University of Kaiserslautern
Department of Computer Science
Research Group Knowledge-Based Systems
Prof. Dr. Andreas Dengel

Crumblr: Personalized Recommendations of Shared Spatial Content in Mobile Environments

– Diploma Thesis –

Dragan Šunjka

September 2008

Advisor: Dipl.-Inf. Darko Obradović
Examiner: Prof. Dr. Andreas Dengel

Erklärung

Ich versichere hiermit, dass ich die vorliegende Diplomarbeit mit dem Thema "Crumblr: Personalized Recommendations of Shared Spatial Content in Mobile Environments" selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen wurden, habe ich durch die Angabe der Quelle, auch der benutzten Sekundärliteratur, als Entlehnung kenntlich gemacht.

Dragan Šunjka

Crumblr: Personalized Recommendations of Shared Spatial Content in Mobile Environments

Dragan Šunjka

September 2008

Abstract

Recently, a convergence of mobile computing technologies and the Internet is apparent. On the one hand, this trend leads to new opportunities for providing spatial assistance to users on the move. On the other hand, while engaging in everyday activities, people develop personal preferences about visited places and routes. By observing people's spatial behavior via mobile location sensing technology, both user preferences and geographic characteristics of places and routes can be implicitly captured, further improving the assistance process.

This thesis proposes a novel approach to collect and disseminate shared spatial content by employing semi-automatic capture of spatial behavior, aggregation of spatial and contextual data, and personalized recommendation and visualization techniques adapted to mobile scenarios. To demonstrate the approach, a prototype has been developed for the Google Android platform for mobile devices.

Acknowledgements

I would like to acknowledge all the people who supported me during this project. A special thanks goes to my supervisor Darko Obradović for his incredibly quick and valuable feedback on all levels which improved this work. I would also like to thank the DFKI in general, especially Stephan Baumann and Darko, for fully supporting my ideas behind Crumblr right away, giving me the opportunity to conduct a thesis in this vibrant domain.

I want to thank my friends Jan and Matthias for the numerous discussions, many constructive comments, and the mutual support while writing our theses. I also want to thank them for the great time we had during our studies in Kaiserslautern.

I would also like to cordially thank my friends Adam, Jan "2.0", Stefan, Florian, Roger, and Rafael for various support during my thesis work and making the project a very pleasant one. A big thanks goes to my flatmate and friend Peter for his constructive comments and feedback as well.

Finally, I also want to thank my family members and my girlfriend Gorica for their support and understanding, especially in the last few months.

Contents

Chapter 1

Introduction

Foreword

The Internet is experiencing exponential growth and global expansion. This has led many people to believe that the Internet is ushering in a new era, the information age, and a new social form, the information society. Web 2.0 applications act as platforms creating collaborative, community-based sites, where users provide and organize the content.

Furthermore, globalization is forcing us to live faster lives, dominated by the need for mobility. Mobility consequently leads to more people traveling and moving in unfamiliar areas. Realizing the modern accelerated lifestyle requires supporting tools and information. This especially holds for spatial information, since it is attached to almost any everyday activity. The rapid spread of laptops, mobile phones and other mobile devices seems to lead to a new technological milestone – the convergence of the Internet and the cellular telephone. The trend to mobility has also brought about a whole plethora of new possibilities for spatial assistance in mobile environments. Supporting tools and mobile technology that are aware of the user's current situation and interests can assist the user, presenting up-to-date spatial information in an individual and flexible way.

On the one hand, a large quantity of spatial data is produced by people engaging in various everyday activities. People's spatial behavior can be observed and analyzed for two purposes: capture of geographic characteristics of places and routes, and acquisition of user preferences. The spatial behavior can be recorded via ubiquitous mobile technology in an unobtrusive way. On the other hand, mobile recommendation systems can benefit from this data in order to deliver personalized spatial recommendations to users on the move.

This thesis proposes a novel context-driven approach to capture and use observed spatial behavior: *Crumblr*. The approach employs a life cycle consisting of three main steps: user observation, aggregation of spatial and contextual data, and personalized recommendation of spatial data. Emphasis is put on supporting everyday activities of mobile users by offering relevant spatial information for quick decisions. The recommended items are backed up by explanations derived from the utilized recommendation model. In order to demonstrate the approach, a proof-of-concept prototype has been developed for the Google Android mobile platform using open-source software solutions.

Thesis Structure

The thesis is structured as follows (see Figure 1.1):

Chapter 2 develops the theoretical and technical background of the thesis. The trends and technologies behind Web 2.0, mobile devices and context-aware systems are described. Finally, the characteristics and the potential of user-generated spatial content are outlined.

Chapter 3 reviews related approaches for capturing and disseminating content and user preferences about places and routes. An evaluation of existing work reveals potentials for improvements, further motivating this thesis.

Chapter 4 introduces *Crumblr*, a novel concept for capturing, aggregating, and disseminating spatial content and user preferences on mobile devices. After a discussion of the rationale and ideas behind it, the visual appearance of the implemented prototype is presented via user stories.

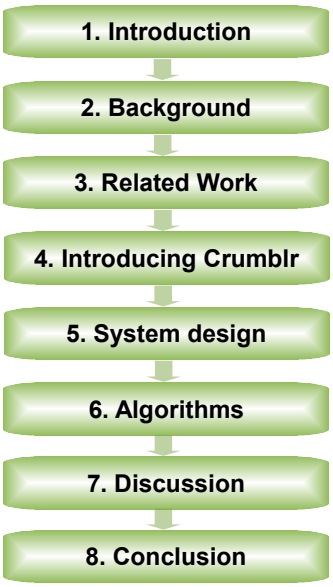


Figure 1.1: Thesis structure

Chapter 5 turns to the aspects related to system design and technological decisions.

Chapter 6 puts the focus on the implemented algorithms for data collection, aggregation, recommendation and explanation. Related work is presented for each algorithm class in question, providing a rationale for the chosen approaches.

Chapter 7 provides some critical remarks, identifies open issues, and discusses general ideas about possible future work.

Chapter 8 closes with a conclusion.

Chapter 2

Background

Today's most popular Web applications and services such as blogs, video sharing and social networking platforms act as collaborative, community-based sites where users provide and organize the content. Given the high adoption rate of mobile devices and the trend to open systems, a new generation of mobile Internet applications begins to gain momentum – location-based services. Recently, a myriad of such services is targeting the area of capturing and disseminating information about spatial entities such as places and routes. To create an effective user experience and provide real added value to the user, developers are facing a series of challenges when designing such information systems for mobile devices. To address the problem of information overload, personalized recommendations based on the user's current situation and preferences are necessary.

This chapter describes these trends in more detail and paves the way for the core work in this thesis, providing a motivational background and positioning it in the related fields of research.

2.1 The Socio-Technological Evolution of the Web

This section looks back at the history of the Internet, outlines the factors that led to a 'new', 'user generated', 'social' version of the Web, and gives an overview of the key principles behind it.

The Early Beginnings

The idea of Social Software dates back to the 1960s and JCR Licklider's thoughts on using networked computing to connect people in order to boost

their knowledge and their ability to learn [Alexander, 2006]. Sir Tim Berners-Lee initially had a similar vision of the World Wide Web when he invented it at CERN in 1991. However, the history of the Web shows that it was not reflecting Berners-Lee's idea of a single, global information space in it's early years [Berners-Lee, 1999].

It is interesting to note that the first Web client built by Berners-Lee, WorldWideWeb [Berners-Lee, 1990], could not only view HTML pages, but also *edit* them. This technology was supposed to spark a participative architecture of the Web where the content would be truly user-generated, enabling seamless information sharing and linking of people. However, to speed up the process of adoption within CERN, a series of ports to other machines and platforms led to the removal of the ability to edit pages through the Web client [Berners-Lee, 1999]. Another possible part of the explanation could be the fact that the HTTP PUT and POST methods had not yet been implemented on the server side — the first protocol revision to specify these and other methods was published in 1996[1]. This had the consequence that files could be edited in a local file system only, which in turn had to be copied onto a HTTP server manually. Subsequently developed browsers like Mosaic (later: Netscape) shaped the image of the Web as a 'lecture' style platform where few 'privileged' people published and the majority only consumed the content.

