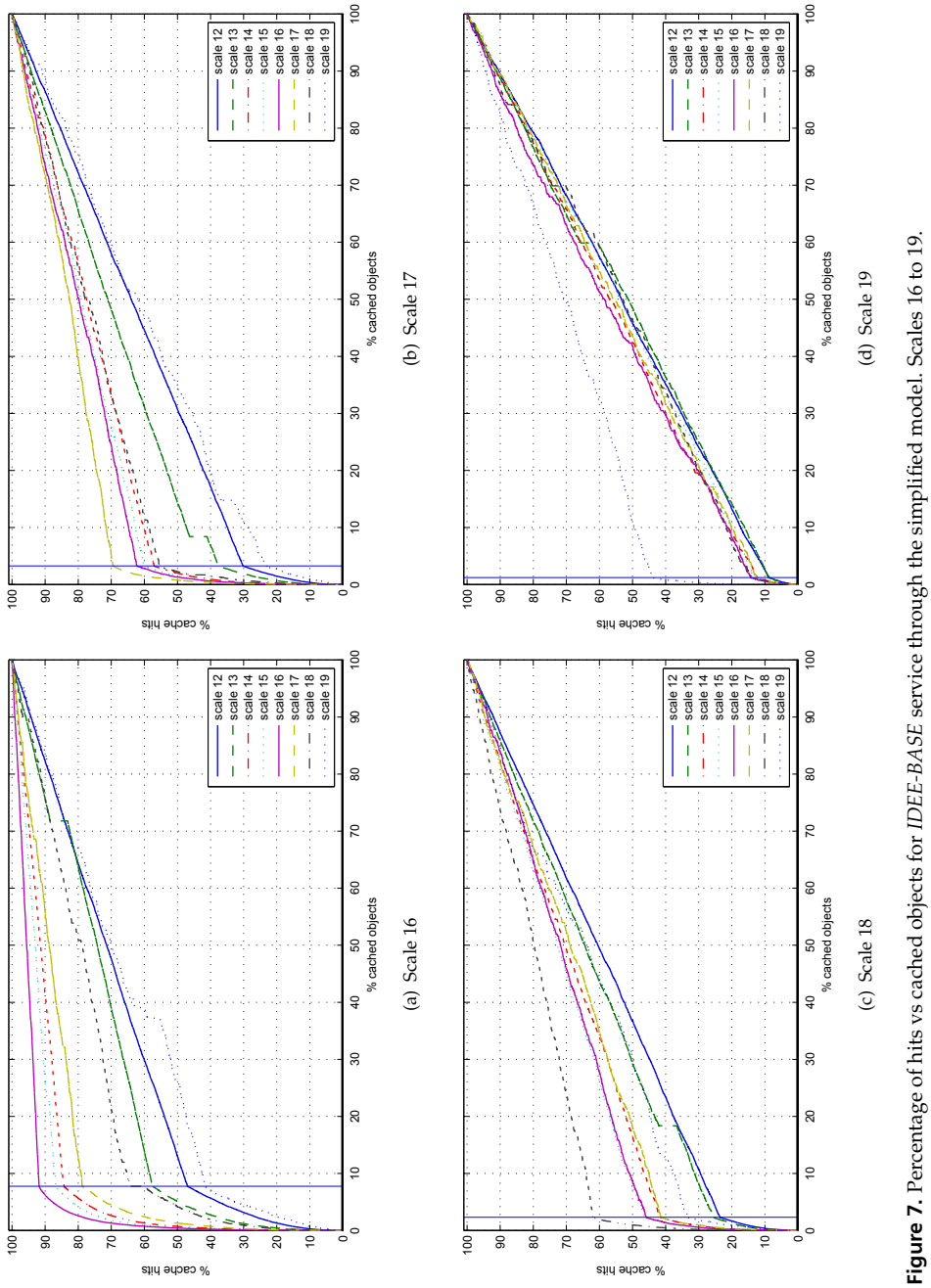


Figure 7. Percentage of hits vs cached objects for *IDEE-BASE* service through the simplified model. Scales 16 to 19.

5.2. Tile pre-fetching

For a given tile request, tile pre-fetching methods try to anticipate which tiles will be requested immediately afterwards. There are several works in the literature that address object prefetching in Web GIS: [9, 10] approximate which tiles will be used in advance based on the global tile access pattern of all users and the semantics of query; [11, 12] use an heuristic method that considers the former actions of a given user.

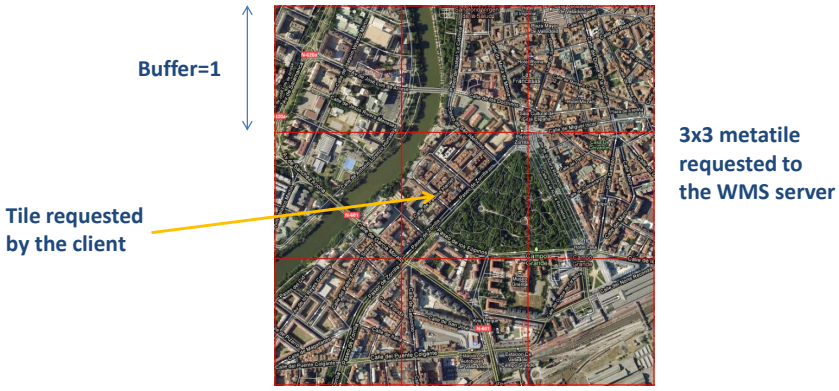


Figure 8. Metatile 3x3 centered in the requested tile.

