A Technical Survey on Decluttering of Icons in Online Map-based Mashups

Haosheng Huang and Georg Gartner

Institute of Geoinformation and Cartography, Vienna University of Technology, Austria

{haosheng.huang, georg.gartner}@tuwien.ac.at

Abstract

Recent years have witnessed rapid advances in online map-based mashups with Application Programming Interfaces (APIs) and web services. Map-based mashups often display different kinds of information (e.g., POIs, represented as icons) on base maps, such as Google Maps and Bing Maps. The visualization of large number of icons in a map on web browsers or mobile devices often results in the icon cluttering problem with icons touching and overlapping each other. This problem decreases map legibility, and thus prevents users from effectively processing the information presented in the map. It also leads to a dramatic degradation of performance, and high transmission load. All these problems will greatly decrease the usability of a mashup application.

This paper surveys and assesses approaches from different disciplines (i.e., computer science and cartography) for avoiding icon cluttering in online map-based mashups. We focus on two issues: filtering of irrelevant POIs, and icon placement and aggregation. Different techniques from information filtering research are analyzed and compared for reducing the number of icons to be displayed in a map. For the latter issue, approaches of aggregating and placing icons from map generalization research are discussed and assessed for their applicability in online mashups. Some related APIs and typical mashup examples are also discussed and compared. This paper concludes that in order to provide more cartographically pleasing map in mashups, techniques from computer science and cartography should be seamlessly integrated.

Key words: icon cluttering, map-based mashups, information filtering, icon aggregation, API

1. Introduction

Compared to Web 1.0, Web 2.0 provides users with more user-friendly interface and software. Users can also contribute their own data (user-generated content) and control the data (Wikipedia, 2011a). Some well-known Web 2.0 sites are Facebook, Twitter, Wikipedia, and OpenStreetMap. Mashup is a new web development method in the era of Web 2.0. The term mashup implies easy, fast integration, frequently using open Application Programming Interfaces (APIs) to combine data, presentation or functionality from two or more sources to create new services (Wikipedia, 2011b). According to the survey made by ProgrammableWeb (2011), mapping is the most popular category in mashup applications.

Map-based mashups often overlap different information (such as Points of Interest (POIs), represented as icons) on base maps, e.g., Google Maps, Bing Maps, and OpenStreetMap. The visualization of a great many icons in a map often results in dramatic degradation of performance, and high transmission load. More importantly, it may lead to the icon cluttering problem with many icons touching and overlapping each other (Burigat and Chittaro, 2008). This problem decreases map legibility, and thus prevents users from effectively processing the information presented in a map.

The goal of this paper is to survey and assess approaches from different disciplines for avoiding icon cluttering in online map-based mashups. After carefully analyzing the problems in Section 2, we focus on two key issues: filtering of irrelevant POIs, and icon placement and aggregation. Related filtering methods from the field of computer science are then analyzed and compared for reducing the number of icons to be displayed in a map (Section 3). Section 4 surveys and analyzes approaches for aggregating and placing icons from research in cartographic generalization. In Section 5, we discuss the implementation (existing APIs) of the described methods and some typical mashup websites. Finally, we summarize the work in Section 6.

2. Icon cluttering in online map-based mashups

A mashup can be simply understood as a new service, which combines functionality or content from different existing sources. In recent years, many APIs and web services have been made available to not only programmers but also to end users. Users with some basic programming knowledge can easily create their own mashups. There are also some tools and editors, such as Yahoo! Pipes, providing graphical interface for building mashups. All users have to do is to drag and drop content from one source to another. They are very useful to end users and require little technical understanding. As a result, more and more mashups are created and published on the web.

Map-based mashup is the most popular type of mashup. This is because about most of information is spatially-related, and map is a logical interface for visualization. Most importantly, mapping APIs, for example, Google Maps API, provide completed base maps, and can be easily integrated with some other data APIs. Map-based mashups often display different kinds of information (e.g., POIs) on base maps. These POIs are usually visualized as icons (e.g., push pins, markers in Google Maps) in a base map, with some multimedia information – mostly text, images, and videos – attached to the icons (Haklay et al., 2008).

Showing a small number of icons or markers in a map (e.g., a Google map) is fairly easy. With the impetus of Web 2.0 applications, such as Facebook, Flickr, and Foursquare, huge amounts of user-generated content (UGC) or Volunteered Geographic Information (VGI) are being continually created. More and more mashups display a large number of icons (POIs) in a single map.

Therefore, some key challenges and problems appear. The performance of the mashup will be dramatically decreased. Users' browsers may freeze or become unresponsive for a period of time. The memory occupied will also be sharply increased. For example, GMarker is used in Google Maps APIs v2 for showing icons. A test by Gabriel Svennerberg on a PC with a 3.60 GHz Pentium 4 HT processor and 2 Gb RAM running Windows XP shows that when visualizing 500 markers in a map on Internet Explorer 8.0 takes 3329 ms (Svennerberg, 2009). Additionally, visualizing a great many of icons in a map also brings a high transmission load, which may be impractical for mobile applications with slower connection speeds.

Furthermore, visualizing large number of icons in a map often results in cluttering problems, especially at a small map scale (Burigat and Chittaro, 2008). Icons will touch and overlap each other. They may also mask other important map features such as roads and place names. It becomes worse when the map changes from a larger scale to a smaller scale. This cluttering problem decreases map legibility, and thus prevents users effectively processing the amount of information presented in the map. For example, Figure 1 shows restaurants near the first district of Vienna as icons in a Google map. In this map, icons overlap each other and mask most of the other map features. End users cannot really get some useful information from it.