Around 2003, the way people and businesses were using the web started to shift noticeably. The rising popularity of applications and services such as blogs, video sharing and social networking platforms has changed the way we interact and network. These applications are the main concentration points of Media coverage about something called *Web 2.0* – a social Web in which people both contribute and consume. Web 2.0 applications act as platforms creating collaborative, community-based sites where users provide and organize the content.

Advancing to the Digital Age

Several economic and technological advancements accelerated the adoption of the new approaches gathered under the label Web 2.0 [Deitel et al., 2007]:

Powerful, cheap hardware — According to Moore's Law the power of hardware doubles every two years, while the price remains essentially the same [Moore, 1965]. Very powerful hardware became widely affordable.

[1] http://www.ietf.org/rfc/rfc1945.txt

Broadband Internet access — Instant access, fast speeds, and simple connections have contributed to the explosion of broadband high-speed Internet. Of the 35% of Internet users who had posted content online, 73% had broadband Internet [Horrigan, 2006]. A multitude of the digital media could not have been uploaded without broadband access.

Open Software — The cost of starting and failing a new business on the Web has been drastically lowered due to a plurality of available, often free, open source software. Those include various developer toolkits, such as the LAMP stack[2].

Accessible business models — Using business models like affiliate programs, advertisements, virtual world monetanization, viral marketing, and peer-production methods, people can start earning modest amounts of money quickly.

These developments provided a foundation for a participative architecture of the Internet as we know it today. The next section sheds more light on the term Web 2.0 and outlines the key principles associated with it.

What is Web 2.0

The term Web 2.0 was coined by Dale Dougherty of O'Reilly Media in 2004 and was further made popular by Tim O'Reilly [O'Reilly, 2005]. Web 2.0 has become a buzzword that is used to describe a wide range of community-based online applications. These applications and services are early manifestations of ongoing Web technology development and evolution of the Web [Anderson, 2007].

When Sir Tim Berners-Lee was asked in an interview whether Web 2.0 was different to what might be called Web 1.0, he replied:

> *"Totally not. Web 1.0 was all about connecting people. It was an interactive space, and I think Web 2.0 is of course a piece of jargon, nobody even knows what it means. If Web 2.0 for you is blogs and wikis, then that is people to people. But that was what the Web was supposed to be all along. And in fact, you know, this Web 2.0, it means using the standards which have been produced by all these people working on Web 1.0."* [Laningham, 2006]

[2]Linux, Apache, MySQL, and PHP

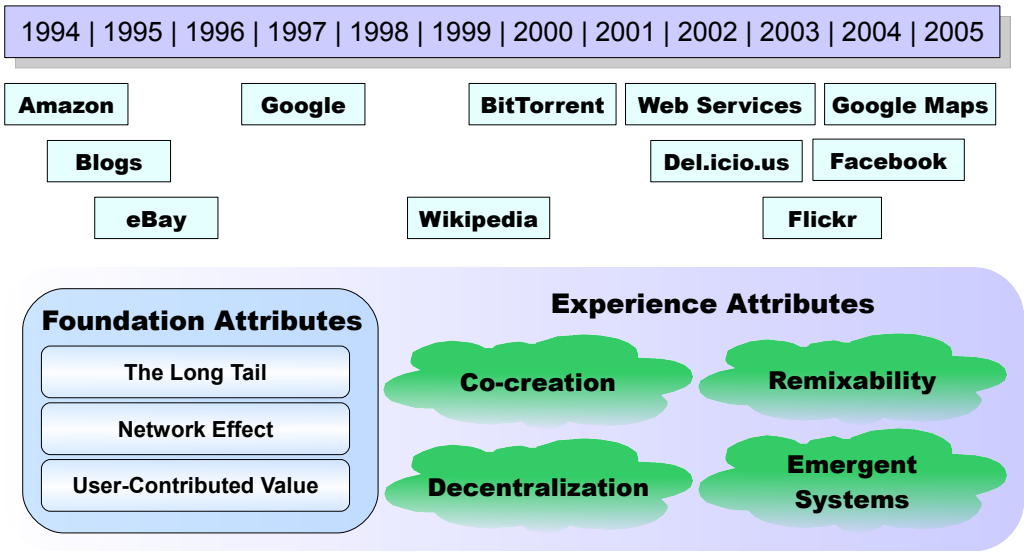


Figure 2.1: Web 2.0 – key principles and applications

Many of the typical Web 2.0 applications are mature, having been in use for a number of years. Some of the most prominent examples of Web 2.0 services and applications are blogs, wikis, social networking and multimedia sharing sites. Basic familiarity with these applications is assumed and they will not be introduced here.

Tim O'Reilly and his colleagues at O'Reilly Media tried to identify the key concepts around the Web 2.0 hype in [O'Reilly, 2005]. Brandon Schauer from Adaptive Path[3] went one step further and provided another perspective on the principles behind Web 2.0, bringing further structure into this phenomenon [Schauer, 2005]. Schauer has identified two layers of attributes that characterize Web 2.0 applications – the Foundation Attributes and the Experience Attributes. Summarizing his work, an overview of the identified attributes with a chronological set of popular Web 2.0 applications is given in Figure 2.1.

Foundation Attributes frame the economic model of Web 2.0 services, pre-dating other attributes by several years. Moreover, these are not identified as key Web 2.0 principles since many non-Web 2.0 services also utilize them (e.g. e-mail and bulletin boards). The following list describes the Foundation Attributes.

[3] `http://www.adaptivepath.com`

User-contributed value — Where O'Reilly talks about user-generated content in general, Schauer differentiates two levels of user interaction. User-contributed value represents the basic level while "co-creation" (explained later) builds on it. In the early beginnings of user-generated platforms, users' contributions were substantial for the overall value of a service. The best example for this concept are blogs.

The Long Tail is a very common business model on the Internet. On sites like Amazon[4] and eBay[5], the total sales volume of low popularity items exceeds the volume of high popularity items. Brynjolfsson, Hu, and Smith found a large proportion of Amazon.com's book sales come from obscure books that are not available in brick-and-mortar stores [Brynjolfsson et al., 2003]. Wired Editor-in-chief Chris Anderson has explained the term 'long tail' as a reference to the tail of a demand curve[6].

Network effect — The value of the network substantially increases with each new user. eBay, for example, relies on the community to buy and sell auction items, and Monster[7] (a job search engine) connects employers with job seekers.

Experience Attributes build upon the Foundation Attributes, delivering unique service experiences that were previously undeliverable. Surfaced around 1999/2000, the Experience Attributes identified by Schauer are described in the following list:

Decentralization in this context means that there is no central authority having control over user experiences. A prominent example is the BitTorrent network [Cohen, 2003]. It is basically an ad-hoc network of peers sharing content with each other.

Co-creation represents Schauer's second level of user interaction, based on user-contributed value. The concept states that collaboration can result in smart ideas. By working together, people can combine their individual knowledge for everyone's benefit. When the U.S. submarine Scorpion sank in 1968, the Navy asked various experts to work individually assessing what might have happened; their answers were then collectively analyzed, determining the accurate location of the submarine [Surowiecki, 2005]. A nice example for a Web 2.0 application

[4] `http://www.amazon.com`
[5] `http://www.ebay.com`
[6] `http://www.npr.org/templates/story/story.php?storyId=4156078`
[7] `http://www.monster.com`

utilizing the principle of co-creation is a spam filter based on "collaborative filtering". The basic idea is to let the users act individually to combine their results and collectively decide what is spam and what not. Such a system would outperform systems that rely on analysis of the messages themselves [O'Reilly, 2005]. Another prime example are wikis.