We propose another pre-fetching strategy, known as metatiling, that works as follows [13]: when the proxy receives a tile request from a client and a cache miss is produced, it requests a larger image tile (called metatile) to the remote backend. This metatile includes the requested tile and also the surrounding ones contained in a specified buffer, as shown in Figure 8. Then, the proxy cuts the metatile into individual tiles, returns the requested tile to the client, and stores all these fragments into the cache, as shown in Figure 9. The main advantage of metatiling is that it can reduce the bottleneck between the proxy cache and the remote Web Map Server.

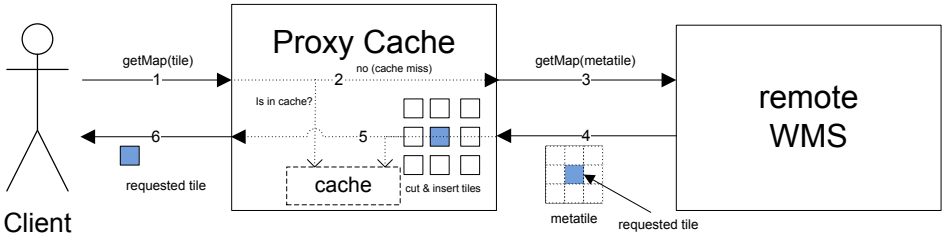
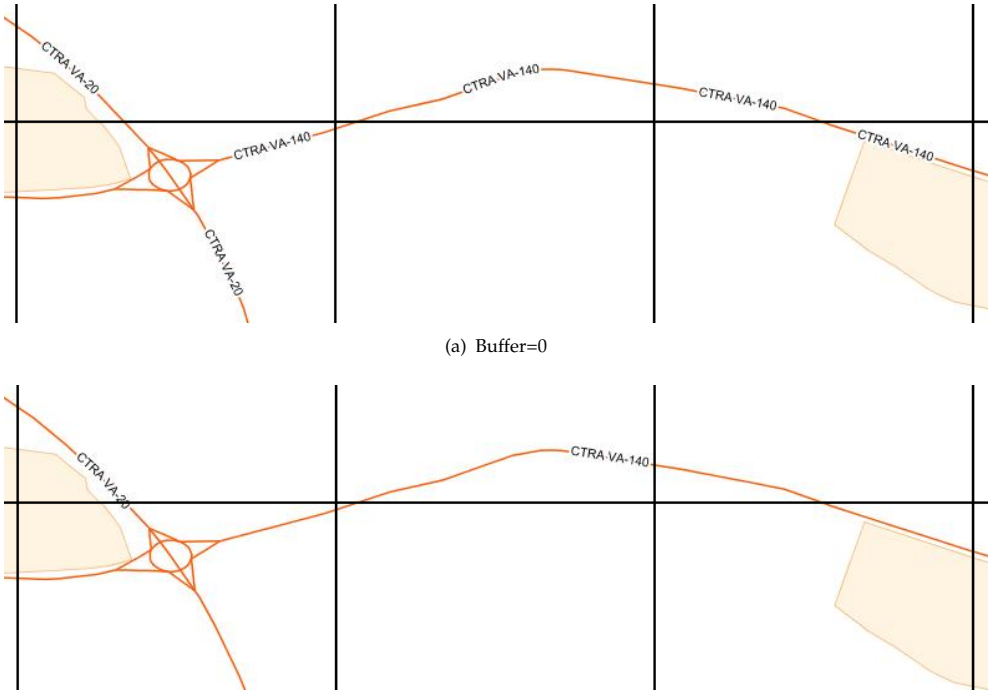


Figure 9. Tile request flow with metatiling.

Moreover metatiling reduces the problem of duplicating the labeling of features that span more than one tile. This problem is illustrated in Figure 10. Depending on the WMS server's configuration, this feature can be labeled once on each tile (Figure 10(a)). By increasing the geographic bounding box of tile requests, the WMS server avoids label duplicates (Figure 10(b)).



(a) Buffer=0

(b) Buffer=2

Figure 10. WMS labelling issues. (a) Requesting individual tiles yields duplicate labels between adjacent tiles. (b) With metatiling labels are not duplicated.

The analyzed tile cache implementations (see Section 4) allow users to configure the size of metatiles. For a given request, the cache orders a metatile of pre-configured size to the WMS server, centered on the requested tile. Considering a scenario where the cache is neither complete nor empty, this selection of the area to generate may not be very efficient, because it is probable that some of the tiles contained in the metatile would already be cached.

Under the assumption that the surrounding area of the requested tile is not uniformly cached, a novel algorithm for the optimal selection of the metatiles to generate has been developed. This procedure, illustrated in Figure 11, seeks to obtain, based on the current state of the cache, the metatile that contains the requested tile (but not necessarily centered in it) and that provides the system with the maximum new information.