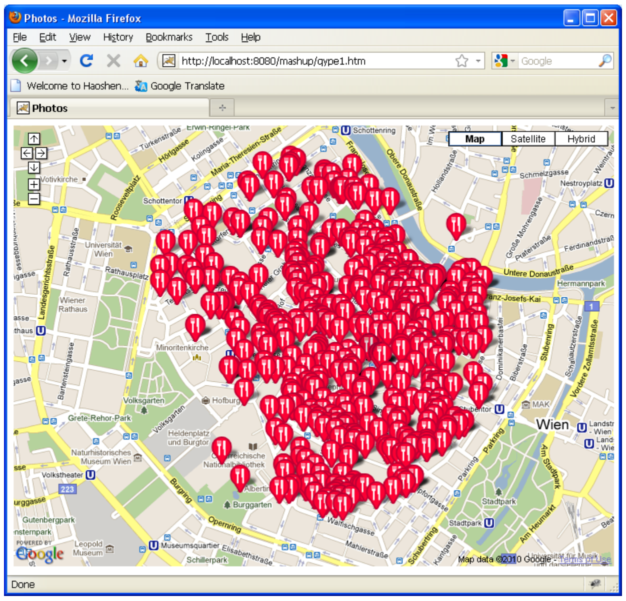
Photos - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/mashup/qype1.htm
Map Satellite Hybrid
Wien
Done

Figure 1. An example of icon cluttering in a map-based mashup. Restaurants near the first district of Vienna are displayed.

Generally, the above problems can be alleviated by two approaches: filtering of irrelevant POIs, and icon placement and aggregation (Burigat and Chittaro, 2008). Filtering of irrelevant POIs is used to reduce the number of icons to be displayed in a map. The later issue displaces and aggregates icons, and shows them in a cartographically pleasing way (Kovanen and Sarjakoski, 2010).

In the literature, many papers have addressed the cluttering problems from a cartographic perspective, and focused on icon placement and aggregation. Different map generalization methods were designed to displace and aggregate icons (Burigat and Chittaro, 2008; Kovanen and Sarjakoski, 2010). However, the cluttering problems can also be tackled from a computer science perspective. For example, by applying information filtering methods, irrelevant POIs can be filtered, and thus the number of icons to be displayed in a map is reduced.

In order to provide uncluttered maps, both filtering of irrelevant POIs, and icon placement and aggregation should be seamlessly combined. In the following, we survey and assess techniques from the fields of computer science and cartography to address the above problems.

3. Filtering of irrelevant POIs

Information filtering (IF) is a useful technique for delivery of relevant information. An IF system removes redundant or unwanted information from a source using (semi)automated or computerized methods prior to presentation to a human (Wikipedia, 2011c). It can help alleviate the problem of "information overload" (Wikipedia, 2011d), and thus improve users' decision-making. Recommender systems (RSs) are active IF systems that attempt to provide the user with relevant information (for example, relevant to his/her interests, needs and context). For map-based mashups, IF/RSs techniques can help to filter irrelevant POIs, and therefore reduce the number of icons to be displayed in a map.

Varieties of techniques have been proposed for RSs/IF. Previous work proposed different classification of RSs (Hanani et al., 2001; Burke, 2007; Ricci et al., 2010). Among them, Ricci et al. (2010) provided an up-to-date classification of RSs: content-based, collaborative filtering, demographic, knowledge-based, community-based, and hybrid RSs. In the following, we analyze and assess these six techniques for reducing the number of icons to be displayed in a map.

A mashup providing users with relevant restaurants (POIs) in maps will be used as the illustrated example. It obtains POI data by using Qype API, and visualizes these POIs in Google Maps with Google Maps API (v2).

3.1 Content-based filtering

Content-based RSs recommend items (e.g., POIs, restaurants) similar to those the user has liked in the past. These systems build a model or profile of user's preferences based on the features (description) of the objects rated/chosen/liked by that user (Lops et al., 2010). The profile is a structured representation of user's preferences. The recommendation process matches up the attributes of the user profile with the attributes of an object (object profile). The result is a relevant judgment that represents the user's level of interest in that object. Objects with higher relevant judgment values are often recommended to the end user. The performance of content-based RSs mainly depends on how accurate the profile reflects the user's preferences. For a state-of-the-art survey, please refer to Lops et al. (2010).

In a simple form, the user himself can also explicitly provide a user profile. For example, in our restaurant finder mashup, a user can explicitly choose some types of restaurants (e.g., Japanese restaurants) to be displayed in a map. In Qype.com, every restaurant (POI) has been labeled with different tags, which can be viewed as attributes (description) of the restaurant. Therefore, irrelevant POIs (restaurants) can be filtered according to the user's choices. Figure 2 illustrates an example that only shows Japanese restaurants.