Remixability relates to the possibility of creating tailored user experiences by combining several Web services and data sets. Examples for reusable services are the Amazon Web API and a myriad of Google APIs.

Emergent systems provide added value to the overall experience by aggregating the results of cumulative user actions at the lowest levels. Examples are "folksonomies", as used in Flickr[8] and Del.icio.us[9]. The system derives value from people using their own vocabulary in order to add explicit meaning to the information or object they are consuming. This also includes implicitly user-generated content that is based on users' online behavior. For example, Amazon tracks the individual's purchase history and compares it to purchases of other users with similar tastes to make personalized recommendations, creating additional value for users.

Recently, there has been an explosion of services focusing on handling the vast amount of user-generated content to prevent information overload. So-called "aggregation services" and "mash-ups" such as RSS aggregators and countless spatial applications based on the Google Maps API collect and aggregate user data, creating tailored, relevant presentations of content to meet specific needs. The explosion of user-generated content leads to a growing need and popularity of such services. The following sections present further forces sparking that need.

2.2 Proliferation of Mobile Devices

As of this writing, there are nearly three billion wireless mobile phone subscribers worldwide and that number is growing rapidly. By 2003 more people had mobile subscriptions than subscriptions to traditional land lines [Lassica, 2007].

Day by day, the number of new users is adding up to Nokia Siemens Networks' (NSN) vision of 5 billion people connected by 2015 [NokiaSiemen-

[8] http://www.flickr.com
[9] http://del.icio.us

sNetworks, 2005]. The capabilities of mobile telephony are used by both rich and poor, educated and uneducated alike. About 90% of new users of mobile phones will be in emerging markets such as Asia, South America and Afrika [NokiaSiemensNetworks, 2005]. According to Veli Sundbäck, Executive Vice President at NSN, several factors have contributed to the fast-paced proliferation of mobile phones:

- In emerging markets like India, South Africa, Morocco and some African countries governments are speeding up the adoption process by increasing regulation and providing an enabling environment for mobile operators. For instance, 26% of the owners of mobile phones in Kenia use their phones for both business and as a source of news and information [NokiaSiemensNetworks, 2007].

- Second, high investments in research and development by the mobile phone industry rapidly led to affordable prices for mobile phones. The devices are also becoming smaller and less bulky.

- Finally, apart from being affordable, the mobile technology is fully utilizing the network effect in the civil society. The mobile phone has become an indispensable part of everyday life providing access to a variety of services.

Nowadays, the average consumer is not constrained to person-to-person voice calls and short text messages (SMS). Accessing wireless local area networks, browsing the World Wide Web, social networking, fully embracing all forms of participative media – the advanced functionalities of mobile devices are shaping the behavior of the emerging Mobile generation [Lassica, 2007].

While the Web stands for anytime and anywhere access, a mobile device brings these concepts to the next level. Why wait until you get home to log on to the PC to tell your closest friends about a nice restaurant you just have discovered? The Internet has become more than just a PC-based phenomenon. A mobile device seems to be the natural extension to the traditional Web [Toivonen, 2007].

Besides the positive effects on the use of wireless Internet, the mobile phone has disadvantages, as well. Foremost, it has a form factor designed for portability. Portability, however, limits the user experience: small display, limited input mechanisms, short user attention spans, and relatively short battery lifetime present additional challenges for system developers. Software development efforts must account for these new types of constraints in order to deliver an effective and satisfying mobile experience.

2.3 Context-Aware Mobile Services

As the owner of a mobile device is trackable utilizing location sensing technologies, the user's frequently changing location provides both a challenge and an opportunity to fulfill the need for highly personalized systems.

Location is just one category of user *context* that can be important. One of the most adopted definitions of context in the field of context-awareness is the one from Abowd et al.:

> *Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.* [Abowd et al., 1999]

More specific context definitions were suggested, e.g. by Schilit et al. [Schilit et al., 1994], who stated that context could be divided into three categories: the computing context (such as network connectivity, communication bandwidth and nearby resources such as printers and displays), the user context (such as the user's profile, location, people nearby and the current social situation), and the physical context (lighting, noise levels, traffic conditions and temperature). Chen and Kotz additionally proposed the time context (such as time of day, week, month and season of the year) and context history, which could be useful information in map applications [Chen and Kotz, 2000].

In [Nivala and Sarjakovski, 2003], Nivala and Sarjakovski developed a classification of context types with specific reference to mobile services that are map-based. Their results are summarized in Table 2.1.

Systems that dynamically change their behavior based on context are called context-aware, reactive and situated [Abowd et al., 1999]. In mobile cartography, the term *adaptive* has become the most commonly used [Reichenbacher, 2004]. [Reichenbacher, 2003] has explored the various kinds of adaptation, resulting in four different levels:

1. Information level: for instance, changing content by filtering information with respect to proximity to a user or place.

2. Technology level: encoding information to suit different device characteristics, e.g. display size, network and positioning availability.

3. User interface level: for example, panning and re-orientating a map as the user moves about.

Context type	Properties
Mobile user	The identity of the user might allow the service to consider such issues as their age, gender, and language.
Location and time	Allows information and services to be localized. Time can refer to the precise time of day or longer intervals such as morning, afternoon, etc.
Orientation	In a navigation service it is important to check the user is heading in the right direction, a tourist guide might determine what historical building the user is facing, etc.
Navigation history	Can help to build up a profile of user interests, enhancing the provision of relevant information.
Purpose of use	Defined by activities, goals, tasks and roles.
Social situation	Characterized by user's proximity to others and social relationships.
Environment	Can include such things as the lighting level or how much ambient noise there is.
System properties	Everything related to the computer infrastructure the user is employing falls under this category: Internet connectivity, communication bandwidth, the type of device and its capabilities (touch screen, color), etc.

Table 2.1: Categorization of context categories for mobile map services [Nivala and Sarjakovski, 2003]

4. Presentation level: the visualization of the information is adapted. Restaurants that are more relevant to a user are shown with more crisp icons and those less relevant use more opaque ones, for example.

Location as a core component of mobile user context has proven to be useful in many case studies and existing applications [Strahan, 2002]. It yields a new and rich modeling dimension that needs to be considered when designing mobile applications in order to provide real added value to the mobile user.

For example, when a person is on the move and wants to have lunch in

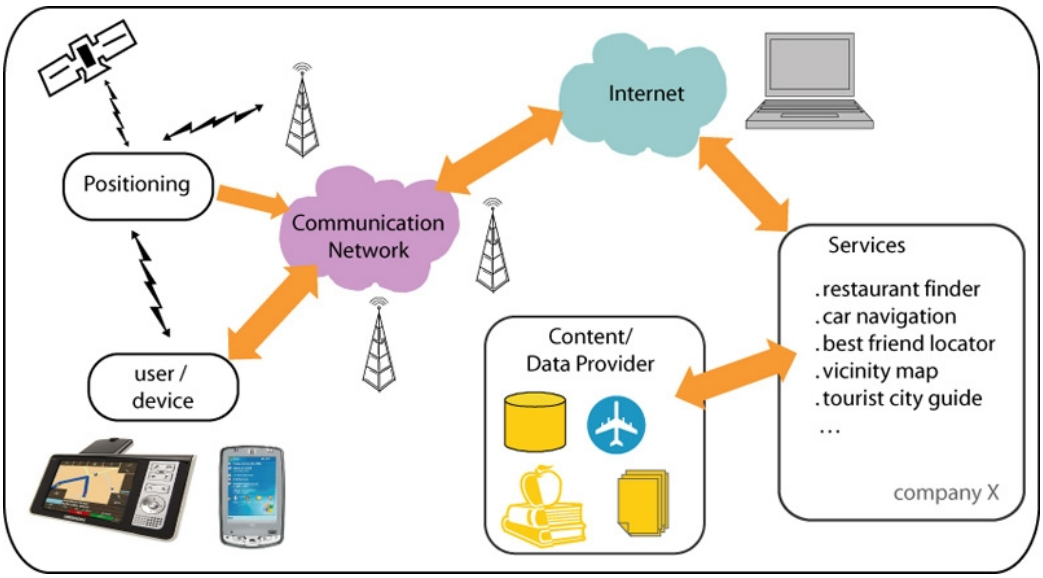


Figure 2.2: LBS components and information flow [Steiniger et al., 2006]

a restaurant in their vicinity, a useful approach would restrict the search in the Internet by adding several constraints, e.g. where is the user (current location) and what kind of restaurant they would like to go to (purpose of use). Such kind of search could be performed using a *location-based service* (LBS):