In order to validate the hypothesis that a performance improvement can be achieved by using metatiles, the following experiment has been realized. A total of 2000 different tiles have been requested to the CORINE (*CoORdination of INformation of the Environment*) service[8] proxied by the *WMSCWrapper* tile cache. The experiment has been repeated for different metatile sizes, always starting from an empty-cache state. The mean latencies measured for each configuration are collected in Table 3.

[8] http://www.ign.es/ign/layoutIn/corineLandCover.do

buffer (B)	$\tau_{m,metatile}$	$\tau_{m,metatile\_n}$	$Gain_{metatiling}$
0 (no metatiling)	1454,10 ms	1454,10 ms	1
1 (metatile 3x3)	2933,94 ms	325,99 ms	4,46
2 (metatile 5x5)	5660,63 ms	226,42 ms	6,42
3 (metatile 7x7)	9561,54 ms	195,13 ms	7,45

Table 3. Mean latencies to obtain an object from the WMS original service for different metatile sizes. Proxy cache: *WMSCWrapper*; remote WMS: CORINE Land Cover.

The first column of the table shows the mean latency of a cache miss $\tau_{m,metatile}$ for different metatile sizes. This delay includes the transmisission and propagation delays in the network, the map image generation time in the remote web map service and the processing time in the proxy cache. The values of the second column $\tau_{m,metatile\_n}$ are computed by normalizing those of the first column by the number of tiles encompassed by each metatile ($[2B+1]^2$). The last column shows the cache gain achieved by the use of metatiling, computed as the average acceleration in the delivery of a tile versus not using metatiling, as depicted in Equation 1.

$$Gain_{metatiling}(B) = \frac{\tau_{m,metatile\_n}(0)}{\tau_{m,metatile\_n}(B)} \tag{1}$$

Results reflect that the latency involved in the request of a metatile increases with the buffer size. However, it increases in less proportion than the number of tiles it is compossed by. Therefore, the mean latency to obtain each individual tile decreases when increasing the size of the metatile requested to the remote web map service. In other words, it is faster to retrieve a metatile composed by $n \times n$ tiles than the n^2 tiles individually.

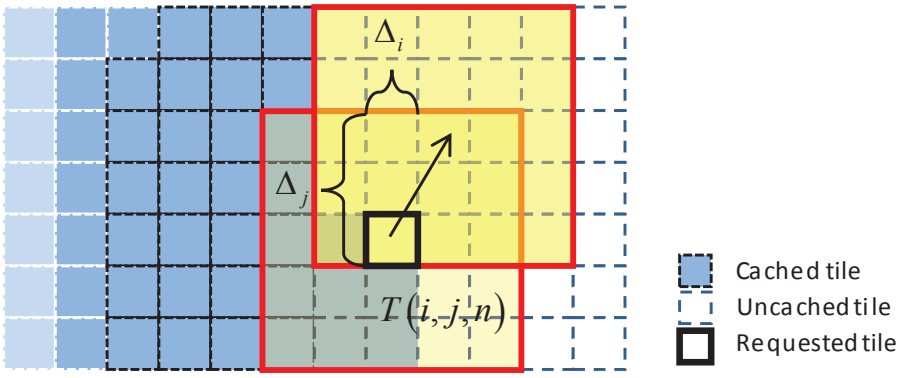


Figure 11. Metatile selection algorithm.

A limiting factor when choosing the metatile size is the overhead in memory consumption required to generate the map image. For example, by default the maximum amount of memory that a single request is allowed to use in *Geoserver* is 16MB, which are sufficient to

render a 2048x2048 image at 4 bytes per pixel, or a 8x8 meta-tile of standard 256x256 pixel tiles.

Table 3 shows the maximum gain that can be achieved by the use of metatiling techniques. This maximum gain occurs when the whole metatile is used to cache new tiles that were not yet cached. While this is the case when automatically seeding tiles in sequential order with non-overlapping metatiles from an empty cache or in the early startup of the service, it would be useful to evaluate metatiling in the most general scenario where the cache is partially filled. In that case, each metatile is likely to add redundant information, since it is probable that some of the tiles encompassed by the metatile were already cached, thus reducing the effective gain of this method.

The performance of metatiling during dynamic cache population with users' requests has been evaluated using the *WMSCWrapper* tile cache, described in Section 4. Simulations were driven by trace files from the public WMS-C tiled web map service of Cartociudad. Being traces recorded in a real, working system, these logs represent a more realistic pattern of user behavior than a synthetic pattern. The CORINE WMS service was used as remote backend.