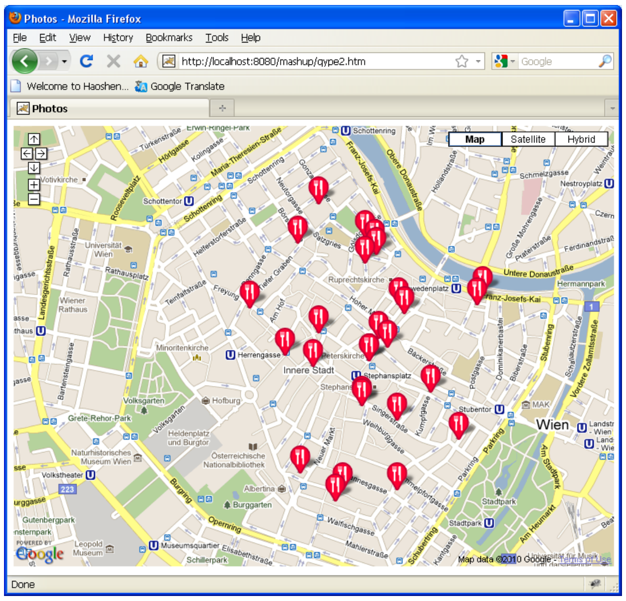
Photos - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/mashup/qype2.htm
Welcome to Haoshen... Google Translate
Photos
Map Satellite Hybrid
Wien
Innere Stadt
Stephansplatz
Done

Figure 2. An example of content-based filtering. Only Japanese restaurants are displayed.

As the increasing popularity of social networking websites, such as Facebook and MySpace, more and more users store (explicitly or implicitly) their personal preferences and interests on these social websites. Therefore, a user profile can also be obtained from his/her social networking accounts, which may exempt the user from active involvement (i.e., explicitly choosing from a list). There are some open APIs available for this purpose, such as OpenSocial API by Google, and Facebook Platform by Facebook. Morandell (2010) provided an example. He used the OpenSocial API to inquire user's preferences that are stored in her/his MySpace account, and then used the preferences to filter irrelevant POIs.

Content-based RSs have several limitations (Adomavicius and Tuzhilin, 2005; Lops et al., 2010), such as limited content analysis in building profiles (both users and objects), and overspecialization. Sometimes, as little description about objects is available, it is hard to build profiles that accurately reflect the characteristics of objects and preferences of users. Overspecialization means content-based RSs have no inherent method for finding something unexpected (Lops et al., 2010). For example, a person with no experience with Thai cuisine would never receive a recommendation for Thai restaurants even if the best restaurant in town is a Thai restaurant.

3.2 Collaborative filtering

Collaborative filtering (CF) recommends a user the items (POIs) that other users with similar tastes liked in the past. It is the most popular and widely implemented technique in RSs. Amazon-like recommendation ("people who bought … also bought …") is a well-known CF application.

Algorithms for CF can be grouped into two general classes: model-based and memory-based (or heuristic-based). Model-based CF uses the collection of ratings to learn a model, which is then used to make rating predictions. Some probabilistic models (such as Bayesian network and cluster model) are often employed for model learning. Heuristic-based CF can be divided into user-based approach and item-based approach. Given an unknown rating (of an item by the current user) to be estimated, heuristic-based CF first measures similarities between the current user and other users (user-based), or, between the item and other items (item-based). Then the unknown rating is predicted by averaging (weighted) the known ratings of the item by similar users (user-based), or the known ratings of similar items by the current user (item-based). User-based CF can be viewed as a heuristic implementation of the "Word of Mouth" phenomenon (Wang et al., 2006). For an up-to-date survey, please refer to Desrosiers and Karypis (2010).

Popularity-based recommendation (e.g., "the most viewed" at YouTube, "the most popular tags" at Flickr) is a non-personalized CF. These "most popular (viewed, discussed)…" like recommendations have been proved to be very useful for end users. Figure 3 depicts our restaurant finder mashup implementing the popularity-based technique to reduce the icons.

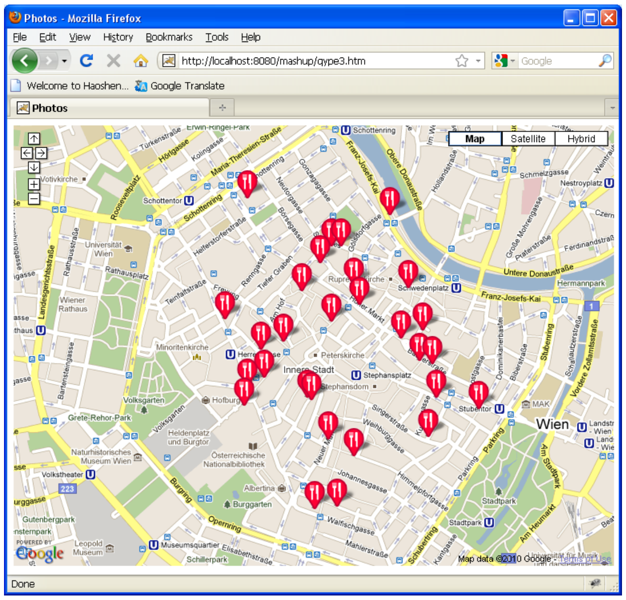
Photos - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/mashup/qype3.htm
Welcome to Haoshen... Google Translate
Photos
Map Satellite Hybrid
Wien
Innere Stadt
Done

Figure 3. An example of popularity-based recommendation in the restaurant finder mashup. Only the 30 top rated restaurants are displayed.

The biggest advantage of CF is that it requires no previous knowledge about the content of the data, and thus can be applied to any type of data, regardless of content. However, pure CF has some disadvantages (Desrosiers and Karypis, 2010). Two of them are cold-start problem (new user problem and new item problem), and data sparsity (too few common ratings). In order to make accurate recommendations, the system must first learn the user's preferences from the ratings that the user gave. For a new user, as he/she has few or no ratings available in the system, it is hard to make recommendations to him/her. It is also impossible to recommend a new item to users when using pure CF. The problem of data sparsity is caused by the fact that users typically rate only a small proportion of the available items. When the rating data are sparse, two users or items are unlikely to have common ratings, and consequently, heuristic-based CF will predict ratings using a very limited number of neighbors (Desrosiers and Karypis, 2010). Therefore, the recommendation performance may suffer from data sparsity.