> *Location-based services are information services accessible with mobile devices through the mobile network and utilizing the ability to make use of the location of the mobile device.* [Virrantaus et al., 2001]

A LBS is typically comprised of five basic components and their connections shown in Figure 2.2, which are explained below:

Devices represent tools for users to request the needed information (e.g. a restaurant recommendation). Possible devices include mobile phones, PDAs and laptops.

Positioning Component is an implementation of a location sensing technology, e.g. GPS. Nowadays, this component is usually physically integrated into the mobile device. Location sensing technologies are described in more detail in section 2.4.

Communication Network and Internet transfer the user data and service requests from mobile devices to the service provider and then the information back to the user.

Service and Application Providers are responsible for processing service requests from users. Offered services include searching yellow pages, calculating routes, recommending places.

Data and Content Providers store and maintain all the data that is used by Service Providers. This includes geographic base data, yellow pages and traffic data.

Many mobile devices are being designed with a geographic location tracking technology for reasons of safety, finding travel destinations, etc. The telecommunications industry has just started to offer LBS to subscribed users [Reichenbacher, 2004]. LBS assist users in mobile environments by being aware of the user's location context, behaving adaptive in the aforementioned sense. Existing applications include emergency services (for calling fire-fighters, medical teams, police), navigation services, travel and tourist guides, billing services, and tracking and fleet management services [Steiniger et al., 2006].

2.4 Location Sensing Technologies

Location-based services require location sensing capabilities in order to work. There exists a variety of different location sensing technologies with significantly different characteristics, different infrastructure and device requirements as well as different cost and limitations. A comprehensive description of characteristics of all available technologies is out of the scope of this thesis. Instead, this section briefly introduces some of the most popular location sensing approaches and systems.

Cell Identifier

The mobile phone network consists of a mosaic of overlapping radio cells (cf. Figure 2.3[10]), each determined by a base station having a radio antenna installed [Siemens, 2001]. Each cell is identifiable by its unique cell ID. The user's position can be determined by simply looking up the stored location of the associated cell ID. Since it does not require any modifications to the

[10]Picture taken from `http://en.kioskea.net/` (Creative Commons License 2.0).

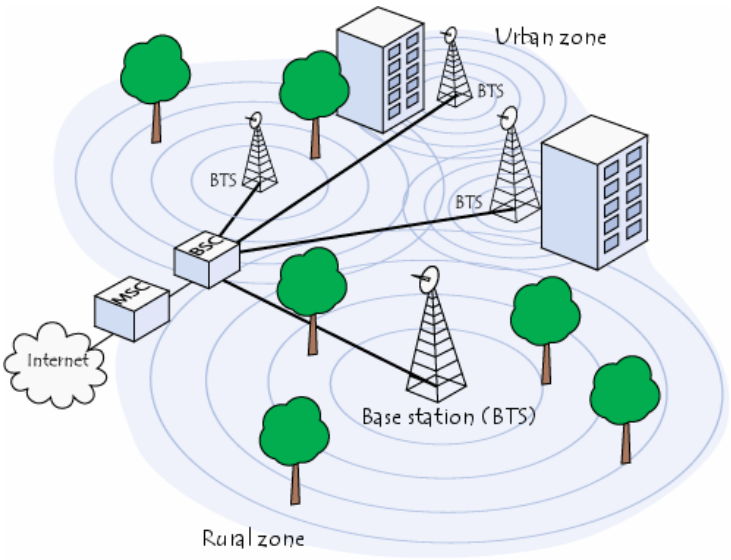


Figure 2.3: Mobile phone network cells

mobile phone nor the network, this is the simplest location sensing technology in terms of deployment cost. The accuracy, however, highly depends on the concentration of cells. The smaller the radius of a cell, the higher is the available bandwidth. So, in highly populated urban areas, there are cells with a radius of a few hundred meters, while huge cells of up to thirty kilometers provide coverage in rural areas. For emergency or navigation services such accuracy is insufficient [Siemens, 2001].

Network-based Location Sensing

There is a large amount of research work focusing on physical parameters that can be derived from the network signal transmission characteristics, such as signal propagation time, signal field strength and angle of arrival [Pandey and Agrawal, 2006]. Using such signal characteristics, the positions of the individual nodes in the communication network can be estimated. Related approaches are outlined in the list below.

Angle of Arrival (AoA) is a method for measuring the direction of an incoming radio-frequent wave and obtaining a position estimate of the source (the mobile phone) by calculating the intersection of apparent arrival directions. This method requires two receiving base stations — each additional base station increases the accuracy, however. Special antenna arrays in base stations are required for AoA calculation.

Received Signal Strength Indicator (RSSI) can be translated into distance using a theoretical or empirical model. For example, the signal strength declines with increasing distance between communicating parties. The RSSI method is susceptible to irregular signal propagation characteristics like fading, interference and multi-path effects.

Time of Arrival (ToA) is another technique used to obtain range information. Distance estimation is performed by measuring the propagation time from source to destination. At least three highly synchronized base stations must receive the signal from the phone to be able to apply trilateration and estimate positions.

Time Difference of Arrival (TDoA) requires three base stations to calculate the relative differences of the received signal from the phone to estimate its position. Strict time synchronization is needed.

The operating system used in the first-generation Apple iPhone triangulates the user's position based on signal strength measurements, for example. Nevertheless, the described network-based techniques require high infrastructure maintenance, increased network signaling, high up-front costs and non-standard hardware. They have not yet reached a level of maturity and accuracy required for deployment in large cellphone networks [Strahan, 2002]. The second generation of Apple's iPhones (iPhone 3G) relies on the Global Positioning System, which is introduced next.

Global Positioning System (GPS)

The Global Positioning System (GPS) is a widely accepted system for outdoor navigation. It was launched 1993 by the US government. Since then, a constellation of more than two dozen GPS satellites broadcasts precise timing signals by radio to electronic GPS receivers which allow them to accurately determine their location in real time. A GPS receiver calculates its current position by employing a triangulation process on the received signals from several satellites. It has an accuracy of about ten meters and requires line-of-sight connection between the mobile device and at least three satellites to function [Bajaj et al., 2002].

Improvements to GPS

Several enhancements have been made to GPS. Two of the most widely accepted improvements will be outlined here - Differential GPS and Assisted GPS.