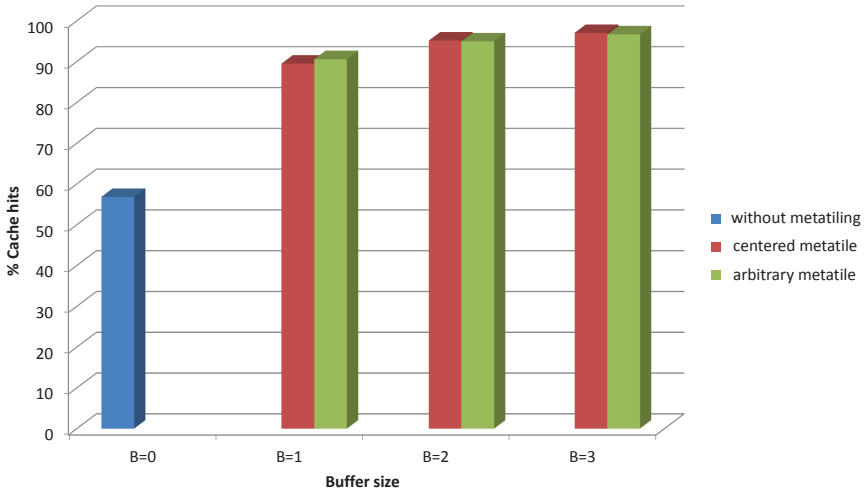


Figure 12. Cache-hit ratios obtained for different buffer sizes and metatile configurations.

A total of 1.000.000 requests were made to the cache. The experiment was repeated for different metatile configurations. For each configuration, the cache-hit ratio and the number of cached tiles after task completion have been collected, starting from an empty cache. Results are shown in Figure 12 and Figure 13.

As can be shown, both the cache-hit ratio and the number of cached tiles grow with the buffer size. For a fixed buffer size, both metatiling strategies (centered and minimum-correlation) obtain similar results. However, the number of cached objects is significantly improved with the minimum-correlation configuration. The improvement increases with the metatile size.

Thus, the advantage achieved with the minimum-correlation metatile configuration is that, maintaining the cache misses, and therefore maintaining the number of requests to the remote

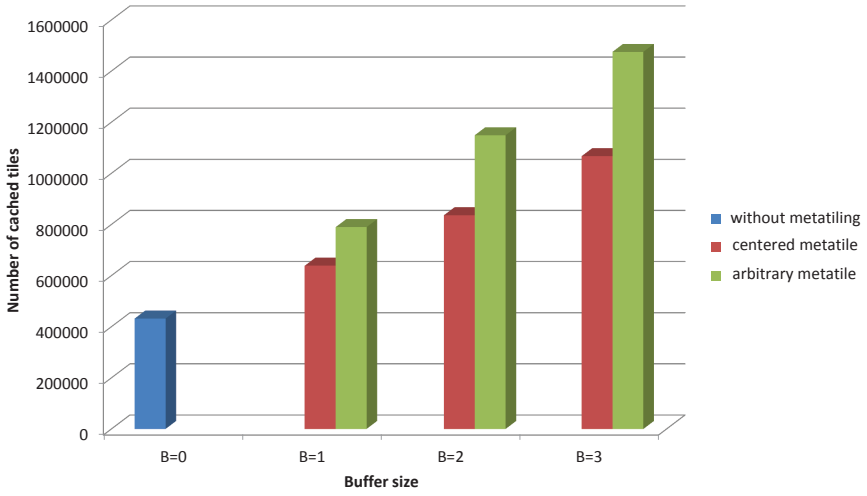


Figure 13. Number of cached tiles for different buffer sizes and metatile configurations.

WMS server, a broaden population of the cache is achieved. These extra pre-generated map image tiles stored in the cache will allow a faster delivery of future requests.

5.3. Cache replacement policies

When the tile cache runs out of space, it is necessary to determine which tiles should be replaced by the new ones. Most important characteristics of Web objects, used to determine candidate objects to evict in Web cache replacement strategies, are: *recency* (time since the last reference to the object), *frequency* (number of times the object has been requested), *size* of the Web object and *cost* to fetch the object from its origin server. These properties classifies replacement strategies as recency-based, frequency-based, recency/frequency-based, function-based and randomized strategies [14]. Recency-based strategies exploit the temporal locality of reference observed in Web requests, being usually extensions of the well-known LRU strategy, which removes the least recently referenced object. Another popular recency-based method is the Pyramidal Selection Scheme (PSS) [15]. Frequency-based strategies rely on the fact that popularity of Web objects is related to their frequency values, and are built around the LFU strategy, which removes the least frequently requested object. Recency/frequency-based strategies combine both, recency and frequency information, to take replacement decisions. Function-based strategies employ a general function of several parameters to make decisions of which object to evict from the cache. This is the case of GD-Size [16], GDSF [17] and Least-Unified Value (LUV) [18]. Randomized strategies use a non-deterministic approach to randomly select a candidate object for replacement.

For a further background, a comprehensive survey of web cache replacement strategies is presented in ([14]). According to that work, algorithms like GD-Size, GDSF, LUV and PSS

were considered "good enough" for caching needs at the time it was published in 2003. However, the explosion of web map traffic did not happen until a few years later.

In this section, we propose a cache replacement algorithm that uses a neural network to estimate the probability of a tile request occurring before a certain period of time, based on the previously discussed properties of tile requests: recency of reference, frequency of reference, and size of the referenced tile [19, 20]. Those tiles that are not likely to be requested shortly are considered as good candidates for replacement.