3.3 Demographic recommendation

This type of system recommends items based on the demographic profile of a user. The assumption is that different recommendations should be made for users with different demographics. Many websites adopt simple and effective personalization solutions based on demographics. For example, users are dispatched to particular websites based on their language or country (Ricci et al., 2010).

Figure 4 shows an example of using demographic information to filter irrelevant POIs in map: making recommendations for a disabled person. Only restaurants tagged with "disabled access" are shown.

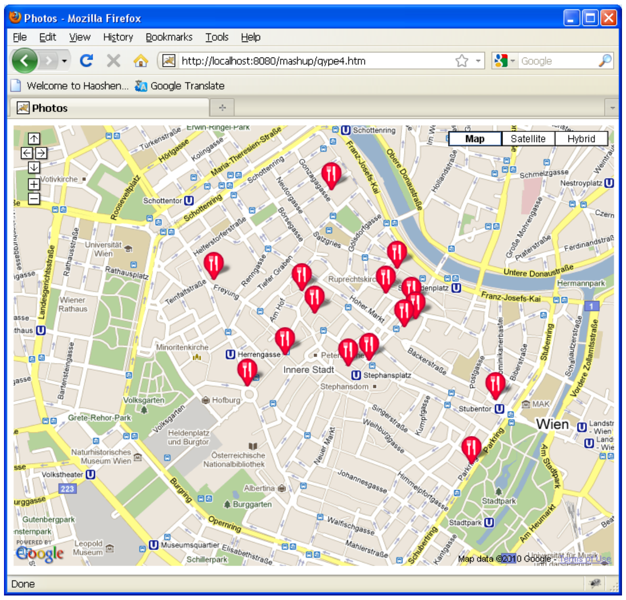
Photos - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/mashup/qype4.htm
Welcome to Haoshen... Google Translate
Photos
Map Satellite Hybrid
Wien
Innere Stadt
Done

Figure 4. An example of demographic recommendation in the restaurant finder mashup. Only restaurants tagged with "disabled access" are shown.

3.4 Knowledge-based recommendation

Knowledge-based RSs recommend items based on predefined knowledge bases that contain explicit rules about how certain item features meet user needs and preferences and, ultimately, how the item is useful for the user (Felfernig et al., 2010). Compared to CF and content-based filtering, knowledge-based RSs have no cold-start problems since the user's requirements are directly elicited within a recommendation session through a series of dialogs. However, they suffer from "the knowledge acquisition bottleneck in the sense that knowledge engineers must work hard to convert the knowledge possessed by domain experts into formal, executable representations" (Felfernig et al., 2010, p187-188).

A knowledge base is typically defined by two sets of variables (V_C, V_{PROD}) and three different sets of constraints (C_R, C_F, C_{PROD}) (Felfernig et al., 2010). Customer Properties V_C describe possible requirements of customers (users), i.e., requirements are instantiations of customer properties, which may be explicitly provided by users via a series of dialogs. Product Properties V_{PROD} describe the properties of a given product assortment. Constraints C_R are systematically restricting the possible instantiations of customer properties. Filter Conditions C_F define the relationship (rule) between potential customer requirements and the given product assortment. Products C_{PROD} store all the products, and represent them by using the properties defined by V_{PROD}. Among them, Filter Conditions C_F plays a key role. An example of C_F rule can be

C_F = {CF1: With_cash = not → Credit_cards_accepted = Yes}

It can be explained as "users without cash should receive recommendations (restaurants) which accept credit cards". Figure 5 shows an example using this rule.

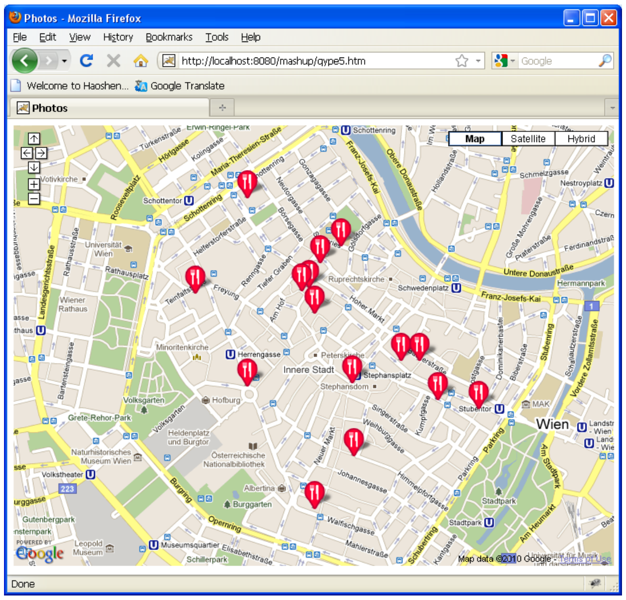
Photos - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/mashup/qype5.htm
Welcome to Haoshen... Google Translate
Photos
Map Satellite Hybrid
Innere Stadt
Wien
Done

Figure 5. An example of knowledge-based recommendation in the restaurant finder mashup.

3.5 Community-based recommendation

Community-based RSs recommend items based on the preferences of the user's friends (Ricci et al., 2010). Evidence suggests that people tend to rely more on recommendations from their friends than on recommendations from similar but anonymous individuals (Sinha and Swearingen, 2001). These types of RSs acquire information about the social relations of the user and the preferences of his/her friends. Community-based recommendation can be viewed as a special CF, which only uses the ratings provided by the user's friends. These RSs follow the rise of social networking applications. Research in this area is still in its early phase.