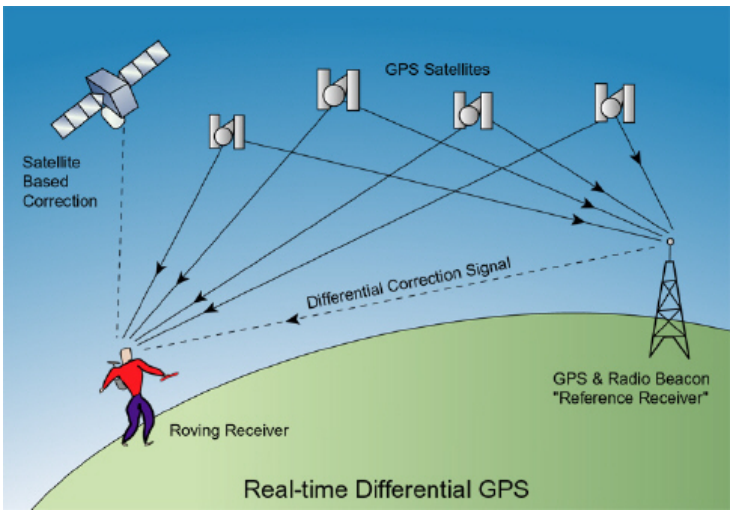


Figure 2.4: Differential GPS [Pendleton, 2002]

When the US Department of Defense made GPS available to consumers, it was intentionally downgraded by imposing a special system mode called "Selective Availability". To deal with the resulting inaccuracies, a group of engineers came up with "Differential GPS" (see Figure 2.4). It is an enhancement to GPS that uses a network of fixed ground stations which are providing correction information to moving receivers. Today, even with Selective Availability turned off, it currently provides a great means of inexpensive, high-accuracy positioning, usually providing results better than pure GPS. Differential GPS can improve accuracy to one meter or better [Bajaj et al., 2002]. Its drawback is the high dependence on a few ground stations and the increased network signaling.

When surrounded by tall buildings or being under trees, conventional GPS might have difficulties providing reliable positions due to poor signal conditions. Additionally, when first powered on under such conditions, a traditional GPS receiver might be blocked for several minutes trying to receive all necessary data from satellites. Assisted GPS describes a system that is trying to solve these problems. An outside source, such as an assistance server and reference network, helps the GPS receiver perform the tasks required to make position calculations [Djuknic and Richton, 2001].

Several outdoor location sensing approaches have been outlined in this section. Offered free of charge and accessible worldwide, GPS is rapidly becoming a universal positioning utility as the cost of integrating the technol-

ogy into vehicles, machinery, computers, and cellular phones decreases [Bajaj et al., 2002]. It has become the de facto standard for mobile devices. Several popular mobile devices support Assisted GPS. These include the Apple iPhone 3G, several Nokia, HTC and Sony Ericsson models.

2.5 Open Mobile Systems

While todays LBS applications have become fairly popular in certain user segments, the real success has yet to materialize [inCode Wireless, 2005]. This section provides a summary of the reasons why LBS have not turned out as the expected success. Trends and developments are outlined that promise to accelerate the adoption of LBS, as well.

The foremost reasons for the lack of success of LBS in its current format are summarized in the list below.

- Software performance and usability: although the functionality is there, most services are simply too slow and too cumbersome to use. This is driven both by poor user interface design and poor browser performance in most cell phones [inCode Wireless, 2005].

- Closed systems: with nearly 3 billion users worldwide, the mobile phone has become the most personal and ubiquitous communications device. Proprietary protocols and systems, restricted network functionality, and the lack of a collaborative effort has made it a challenge for developers, wireless operators and handset manufacturers to respond as quickly as possible to the ever-changing needs of savvy mobile consumers [Open Handset Alliance, 2008]. Market analysts agree that this is more of a struggle of the carriers trying to keep control of some of the revenue source. However, they are being forced to open their networks more and more in order to compete, opening doors for companies like Yahoo and Google to take a considerable share of the revenue [Informa Telecoms & Media, 2006].

- Network quality: for many consumers, the early days of mobile Internet and location-based services were filled with great potential, but experiences were limited by network quality [inCode Wireless, 2005].

The hurdles seem to be diminishing, however. 3G (third generation) networks bring tremendous improvements to the consumer experience of mobile Internet. 3G networks, which first launched in Japan in 2001, are now becoming more ubiquitous in the US and Europe, significantly enhancing the

consumer mobile data experience [Nielsen Mobile, 2008]. The buzzword of the years 2007 and 2008 might be "open" – organizations including the LiMo Foundation, the Open Handset Alliance, and AOL have launched open mobile platforms and services.

In January 2007, the LiMo Foundation was founded. Its goal of establishing a globally competitive, Linux-based software platform for mobile devices has been endorsed by leading players in the mobile industry. The declared mission of the LiMo Foundation is "to create an open, Linux-based software platform for use by the whole global industry to produce mobile devices through a balanced and transparent contribution process enabling a rich ecosystem of differentiated products, applications, and services from device manufacturers, operators, ISVs and integrators" [LiMo Foundation, 2007].

In November 2007, companies including Google, HTC, Samsung, LG, and Motorola have formed the Open Handset Alliance (OHA), which aims at providing technologies that will significantly lower the cost of developing and distributing mobile devices and services. Google has developed Android, "the first complete, open, and free mobile platform" [Open Handset Alliance, 2008]. The Android platform is the first step in this direction – a fully integrated mobile software stack that consists of an operating system, middleware, graphical interface and applications. Consumers are expecting the first phones based on Android to be available in October 2008 [Open Handset Alliance, 2008].

In early 2008, AOL announced the Open Mobile Platform — a software development platform for multiple operating systems that aims at making it easier for developers to deploy applications across the countless mobile operating systems and platform options [AOL Developer Network, 2008].

In a nutshell, 3G networks and open systems give mobile operators and device manufacturers significant freedom and flexibility to design products. Furthermore, it should significantly lower the entry barrier for new developers, fostering a community-based ecosystem for innovative applications.

2.6 Mobile User-Generated Content

As previously outlined, location-based services depend on content that is provided by various sources like mapping agencies and yellow page services. However, mobile user-generated content and social networking services for mobile phones are quickly becoming the most important long-term business model: in October 2006 the International Herald Tribune [Doreen Carvajal, 2006] featured a story in which it examined the rapidly growing value of mobile phone based digital community services — user-generated pictures and

videos on such sites as See Me TV, virtuality sites like Cyworld, etc. — and estimated it to be $3.45 billion. 2004, the total value of mobile digital community services was well below $200 million. With the growing popularity of sophisticated telephones, Informa Telecoms & Media forecasts that globally, operator revenue from such services will rise to more than $13 billion by 2011 [Informa Telecoms & Media, 2006].

There are several location-based services outside the multimedia entertainment segment which benefit from user-generated content and other Web 2.0 principles outlined in section 2.1. For example, traffic jams and radars can be detected based on information shared voluntarily between all users of the system [Michel Deriaz , 2008]. Nokia teamed up with TomTom to develop a system for automatically detecting traffic jams by analyzing motion patterns of mobile phones [Hilmar Schmundt , 2007]. The concept is simple: if a large number of mobile phones are moving along a highway at a snail's pace, there must be a traffic jam.

The task of content provision has traditionally been handled by commercial institutions, travel agencies and professional critics. With the ever increasing popularity and recently observable convergence of mobile devices and the Internet, a new magnitude of user-generated content creates new possibilities for content provision. User-generated content can substitute, extend and enrich commercially available content. For example, by collaborating, users themselves can maintain an up-to-date database of interesting jogging and biking routes in a large metropolitan area, being independent of commercial content providers.

Motivations for Contributions

Despite the appeal of online communities, large numbers of them fail. For example, in many online groups, participation drops to zero. On the popular peer-to-peer file sharing service, Gnutella, ten percent of users provide 87% of users with all the content. In open-source development communities, four percent of members contribute 50% of answers on a user-to-user help site, and four percent of developers account for 88% of new code and 66% of code fixes [Beenen et al., 2004]. Although not everyone needs to contribute for a group to be successful, groups with a large proportion of non-contributors have difficulty providing needed services to members. To design technical features of online communities and seed their social practices in a way that generates ongoing contributions from a larger fraction of the participants is considered an important and difficult challenge [Beenen et al., 2004].