5.3.1. Related work

The use of neural networks for cache replacement was first introduced by Khalid [21], with the KORA algorithm. KORA uses backpropagation neural network for the purpose of guiding the line/block replacement decisions in cache. The algorithm identifies and subsequently discards the dead lines in cache memories. It is based on previous work by [22], who suggested the use of a shadow directory in order to look at a longer history when making decisions with LRU. Later, an improved version of the former, KORA-2, was proposed [23, 24]. Other algorithms based on KORA were also proposed [25, 26]. A survey on applications of neural networks and evolutionary techniques in web caching can be found in [27]. [28–32] proposes the use of a backpropagation neural network in a Web proxy cache for taking replacement decisions. A predictor that learns the patterns of Web pages and predicts the future accesses is presented in [33]. [34] discusses the use of neural networks to support the adaptivity of the Class-based Least Recently Used (C-LRU) caching algorithm.

5.3.2. Neural network cache replacement

Artificial neural networks (ANNs) are inspired by the observation that biological learning systems are composed of very complex webs of interconnected neurons. In the same way, ANNs are built out of a densely interconnected group of units. Each artificial neuron takes a number of real-valued inputs (representing the one or more dendrites) and calculates a linear combination of these inputs. The sum is then passed through a non-linear function, known as *activation function* or *transfer function*, which outputs a single real-value, as shown in Figure 14.

In this work, a special class of layered feed-forward network known as multilayer perceptron (MLP) has been used, where units at each layer are connected to all units from the preceding layer. It has an input layer with three inputs, two-hidden layers each one comprised of 3 hidden nodes, and a single output (Figure 15). According to the standard convention, it can be labeled as a 3/3/3/1 network. It is known that any function can be approximated to arbitrary accuracy by a network with three layers of units [35].

Learning an artificial neuron involves choosing values for the weights so the desired output is obtained for the given inputs. Network weights are adjusted through supervised learning using subsets of the trace data sets, where the classification output of each request is known. Backpropagation with momentum is the used algorithm for training. The parameters used for the proposed neural network are summarized in Table 4.

The neural network inputs are three properties of tile requests: recency of reference, frequency of reference, and the size of the referenced tile. These properties are known to be important

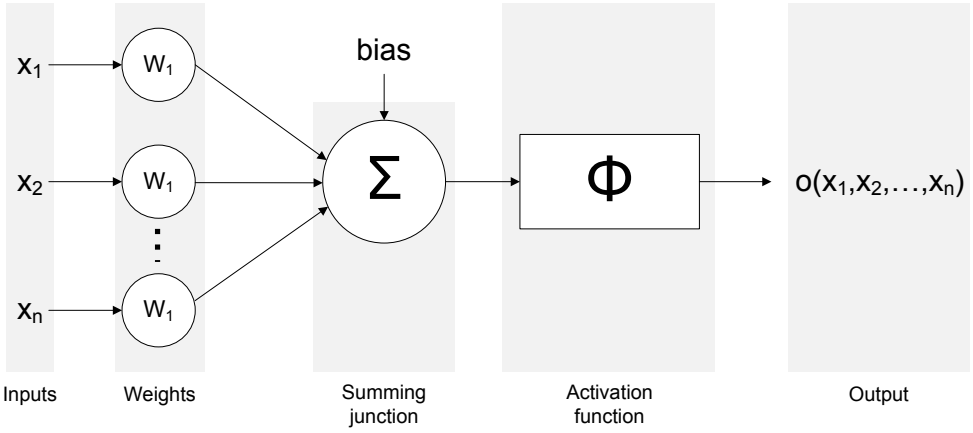


Figure 14. Artificial neuron.

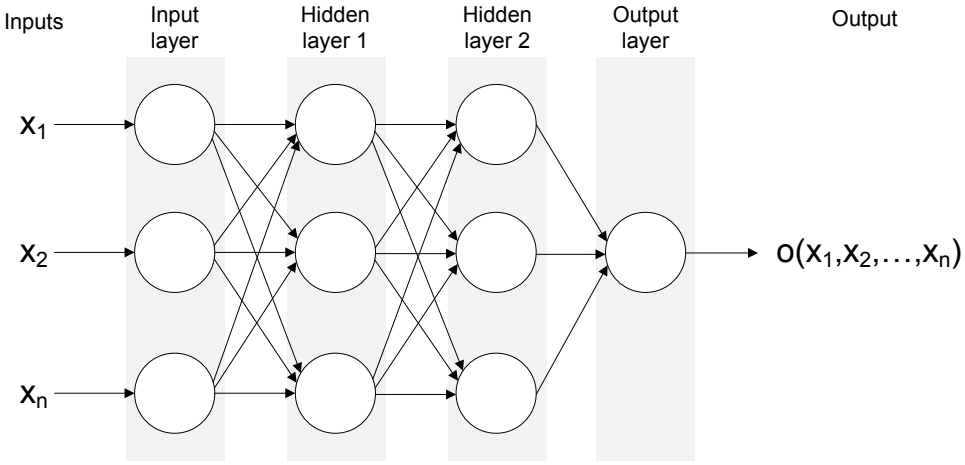


Figure 15. Proposed two-hidden layer feed-forward artificial neural network.

in web proxy caching to determine object cachability. Inputs are normalized so that all values fall into the interval $[-1, 1]$, by using a simple linear scaling of data as shown in Equation 2, where x and y are respectively the data values before and after normalization, x_{min} and x_{max} are the minimum and maximum values found in data, and y_{max} and y_{min} define normalized interval so $y_{min} \leq y \leq y_{max}$. This can speed up learning for many networks.

$$y = y_{min} + (y_{max} - y_{min}) \times \frac{x - x_{min}}{x_{max} - x_{min}} \tag{2}$$

Recency values for each processed tile request are computed as the amount of time since the previous request of that tile was made. Recency values calculated this way do not address the case when a tile is requested for the first time. Moreover, measured recency values could be too disparate to be reflected in a linear scale.