3.6 Hybrid recommendation

The above techniques have some advantages and disadvantages. Table 1 compares them according to their information sources (input), advantages, and disadvantages.

Table 1. Comparison of different techniques (adapted from Bruke (2002)).

Technique	Information sources (input)	Advantages	Disadvantages
Content-based filtering (CBF)	1. the features associated with items 2. the ratings a user has given to items	a. domain knowledge not needed b. implicit feedback sufficient	A. quality dependent on large historical dataset B. overspecialization C. new user problem
explicit_CBF: Explicitly providing profile	1. the features associated with items 2. the profile explicitly provided by user	a, c. no historical dataset required	B, D. user must input their profile
Collaborative filtering	1. ratings for items from different users	a, b, d. can identify cross-genre niches	A, C, E. new item problem F. data sparsity
CF: Popularity-based	1. ratings for items from different users	a, b, d	E, G. non-personalized
Demographic recommendation	1. demographic information about a user 2. the features associated with items 3. knowledge about features of item and demographic information	a, c	H. must gather demographic information I. knowledge engineering required
Knowledge-based	1. user's need provided by the user via a series of dialogs 2. the features associated with items 3. Knowledge about how these items meet a user's need	c, e. can explain the reason for recommending a specific item	I, J. must gather user's need
Community-based	1. ratings for items from the user's friends	a, d	K. privacy concerns

As mentioned above, each RS technique has advantages and disadvantages. Hybrid RSs combine two or more of the above techniques. A hybrid system combining techniques A and B tries to use the advantages A to fix the disadvantages of B (Ricci et al., 2010). For instances, pure CF suffers from the cold-start problem (new item and new user), i.e., they cannot recommend items that have no ratings, and they cannot make recommendations to users who have not given ratings. These can be solved by applying a knowledge-based technique at the beginning. Burke (2007), and Adomavicius and Tuzhilin (2005) provided some surveys on hybrid RSs.

To summarize, different techniques lead to results with different qualities, and require different inputs (i.e., information sources). When choosing a suitable filtering method for a mashup application, it is important to consider each of the aspects in Table 1.

4. Icon placement and aggregation

The techniques described in section 3 can be used to filter irrelevant POIs, and thus reduce the number of icons to be displayed in a map. Reducing the number of POIs can greatly alleviate the problems of performance degradation, high transmission load, and "information overload". However, it may not solve all the overlapping problems (refer to Figure 2 for an example).

There is some research from cartographic generalization focusing on automatic symbol placement. The aim is to place symbols on maps while avoiding the overlap with other symbols and underlying map features. Many techniques have been developed for automatic symbol placement (Mackaness and Fisher, 1987; Harrie et al., 2004; Burigat and Chittaro, 2008; Kovanen and Sarjakoski, 2010). In the following, we focus on displacement and aggregation.

Displacement: In order to solve the problem of competition of limited map space between symbols or map features that overlap or lie too close to each other, the displacement operation is often applied. It shifts symbols or map features to other places to prevent coalescence. According to Foerster and Stoter (2008), displacement is the most important operation when considering how often the operation is applied and how dominant a role it plays. However, the displacement problem is Non-deterministic Polynomial-time hard (NP-hard) (Marks and Shieber, 1991). Many methods have been proposed for the displacement problem, e.g., Ruas (1998), Harrie (1999), Mackaness and Purves (2001), and Lonergan and Jones (2001). Most of them work in an iterative manner. Each iteration is composed of three phases: detection of conflict (overlap), calculation of a new location, and evaluation of the result (Mackaness and Purves, 2001). Multiple iterations are needed when new conflicts are created due to the previous iteration or when some constraint is still violated (Kovanen and Sarjakoski, 2010). In worst cases, the conflict problem cannot be solved at all.

In general, these displacement techniques proposed in the literature are also suitable for map-based mashups. However, two important issues have to be kept in mind. Firstly, the purpose of placing icons in a map is to show the existence of specific POIs at specific places. A POI icon usually has a predefined fixed location. As a result, the displacement movement of POI icons should be as small as possible. Secondly, as the map features of the underlying map such as roads and street cannot be detected or changed, displacement operation sometimes cannot solve the problem of poor legibility.

Aggregation (cartographic): The basic idea of aggregation operation is to identify clusters of mutually overlapping icons, and replace them with special aggregator icons (Burigat and Chittaro, 2008). The purpose of aggregator icons includes pointing out the presence of these icons as well as providing users with a means to access information about each of them. Burigat and Chittaro (2008) first created a conflict graph to store overlap relationship between icons, and then developed a maximum aggregation algorithm and a fast aggregation algorithm to identify a set of aggregator icons without conflicts. They added some mouse events to the aggregator icons. With the events, clicking an aggregator icon opens a pop-up window that lists the aggregated POIs and allows the user to obtain more information about each of them.

In addition to the visual overlapping aspect, aggregation can also be based on some semantic aspects, for example, categories and features of POIs. Therefore, more meaningful and semantic-enhanced cartographic aggregation can be provided.

5. Implementation, APIs and mashup examples

In this section, we discuss and analyze the implementation through existing APIs of the described methods in the previous sections. Two typical mashups that address the problem of icon cluttering are also introduced.