In order to design an application that relies on user contributions, the various reasons why people contribute to such platforms should be understood

first. Kindberg et al. [Kindberg et al., 2005] have classified the reasons for camera phone image capture into two dimensions: social vs. personal, and affective vs. functional. In a nutshell, social/affective photos were taken for sharing experience or to connect with people, and social/functional photos are intended to support a task. Individual/affective photos are images intended for personal reflection or reminiscing, whereas individual/functional photos support some future tasks not involving sharing. A related study was performed by Ames and Naaman [Ames and Naaman, 2007], who examined the tagging activity in two active photo sharing web sites, Flickr and ZoneTag. Similar to Kingberg et al.'s work, Ames and Naaman describe a taxonomy of motivations to contribute annotations in these systems along two dimensions, sociality and function. The functional dimension is defined by the need for "organization" and "communication", whereas the sociality dimension differentiates between personal and social use of annotations.

To conclude, while user contributions typically seem to have a personal and social side, the nature of the functional dimension depends on the nature of the shared information and its possible uses.

Automatic Content Creation

Mobility brings another new possibility – the automation of content creation and consumption. While on the move, the user's context is changing, making things and events around the user become interesting. Mobility adds triggers which can be automatically detected and utilized without disturbing the user. Typically while on the move, the user has to direct attention to something else than the mobile device. At the same time, however, there can be interesting things and events around the user, the sharing of which could be relevant to companies and other users (e.g. the sheer fact that the user has been somewhere). The aforementioned traffic jam detection system from TomTom and Nokia is an example for this approach.

In [Toivonen, 2007], two main reasons have been identified why automatic content creation might be useful. First, a previously unimaginable amount of data can be collected. Data can be gathered from sensors and devices and sent either as raw or aggregated data to a Web server. Context data like the user's location, weather conditions, services nearby, and the state of the device are typical examples of data that can be continuously or periodically collected without the need for a direct conscious user interaction each time.

Second, it is not feasible to expect people to manually tell the system that they are in a certain context. However, not all information can be automatically detected with good accuracy and reliability, and for those also manual input is often needed. It depends on the nature of the shared con-

tent/context, whether doing it manually or automatically is more useful. For example, the purpose of use is one of Nivala's context categories (see Table 2.1) that is probably better collected manually. By merely observing a user walking into a building with lots of shops and restaurants, it is difficult to deduce the performed task or activity with reasonable certainty. Nevertheless, activity detection is an active research topic with promising results [Bellotti et al., 2008].

Automatic content sharing entails both positive and negative features. By analyzing large amounts of automatically collected user data, comprehensive user preferences can be extracted, for example. The biggest issue with this, however, is privacy [Toivonen, 2007]. In [Iqbal and Lim, 2007], Iqbal and Lim have conducted comprehensive research to assess issues and threats in a real-life scenario involving location data obtained from people. By employing anonymization techniques, opt-in/opt-out mechanisms, a priori mechanisms like setting control details, and a posteriori mechanisms for altering or deleting shared content, the privacy issue can be handled to a satisfying extent [Iqbal and Lim, 2007].

2.7 Capturing and Using Location Information

To give a general sense of key terms used in this thesis and aid understanding of the domain, informal definitions of the terms place and route are given below.

A **place** is a real-world location where people can engage in daily activities like eating, relaxing, exercising, socializing, etc. Examples for places include parks, museums, cinemas, health clubs, bars, and restaurants.

A **route** is regarded as a well-established course of travel, a way which is passed or is to be passed. Examples for routes include routes for jogging, biking, hiking, or sightseeing. It is important to note that the route itself is the object of interest here, not its start or end points.

Mobile Recommendation Systems

As the modern world spins faster and faster, people are faced with the challenge of organizing their accelerated daily lives and their mobile activities. Supporting tools and technology carry the potential to assist the user in the organization of their daily activities which get more and more complicated and time consuming [Reichenbacher, 2004]. The number of potential places to visit and routes to follow can be quite overwhelming, especially in

dense touristic regions. Personalized mobile recommendation systems can help users find places matching their interests and current situation.

Tourists or new residents commonly need local place and route information: a businessman would like to know where to go for a run in the vicinity of his hotel, a new resident wants to know where the nearest upper-class gym is located, a group of exchange students would like to know where the nearest student pubs and bars are, a group of tourists wants to go on sightseeing tours, etc. Even longtime residents need local information when their habits, interests or obligations in life change. For example, new parents suddenly need to discover places where they can find children's activities or other parents who want to socialize. When a person moves from one part of a big city to another, they must re-establish a bond to the places nearby and the new surroundings.

In order to effectively assist the user, a significant amount of adaptation is indispensable when providing such information. In [Reichenbacher, 2004], Reichenbacher has identified three reasons for this:

- First, the increasing quantity of user-generated spatial content and the danger of overstimulation are pushing for a suitable channeling of the information stream.
- Second, adaptation could lead to greater acceptance of new, yet partially still immature technologies.
- Third, new value-added services which users have to pay for need customization to guarantee user satisfaction.

However, a major problem in designing adaptive recommendation systems is finding an effective, reliable method for acquiring user preferences [Ricci and Nguyen, 2007].

Acquiring User Preferences about Places and Routes

The National Human Activity Pattern Survey [Klepis et al., 2001] in the United States showed that people frequent everyday places including malls, stores, health clubs, and parks for over two and a half hours a day. Doing so, they develop personal preferences about nearby places and routes, and a personal sense about the more or less subtle distinctions between them. The system design must find the right balance between having precise information about user preferences and the cost of acquiring it. For example, explicitly querying the user focuses on the former, while user navigation mining focuses on the latter.

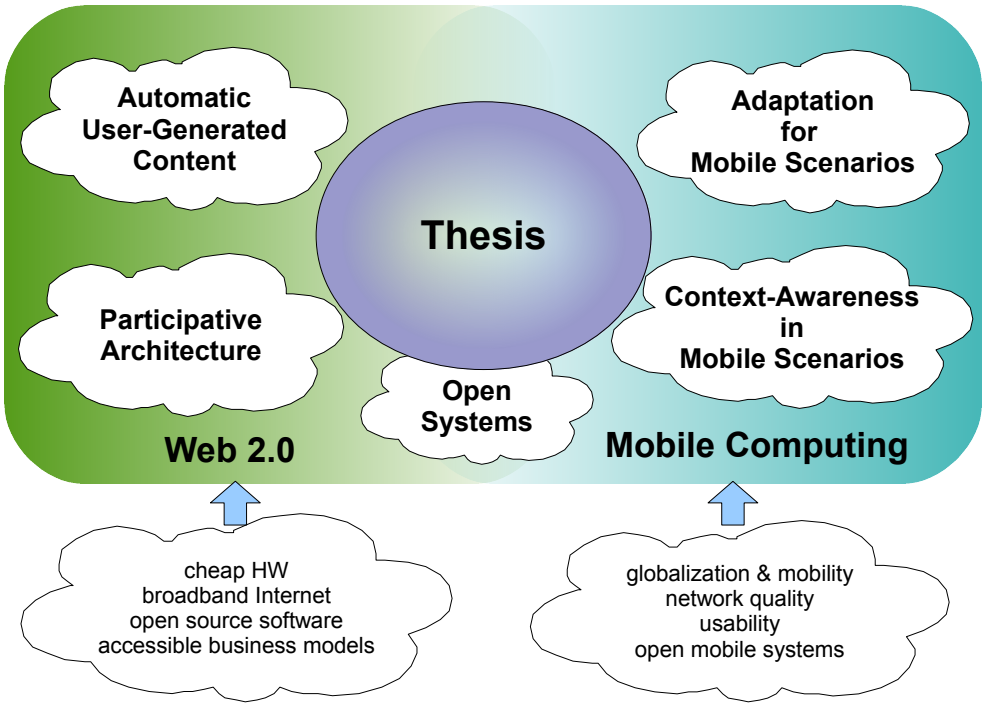


Figure 2.5: Thesis scope

When talking about acquiring user preferences about places or routes, a central question arises:

Will people share location information?