To address this problem, a sliding window is considered around the time when each request is made, as done in [28]. With the use of this sliding window, recency values are computed as shown in Equation 3.

$$recency = \begin{cases} max(SWL, \Delta T_i) & \textit{if object i was requested before} \\ SWL & \textit{otherwise} \end{cases} \tag{3}$$

where ΔT_i is the time since that tile was last requested.

Recency values calculated that way can already be normalized as stated before in Equation 2.

Frequency values are computed as follows. For a given request, if a previous request of the same tile was received inside the window, its frequency value is incremented by 1. Otherwise, frequency value is divided by the number of windows it is away from. This is reflected in Equation 4.

$$frequency = \begin{cases} frequency + 1 & \textit{if } \Delta T_i \leq SWL \\ max\left[\frac{frequency}{\frac{\Delta T_i}{SWL}}, 1\right] & \textit{otherwise} \end{cases} \tag{4}$$

Size input is directly extracted from server logs. As opposite to conventional Web proxies where requested object sizes can be very heterogeneous, in a web map all objects are image tiles with the same dimensions (typically 256x256 pixels). Those images are usually rendered in efficient formats such as PNG, GIF or JPEG that rarely reach 100 kilobytes in size. As discussed in [8], due to greater variation in colors and patterns, the popular areas, stored as compressed image files, use a larger proportion of disk space than the relatively empty non-cached tiles. Because of the dependency between the file size and the "popularity" of tiles,

Parameter	Value
Architecture	Feed-forward Multilayer Perceptron
Hidden layers	2
Neurons per hidden layer	3
Inputs	3 (recency, frequency, size)
Output	1 (probability of a future request)
Activation functions	Log-sigmoid in hidden layers, Hyperbolic tangent sigmoid in output layer
Error function	Minimum Square Error (mse)
Training algorithm	Backpropagation with momentum
Learning method	Supervised learning
Weights update mode	Batch mode
Learning rate	0.05
Momentum constant	0.2

Table 4. Neural network parameters

tile size can be a very valuable input of the neural network to correctly classify the cachability of requests.

During the training process, a training record corresponding to the request of a particular tile is associated with a boolean target (0 or 1) which indicates whether the same tile is requested again or not in window, as shown in Equation 5.

$$target = \begin{cases} 1 \ \textit{if the tile is requested again in window} \\ 0 \ \textit{otherwise} \end{cases} \tag{5}$$

Once trained, the neural network output will be a real value in the range [0,1] that must be interpreted as the probability of receiving a successive request of the same tile within the time window. A request is classified as *cacheable* if the output of the neural network is above 0.5. Otherwise, it is classified as *non cacheable*.

The neural network is trained through supervised learning using the data sets from the extracted trace files. The trace data is subdivided into training, validation, and test sets, with the 70%, 15% and 15% of the total requests, respectivelly. The first one is used for training the neural network. The second one is used to validate that the network is generalizing correctly and to identify overfitting. The final one is used as a completely independent test of network generalization.

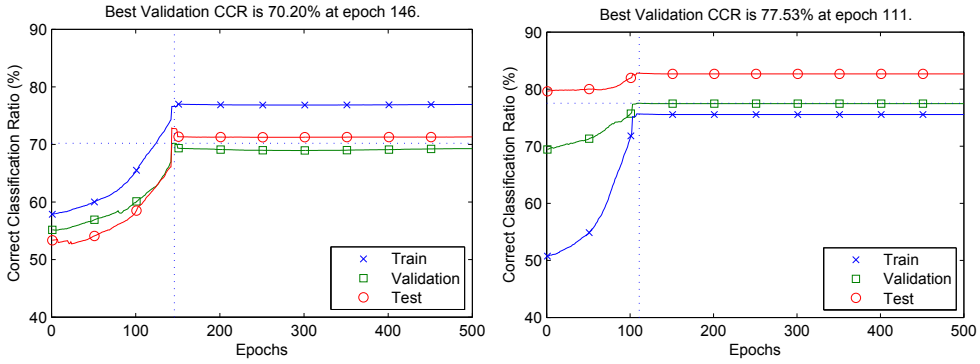
Each training record consists of an input vector of recency, frequency and size values, and the known target. The weights are adjusted using the backpropagation algorithm, which employs the gradient descent to attempt to minimize the squared error between the network output values and the target values for these outputs [36]. The network is trained in batch mode, in which weights and biases are only updated after all the inputs and targets are presented. The pocket algorithm, which saves the best weights found in the validation set, is used.