5.1 Implementation of irrelevant POIs filtering

As discussed in section 3, different filtering techniques can be used to reduce the number of POIs to be displayed in a map. All of them have some advantages and disadvantages, and may lead to results with different qualities. In terms of implementation, the needed technical skills are also different.

Technical speaking, explicit_CBF and simple demographic recommendation can be viewed as filtering by some querying parameters (such as category, location and language). There are some data APIs enabling developers to specify querying parameters. For example, Qype API can be used to query POIs near a certain position, search POIs in a category, and find POIs by their name or a keyword. YouTube data API defines different query parameters that can be used for filtering and ordering results, such as category, format, caption, language, and location. Yahoo! Local Search Web Service allows users to search the Internet for POIs near a specific location, as well as search by categories. These APIs and web services can be easily used to implement explicit_CBF and demographic recommendations. For knowledge-based filtering, the Filter Conditions C_F (i.e., rules) plays a key role. When these rules are identified, the above APIs and web services can also be employed to implement simple knowledge-based recommendation.

There are also some APIs and web services implementing popularity-based CF. For example, Qype API provides “order” to sort results by “distance” or “rating.” YouTube data API enables developers to order results by specifying an “orderby” parameter, such as relevance, published (chronological), viewCount, and rating. For Yahoo! Local Search Web Service, the “sort” parameter is used to order the results by the chosen criteria (e.g., relevance, title, distance, or rating).

Compared to the above four kinds of filtering methods, standard content-based filtering and standard collaborative filtering require more advanced programming skill to implement. Currently, there are few open APIs available on the web. To implement them, developers need to code the algorithms. To get implementation ideas, please have a look at some state-of-the-art surveys, such as Lops et al. (2010), and Desrosiers and Karypis (2010).

The community-based recommendation is a special collaborative filtering. As discussed in section 3.1, there are some open APIs for obtaining information from user's social networking accounts (Facebook, MySpace), such as OpenSocial API by Google, and Facebook Platform by Facebook. These APIs provide a basis for community-based recommendations by obtaining preferences from friends. Simple community-based recommendations may just display all the friends' favorite POIs, while complex community-based recommendations may also have to adopt the method of standard collaborative filtering. As a result, different programming skills may be needed.

To summarize, different methods lead to a different qualities of filtering, and require different technical skills to implement. When choosing a suitable filtering method for your application, it is critical to consider what you can get from the data sources. Candidate methods can be identified by checking whether the required inputs are available. When different filtering methods are feasible with the current data sources, the best method can be determined by analyzing the ease of implementation, advantages, and disadvantages of each method.

5.2 Existing APIs on icon displacement and aggregation

There are few open APIs available for icon displacement. To implement, developers should design their own algorithms. Implementation can be focused on the three components of icon displacement (detection of conflict, calculation of a new location and evaluation of the result).

For icon aggreagation in map-based mashups, there are some open APIs available. In the following, we discuss and compare some existing APIs designed for Google Maps API (v2 and v3).

MarkerManager: A utility library provided by Google for addressing the problem of slow rendering of the map and visual cluttering when adding a large number of markers to a Google map (GMaps Utility Library, 2011). When using MarkerManager, you have to define at which zoom-levels the marker will be visible. MarkerManager keeps tracking of all added markers, and shows them according to the defined zoom-levels. Recently, an updated version of the MarkerManager API is developed to support managing markers in Google Maps API v3 (MarkerManager v3, 2011).

Clusterer: Designed by Jef Poskanzer from ACME labs (ACME, 2011). Two techniques are implemented in Clusterer: only the markers currently visible actually are created; if too many markers would be visible, then they are grouped together into cluster markers. With these, adding thousands of markers to a map can also maintain decent performance.

ClusterMarker: Developed by Pearman (2011). It detects any group(s) of two or more markers whose icons visually overlap when displayed. Each group of overlapping markers is then replaced with a single cluster marker. The cluster marker, when clicked, simply centers and zooms the map in on the markers whose icons overlap. Currently, it only works for Google Maps API v2.

MarkerClusterer: Developed by Xiaoxi Wu and is part of the Google Maps Open Source Utility Library (Wu, 2011). It groups markers into different clusters and displays the number of markers in each cluster with a label, creating new clusters as the zoom level of the map changes. The clustering algorithm is simple: for each new marker it sees, it either puts it inside a pre-existing cluster, or creates a new cluster if the marker does not lie within the bounds of any current cluster.

Gabriel Svennerberg compared the above APIs on major browsers (Internet Explorer, Firefox, Google Chrome, Safari and Opera), and concluded that Clusterer was the fastest technique of them when only considering how long it takes before the markers are passed to the map (Svennerberg, 2009). However, when considering the actual time until all markers are visible on the map, MarkerClusterer is the fastest technique closely followed by ClusterMarker.

In the following, we analyze the functions provided by the above APIs, and their applicability in map-based mashups from the perspective of cartography. The MarkerManager only implements a Level of Detail (LoD) method of showing large number of icons. It can be easily incorporated with some techniques from cartographic generalization to provide comprehensive solutions for avoiding icon cluttering in map-based mashups. The last three APIs implement the cartographic aggregation function (section 4) which groups markers into different clusters to avoid icon cluttering. However, the clustering algorithms in them are simply based on the visual overlapping of icons. In order to provide meaningful and semantic-enhanced clustering, some other features of the icons (POIs) can also be considered, such as clustering overlapping icons according to their semantic categories.