Related studies about the motivations for contributions in online communities have already been introduced in section 2.6. More specifically, Ludford et al. investigated people's willingness to share their *location information* and messages associated with that location [Ludford et al., 2007]. For this purpose, they utilized a system which combined a personal notification tool with a community bookmark/recommender system. From this study the authors were able to create heuristics about how people decided what kinds of location information to share. In summary, most people were willing to share information about public places, any messages about typical activities occurring at those places, and places they wish to recommend. Places that they do *not* want to share are locations that include names (especially those of children), residences, or private workplaces.

Ludford et al. also investigated the *motivations* for sharing place information. The results are comparable to Kindberg et al.'s two-dimensional taxonomy. Basically, the subjects mentioned four categories of uses: per-

sonal organization, personal diary, collaborative place discovery and social matching. For example, people can search for places in their vicinity when in need of, and review/recall the places they have visited in the past months. One person who had lived in the neighborhood for over 40 years noticed a place two other people had bookmarked. Since he did not recognize the place, he wanted to find out what it was. Another person said that if he saw that a restaurant was popular with others, he would create a message reminding himself to go there. Additionally, subjects wanted to meet others who have bookmarks for the same lesser-known places. They took this as an indicator of shared interests, and they wanted to use this information for social matching [Ludford et al., 2007].

2.8 Summary

A large quantity of spatial data is produced by people engaging in various everyday activities, on the one hand. People on the move have a growing need for spatial assistance, on the other hand. In summary, there is a well founded use case for capturing user-generated content and personal preferences about places and routes and making it available to everyone who needs it in an adequate way. Figure 2.5 narrows down the elaborated scope of this thesis, indicating a convergence of the described trends and technologies.

Chapter 3

Related Approaches and Systems

The previous chapter raised two fundamental questions:

- How to capture long-term user preferences about places and routes?
- How to adapt shared spatial content to users on the move?

This chapter analyzes related research work and available commercial systems with respect to these two questions. Special attention is put on the suitability for mobile scenarios, context-awareness, and personalization techniques employed in the examined systems. The chapter closes with an evaluation of related work, revealing deficits and potential for improvements.

Existing approaches and systems typically deal with either places or routes, but not both. Therefore, the chapter structure follows this distinction and splits the discussion in two parts.

3.1 Places

On the one hand, currently there exists a vast number of commercial location-based services supporting people on the move by means of providing on-site information about places, often called points of interest (POI). On the other hand, there is a series of academic research work on mobile context-aware recommendation systems focusing on places. This section describes a selection of existing commercial applications and academic work in the field of context-aware mobile place guides.

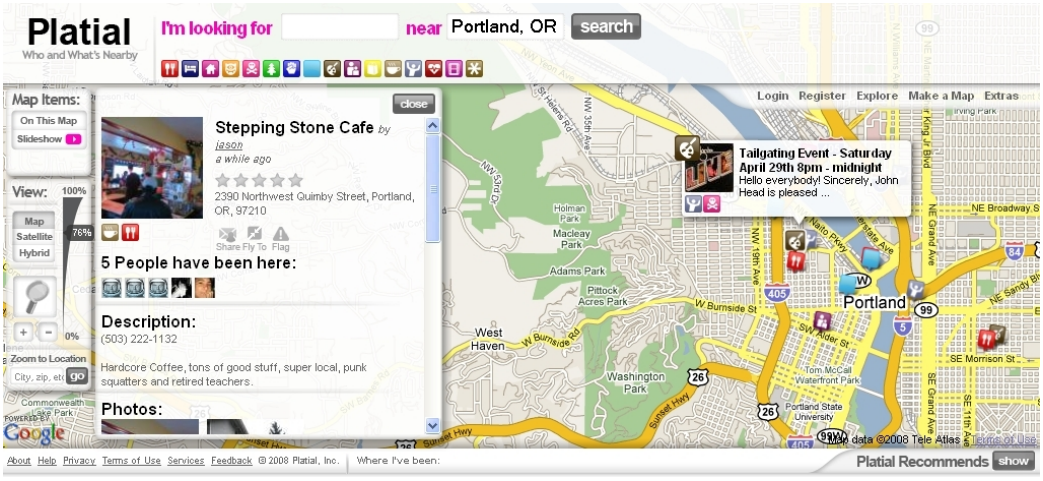


Figure 3.1: Platial - "Who and What's Nearby" (platial.com)

3.1.1 Platial

As mentioned in Chapter 2, user-generated content about places and routes (personal opinions, reviews, statements of visit, etc.) is growing rapidly. Individuals have been able to share place information online via explicit authoring since the early days of Web 2.0. For instance, PlaceOpedia[1] users connect Wikipedia articles with locations on a map. Sites such as TripAdvisor[2] and Yahoo Maps[3] allow people to write place reviews. Such sites are starting to offer clients for mobile phones nowadays. This section describes a selection of online services dealing with place recommendation.

Recently, several platforms like Platial[4], Google Earth[5], and other sites let users associate tags, photos and videos with places, and retrieve information about them on mobile phones. Platial also lets users see detailed information about a place: the categories it belongs to (Food, Lodging/Hotel, Parks and Nature, Art and Culture, etc.), the people that have been there, a textual description, user comments, and attached photos (cf. Figure 3.1). Basically, the content is contributed by users via the website. A mobile, location-aware client is available for the Apple iPhone as well.

According to Platial's homepage, users can add their own places, stories and photos to the map and search or browse "millions" of places by selecting a desired place category, a user, or a tag. A simple keyword search is

[1] http://placeopedia.com
[2] http://www.tripadvisor.com
[3] http://maps.yahoo.com
[4] http://www.platial.com
[5] http://earth.google.com

available as well.

Flagr[6], a service similar to Platial, allows users to submit place reviews via text messages from their phones (SMS). The message needs to obey a certain structure and the service works for U.S. locations only.

3.1.2 Qype

Qype[7] is an open platform for sharing local knowledge and advices about places. Qype users can write place reviews, upload photographs of the places they have visited, join groups to discuss their particular interests, and create personal lists of their favorite places. Qype has more than two million unique visitors per month, according to its homepage. There is no context-aware mobile client for Qype – the website is the only channel for interaction. Users can search for places by entering keywords, browsing through a predefined taxonomy of place categories, or reading reviews from other users.

3.1.3 Whrrl

Whrrl[8] is a mobile network that aggregates information as users visit different places. It is a place review service that is wrapped up in a map mash-up and a social network. Unlike Platial or Qype, Whrrl has been built from the ground up as an integrated mobile and Web experience. To tell the system about a place visit, the user has to explicitly check in via a mobile application, a SMS, or the website. Explicit ratings, descriptions and reviews can be specified along with user's presence. Adding new places and moving existing ones on the map is supported via the web interface, as well. To search for places, mobile users may specify what they are looking for by entering keywords or names, e.g. *pizza* or *music*. Whrrl uses the distance from the user's current location, explicit place ratings by people in the user's social network and "other elements" to provide a ranked list of places and events that are relevant to the user.