Neural network performance is measured by the correct classification ratio (CCR), which computes the percentage of correctly classified requests versus the total number of processed requests.

	CartoCiudad	IDEE-Base
training	76.5952	75.6529
validation	70.2000	77.5333
test	72.7422	82.7867

Table 5. Correct classification ratios (%) during training, validation and testing for Cartociudad and IDEE-Base.

Figure 16 shows the CCRs obtained during training, validation and test phases for Cartociudad and IDEE-Base services. As can be seen, the neural network is able to correctly classify the cachability of requests, with CCR values over the testing data set ranging between 72% and 97%, as shown in Table 5. The network is stabilized to an acceptable CCR within 100 to 500 epochs.



(a) Cartociudad (b) IDEE-Base

Figure 16. Correct classification ratios achieved with the neural network for CartoCiudad and IDEE-Base.

6. Conclusion

Serving pre-generated map image tiles from a server-side cache has become a popular way of distributing map imagery on the Web. However, in order to achieve an optimal delivery of online mapping, adequate cache management strategies are needed. These strategies can benefit of the intrinsic spatial nature of map tiles to improve its performance. During the startup of the service, or when the cartography is updated, the cache is temporarily empty and users experiment a poor Quality of Service. In this chapter, a seeding algorithm that populates the cache based on the history of previous accesses has been proposed. The seeder should automatically cache tiles until an acceptable QoS is achieved. Then, tiles could be cached on-demand when they are first requested. This can be improved with short-term prefetching; anticipating the following tiles that will be requested after a particular request can improve users' experience. The metatiling approach presented here requests, for a given tile request, a bigger map image containing adjacent tiles, to the remote WMS backend. Since the user is expected to pan continuously over the map, those tiles are likely to be requested. Finally, when the tile cache runs out of space, it is necessary to determine which tiles should be replaced by the new ones. A cache replacement algorithm based on neural networks has been presented. It tries to estimate the probability of a tile request occurring before a certain period of time, based on the following properties of tile requests: recency of reference, frequency of reference, and size of the referenced tile. Those tiles that are not likely to be requested shortly are considered as good candidates for replacement.

Acknowledgements

This work has been partially supported by the Spanish Ministry of Science and Innovation through the project "España Virtual" (ref. CENIT 2008-1030), a FPI research fellowship from the University of Valladolid (Spain), the National Centre for Geographic Information (CNIG) and the National Geographic Institute of Spain (IGN).

Author details

Ricardo García, Juan Pablo de Castro, Elena Verdú, María Jesús Verdú and Luisa María Regueras
Department of Signal Theory, Communications and Telematics Engineering, Higher Technical School of Telecommunications Engineering, University of Valladolid, Campus Miguel Delibes, Paseo Belén 15, 47011 Valladolid, Spain

7. References

[1] Jeff de la Beaujardiere, editor. *OpenGIS Web Map Server Implementation Specification*. Open Geospatial Consortium Inc, OGC 06-042, 2006.

[2] Elias Ioup John T. Sample. *Tile-Based Geospatial Information Systems, Principles and Practices*. Springer Science+Business Media, LLC, Boston, MA, online-ausg. edition, 2010.

[3] Open Geospatial Foundation. WMS-C wms tile caching - OSGeo wiki, 2008.

[4] Núria Juliá Juan Masó, Keith Pomakis, editor. *OpenGIS Web Map Tile Service Implementation Standard*. Open Geospatial Consortium Inc, OGC 07-057r7, 2010.

[5] J. C. Muller. Generalization of spatial databases. *Geographical Information Systems*, 1:457–475, 1991.

[6] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: a distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, pages 15–15, Berkeley, CA, USA, 2006. USENIX Association.

[7] R. García, J.P. de Castro, M.J. Verdú, E. Verdú, L.M. Regueras, and P. López. A descriptive model based on the mining of web map server logs for tile prefetching in a web map cache. *International Journal of Systems Applications, Engineering & Development*, 5(4):469–476, 2011.

[8] Sterling Quinn and Mark Gahegan. A predictive model for frequently viewed tiles in a web map. *T. GIS*, 14(2):193–216, 2010.

[9] Yong-Kyoon Kang, Ki-Chang Kim, and Yoo-Sung Kim. Probability-based tile pre-fetching and cache replacement algorithms for web geographical information systems. In *Proceedings of the 5th East European Conference on Advances in Databases and Information Systems*, ADBIS '01, pages 127–140, London, UK, 2001. Springer-Verlag.

[10] V. K Kang, Y. W Park, B. Hwang, and Y. S Kim. Performance profits of tile pre-fetching in multi-level abstracted web geographic information systems. In *Proceedings of the Int. Symposium on Internet and Multimedia*, Indonesia, December 2002.

[11] Dong Lee, Jung Kim, Soo Kim, Ki Kim, Kim Yoo-Sung, and Jaehyun Park. Adaptation of a neighbor selection markov chain for prefetching tiled web gis data. In Tatyana Yakhno, editor, *Advances in Information Systems*, volume 2457 of *Lecture Notes in Computer Science*, pages 213–222. Springer Berlin / Heidelberg, 2002.