5.3 Mashup examples

The following two map examples show, how uncluttered maps are provided in mashups. The example in Figure 6 combines information filtering techniques and icon aggregation techniques. POIs can be filtered by categories and keywords. It develops its own aggregation algorithm. Overlapping icons are aggregated into a translucent rectangle. When moving the mouse to the top of a rectangle, the number of icons in the rectangle is showed. Clicking the rectangle will show and zoom the map to the area of the rectangle.

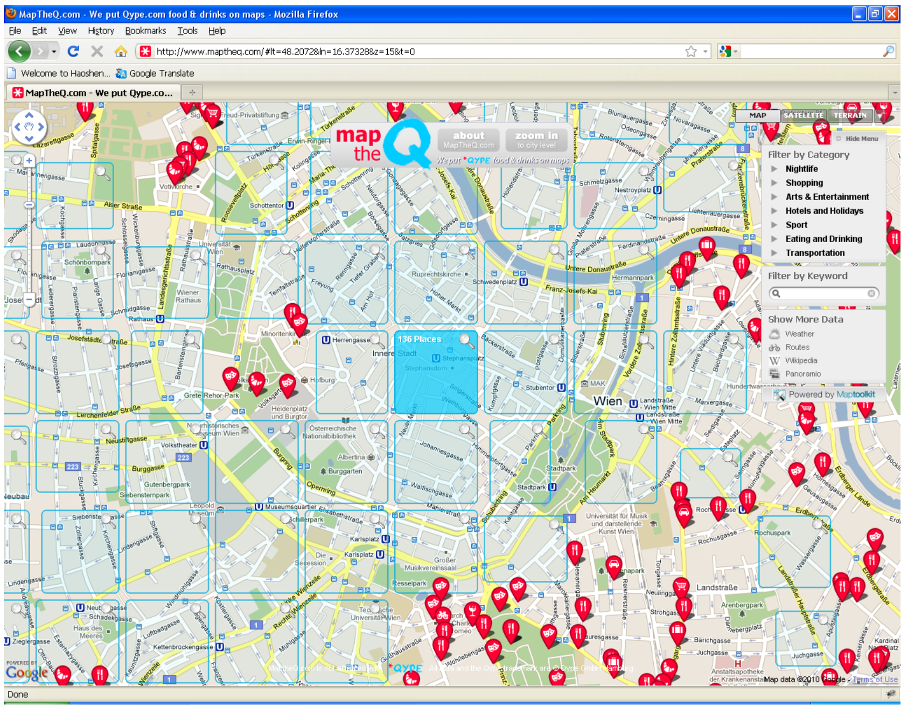

Figure 6. An example of decluttering icons in mashup taken from http://www.maptheq.com.

The mashup in Figure 7 also combines information filtering and icon aggregation. POIs can be filtered by categories. The MarkerClusterer API (see Section 5.1) is employed to aggregate icons in maps. The number of markers in a cluster is displayed on the cluster marker. Clicking the cluster marker shows the markers in this cluster.

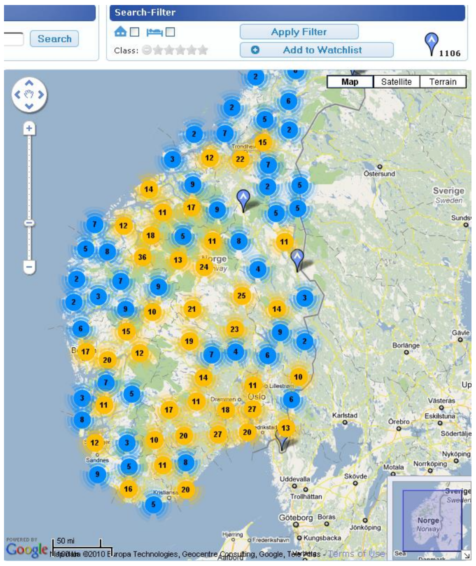

Figure 7. An example of decluttering icons in mashup taken from http://www.norwegen-reise.com. The MarkerClusterer API is employed to aggregate icons in maps.

It is important to note that, the above two examples only implement some very simple information filtering techniques. More information filtering techniques can also be implemented to filter irrelevant POIs, which will greatly alleviate the problems of information overload, and high transmission load. In addition, the clustering/aggregating algorithms in them are simply based on the visual aspect. More advanced techniques (e.g., semantic-enhanced aggregation and displacement) from cartographic generalization should be employed to address the icon cluttering problem.

6. Conclusions

Recent years have seen an increased interest in online map-based mashups. Visualizing large number of icons in a map often results in dramatic degradation of performance, high transmission load, information overload, and icon cluttering problems. In order to alleviate the above problems and provide a cartographically pleasing map, improving the methods of visualizing many icons in a map is becoming more and more crucial.

In this paper, we surveyed and assessed different techniques from computer science and cartography. Related techniques from information filtering were analyzed and compared for reducing the number of icons to be displayed in a map. Additionally, approaches for aggregating and placing icons from map generalization research were surveyed and analyzed for providing uncluttered maps.

Implementation using existing APIs of the described methods and some typical mashup examples were discussed and analyzed. The analysis showed that decluttering icons in map-based mashups is still in the early stage of development. Expertise from computer science and cartography should be seamlessly integrated to provide more cartographically pleasing maps in online map-based mashups.

References

ACME (2011): JavaScript Utilities. http://www.acme.com/javascript/#Clusterer. Accessed on June 2011.

Adomavicius, G. and Tuzhilin, A. (2005): Towards the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering, 17(6), 734-749.