[12] Serdar Yesilmurat and Veysi Isler. Retrospective adaptive prefetching for interactive web gis applications. *GeoInformatica*, pages 1–32, 2011. 10.1007/s10707-011-0141-8.

[13] R. Garcia, J.P. de Castro, M.J. Verdu, E. Verdu, L.M. Regueras, P. Lopez, and D. Garcia. Estrategias de metatiling para la aceleración de servicios de mapas teselados en las

infraestructuras de datos espaciales. In *The X Symposium on Telematic Engineering, JITEL 2011*, 2011.

[14] S. Podlipnig and L. Böszörmenyi. A survey of web cache replacement strategies. *ACM Computing Surveys (CSUR)*, 35(4):374–398, 2003.

[15] C. Aggarwal, J.L. Wolf, and P.S. Yu. Caching on the world wide web. *Knowledge and Data Engineering, IEEE Transactions on*, 11(1):94 –107, jan/feb 1999.

[16] Pei Cao and Sandy Irani. Cost-aware www proxy caching algorithms. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems on USENIX Symposium on Internet Technologies and Systems*, pages 18–18, Berkeley, CA, USA, 1997. USENIX Association.

[17] Martin Arlitt, Ludmila Cherkasova, John Dilley, Rich Friedrich, and Tai Jin. Evaluating content management techniques for web proxy caches. *SIGMETRICS Perform. Eval. Rev.*, 27:3–11, March 2000.

[18] Hyokyung Bahn, Kern Koh, S.H. Noh, and S.M. Lyul. Efficient replacement of nonuniform objects in web caches. *Computer*, 35(6):65 –73, June 2002.

[19] R. García, J.P. de Castro, M.J. Verdú, E. Verdú, L.M. Regueras, and P. López. An adaptive neural network-based method for tile replacement in a web map cache. In Beniamino Murgante, Osvaldo Gervasi, Andrés Iglesias, David Taniar, and Bernady Apduhan, editors, *Computational Science and Its Applications - (ICCSA 2011)*, volume 6782 of *Lecture Notes in Computer Science*, pages 76–91. Springer Berlin / Heidelberg, 2011. 10.1007/978-3-642-21928-3_6.

[20] R. Garcia, J.P. de Castro, M.J. Verdu, E. Verdu, L.M. Regueras, and P. Lopez. A cache replacement policy based on neural networks applied to web map tile caching. In *The 2011 International Conference on Internet Computing - (ICOMP)*, 2011.

[21] Humayun Khalid. A new cache replacement scheme based on backpropagation neural networks. *SIGARCH Comput. Archit. News*, 25:27–33, March 1997.

[22] J. Pomerene, T. R. Puzak, R. Rechtschaffen, and F. Sparacio. Prefetching mechanism for a high-speed buffer store. *US patent*, 1984.

[23] H. Khalid and M.S. Obaidat. Kora-2: a new cache replacement policy and its performance. In *Electronics, Circuits and Systems, 1999. Proceedings of ICECS '99. The 6th IEEE International Conference on*, volume 1, pages 265 –269 vol.1, 1999.

[24] H. Khalid. Performance of the KORA-2 cache replacement scheme. *ACM SIGARCH Computer Architecture News*, 25(4):17–21, 1997.

[25] M.S. Obaidat and H. Khalid. Estimating neural networks-based algorithm for adaptive cache replacement. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 28(4):602 –611, August 1998.

[26] H. Khalid and MS Obaidat. Application of neural networks to cache replacement. *Neural Computing & Applications*, 8(3):246–256, 1999.

[27] P. Venketesh and R. Venkatesan. A Survey on Applications of Neural Networks and Evolutionary Techniques in Web Caching. *IETE Technical Review*, 26(3):171–180, 2009.

[28] H. ElAarag and S. Romano. Training of nnpcr-2: An improved neural network proxy cache replacement strategy. In *Performance Evaluation of Computer Telecommunication Systems, 2009. SPECTS 2009. International Symposium on*, volume 41, pages 260 –267, july 2009.

[29] S. Romano and H. ElAarag. A neural network proxy cache replacement strategy and its implementation in the Squid proxy server. *Neural Computing & Applications*, pages 1–20.
[30] H. ElAarag and J. Cobb. A Framework for using neural networks for web proxy cache replacement. *SIMULATION SERIES*, 38(2):389, 2006.
[31] H. ElAarag and S. Romano. Improvement of the neural network proxy cache replacement strategy. In *Proceedings of the 2009 Spring Simulation Multiconference*, pages 1–8. Society for Computer Simulation International, 2009.
[32] J. Cobb and H. ElAarag. Web proxy cache replacement scheme based on back-propagation neural network. *Journal of Systems and Software*, 81(9):1539–1558, 2008.
[33] W. Tian, B. Choi, and V. Phoha. An Adaptive Web Cache Access Predictor Using Neural Network. *Developments in Applied Artificial Intelligence*, pages 113–117, 2002.
[34] R.A. El Khayari, M.S. Obaidat, and S. Celik. An Adaptive Neural Network-Based Method for WWW Proxy Caches. *IAENG International Journal of Computer Science*, 36(1):8–16, 2009.
[35] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2:303–314, 1989. 10.1007/BF02551274.
[36] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.