Burigat, S., and Chitttaro, L. (2008): Decluttering of icons based on aggregation in mobile maps. In: Meng, L., Zipf, A. and Winter, S. (eds), Map-based Mobile Services – Design, Interaction and Usability, Springer, 13-32.

Burke, R. (2002): Hybrid recommender systems: Survey and experiments. User Modeling and User-Adapted Interaction, 12(4), 331-370

Burke, R. (2007): Hybrid web recommender systems. In: The AdaptiveWeb, Springer Berlin / Heidelberg, 377-408.

Desrosiers, C. and Karypis, G. (2010): A comprehensive survey of neighborhood-based recommendation methods. In: Ricci, F., Rokach, L., Shapira, B. and Kantor, P. (eds), Recommender Systems Handbook. Springer, 107-144.

Felfernig, A., Friedrich, G., Jannach, D. and Zanker, M. (2010): Developing constraint-based recommenders. In: Ricci, F., Rokach, L., Shapira, B. and Kantor, P. (eds), Recommender Systems Handbook. Springer, 187-215.

Foerster, T. and Stoter, J.E. (2008): Generalisation operators for practice: A survey at national mapping agencies. In: Proceedings of the 11th ICA workshop on generalisation and multiple representation, 20-21 June, Montpellier.

GMaps Utility Library (2011): http://code.google.com/apis/maps/documentation/javascript/v2/overlays.html #Marker_Manager. Accessed on June 2011.

Haklay, M., Singleton, A. and Parker, C. (2008): Web mapping 2.0: The neogeography of the geoweb. Geography Compass, 2(6), 2011-2039.

Hanani, U., Shapira, B. and Shoval, P. (2001): Information filtering: Overview of issues, research and systems. User Modeling and User-Adapted Interaction, 11(3), 203-259.

Harrie, L. (1999): The constraint method for solving spatial conflicts in cartographic generalisation. Cartography and Geographic Information System, 26(1), 55-69.

Harrie, L., Stigmar, H., Koivula, T. and Lehto, L. (2004): An algorithm for icon placement on a real-time map. In: Fisher, P. (ed), Developments in Spatial Data Handling, Proceedings of the 11th International Symposium on Spatial Data Handling, Springer, Leicester, 493-507.

Kovanen, J. and Sarjakoski, L.T., (2010): Displacement and grouping of points of interest for multi-scaled mobile maps. In: Proceedings of LBS 2010, Guangzhou, 20-22 September 2010.

Lonergan, M. and Jones, C.B. (2001): An iterative displacement method for conflict resolution in map generalization. Algorithmica 30, 287-301

Lops, P., de Gemmis, M. and Semeraro, G. (2010): Content-based recommender systems: State of the art and trends. In: Ricci, F., Rokach, L., Shapira, B. and Kantor, P. (eds), Recommender Systems Handbook. Springer, 73-105.

Mackaness, W.A. and Fisher, P.F. (1987): Automatic recognition and resolution of spatial conflicts in cartographic symbolisation. In: Proceedings of AutoCarto 8, 29.03-03.04, Baltimore, USA, 709-718.

Mackaness, W.A. and Purves, R.S. (2001): Automated displacement for large numbers of discrete map objects. Algorithmica, 30, 302-311.

MarkerManager v3 (2011): http://google-maps-utility-library-v3.googlecode.com/svn/tags/markermanager/1.0/ docs/reference.html. Accessed on June 2011.

Marks, J., and Shieber, S. (1991): The computational complexity of cartographic label placement. Technical Report TR-05-91, Center for Research in Computing Technology, Harvard University.

Morandell, C. (2010): Möglichkeiten der nutzerspezifischen Gestaltung von Location Based Services mit Daten aus Social Networks. Master thesis of Vienna University of Technology.

Pearman, M. (2011): Google Maps API Projects. http://googlemapsapi.martinpearman.co.uk/readarticle.php? article_id=4. Accessed on June 2011.

ProgrammableWeb (2011): http://www.programmableweb.com/mashups#topt-2. Accessed on June 2011.

Ricci, F., Rokach, L. and Shapira, B. (2010): Introduction to recommender systems handbook. In: Ricci, F., Rokach, L., Shapira, B. and Kantor, P. (eds), Recommender Systems Handbook. Springer, 1-35.

Ruas, A. (1998): A method for building displacement in automated map generalisation. International Journal of Geographic Information Science, 12(8), 789-803.

Sinha, R. and Swearingen, K. (2001) Comparing recommendations made by online systems and friends. In: DELOS Workshop: Personalisation and Recommender Systems in Digital Libraries.

Svennerberg, G. (2009): Handling Large Amounts of Markers in Google Maps. http://www.svennerberg.com/2009/01/handling-large-amounts-of-markers-in-google-maps/. Accessed on June 2011.

Wang, J., Vries, A. and Reinders, M. (2006): Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In: Proceedings of the 29th ACM SIGIR Conference on Information Retrieval, 501-508.

Wikipedia (2011a): http://en.wikipedia.org/wiki/Web_2.0. Accessed on June 2011.

Wikipedia (2011b): http://en.wikipedia.org/wiki/Mashup_%28web_application_hybrid%29. Accessed on June 2011.

Wikipedia (2011c): http://en.wikipedia.org/wiki/Information_filtering_system. Accessed on June 2011.

Wikipedia (2011d): http://en.wikipedia.org/wiki/Information_overload. Accessed on June 2011.

Wu, X. (2011): MarkerClusterer: A Solution to the Too Many Markers Problem. http://googlegeodevelopers.blogspot.com/2009/04/markerclusterer-solution-to-too-many.html. Accessed on June 2011.