3.1.4 CitySense

The systems described so far focus on manual acquisition of short-time user preferences, completely ignoring the opportunity to utilize long-term preferences. CitySense[9] chooses a different approach. It is a mobile application for

[6] `http://www.flagr.com`
[7] `http://www.qype.co.uk`
[8] `http://www.whrrl.com`
[9] `http://www.sensenetworks.com`

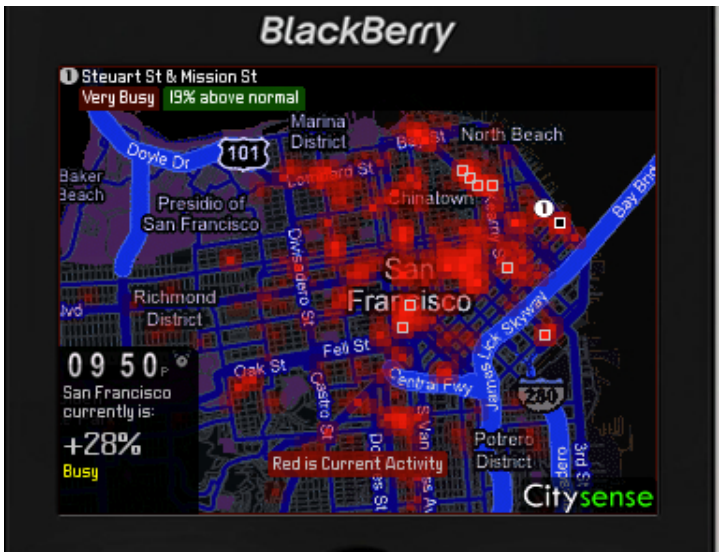


Figure 3.2: CitySense on a BlackBerry [Sense Networks , 2008]

local nightlife discovery and social navigation. CitySense shows the overall activity level of the city, top activity hotspots, and places with unexpectedly high activity in real time (Figure 3.2). However, it does not store information about places – it links to Google to show what venues are operating at hotspot locations.

CitySense learns about where each user likes to spend time and it processes the movements of other users with similar patterns. Professor Tony Jebara, Chief Scientist and Director of the Columbia University Machine Learning Laboratory, has determined seven different groups (or tribes) of people related to nightlife behavior, whereby each group has its own type of destination. Users who go to rock clubs see rock club hotspots, users who frequent hip-hop clubs see hip-hop hotspots, and those who go to both see both. According to Professor Jebara, those destination types translate across multiple cities – it becomes a lot easier to figure out where to go out in a new city [Sense Networks , 2008]. Currently, however, CitySense only answers the question "Where is everbody?". In the future, the system is supposed to know "Where is everybody like me?" by utilizing the aforementioned groups concept. An excellent video giving an overview on how the CitySense algorithm works can be found at [Sense Networks , 2008].

CitySense uses origin and destination data from taxis to model the city. The original plan was to use data from mobile phone usage or other sensors. However, the SenseNetworks founders were only able to get taxi data and found that it could be used to get a very clear picture of how a city ebbed

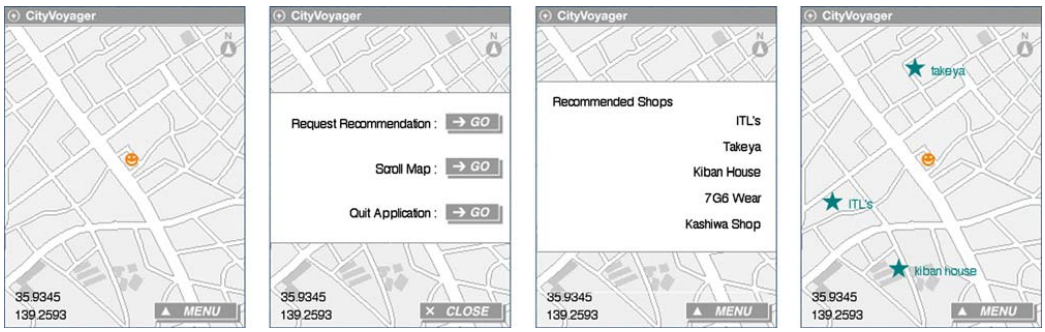


Figure 3.3: CityVoyager screenshots [Yuichiro and Masanori, 2006]

and flowed [Brady Forrest , 2008]. SenseNetworks is paying special attention to personal data management and privacy. There are buttons in CitySense to delete any data acquired in the last 24 hours and to delete all historical data. After these deletions are made, personalized services will no longer operate, but users always have this choice.

3.1.5 CityVoyager

CityVoyager is an academic-class city guide system for GPS-equipped mobile devices which finds and recommends shops that match each user's preferences. The basic idea behind CityVoyager is to apply the techniques used in sophisticated recommendation systems, found in online shopping sites such as Amazon.com, to real-world shopping [Yuichiro and Masanori, 2006].

Long-term user preferences are estimated from the user's past location history. The analysis of history location data includes a place learning algorithm that can detect users' frequented shops by their proper names. Whereas online recommendation systems estimate long-term preferences from the users' online activity records, such as items bought or checked in the past, CityVoyager estimates preferences based on their location history during shopping in the city. It analyzes raw location data acquired using GPS and transforms it into a list of each user's frequently visited shops, which is stored inside the server and used when making recommendations[10]. Rating values, which indicate how fond the user is of each visited shop, are also calculated automatically based on visit frequencies.

The recommendation process consists of two sub-phases: filtering and adding weights according to certain geographic areas. Filtering of shops is done using item-based collaborative filtering, a proven algorithm used in

[10]Section 6.1 analyzes existing algorithms for detecting place visits.

Figure 3.4: Magitti's Main Screen (left) and Detail Screen (right) [Bellotti et al., 2008]

many existing recommendation systems[11]. The shops selected by the filtering algorithm are weighted according to the areas in which they are located, with large weight values being added to shops in the same area as the user, or in areas with large transition probabilities. Areas are defined in CityVoyager so that any two points located in the same area are easily accessible from one to the other, e.g. by roads. Transition probabilities are calculated from periodically plotted user locations, and a higher probability indicates more chances of a user advancing to the area.

Shops picked out through the previous steps are shown using simple icons on a map, on the screen of the client device (cf. Figure 3.3). A "metal detector" interface uses auditory cues to convey information in an unobtrusive way. Repetitive beeping sounds are emitted when the user is within a certain distance from a recommended shop, and the intervals between the beeps become shorter as the user moves closer to the shop.

In summary, CityVoyager utilizes the current location and the movement history as context types. A method for further narrowing down shops based on prediction of user movement and geographical conditions of the city is proposed, as well.

3.1.6 Magitti

Another academic-class system that uses several context types to avoid the overload of leisure time offerings in dense urban areas is the academic prototype Magitti. Similar to CityVoyager, it can do so without the user having

[11]Section 6.6 deals with recommendation algorithms in more detail.

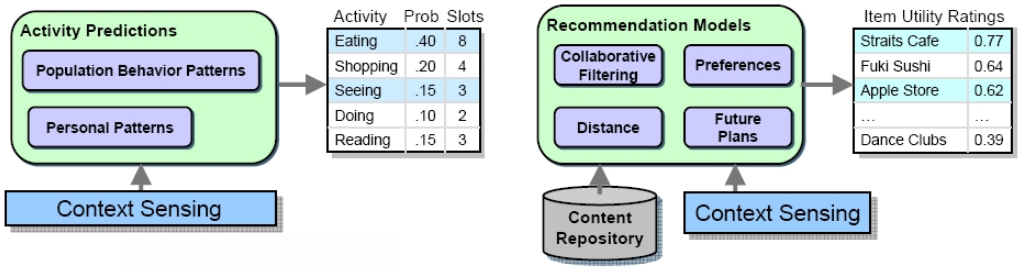


Figure 3.5: Magitti – activity prediction process (left) and recommendation system (right) [Bellotti et al., 2008]

to explicitly define their profile or long-term preferences.

Magitti's Main Screen has a scrollable list of up to 20 recommended items (places) that match the user's current situation and profile (cf. Figure 3.4). By tapping an item on the screen, a detailed description, a formal review, and user comments are displayed. As the user walks around, the list updates automatically to show relevant nearby places.

The system infers interests and activities from models that are learned over time implicitly, based on individual and aggregate user behavior, such as places visited, web browsing, and communications with friends [Bellotti et al., 2008]. Magitti knows about current time, location, weather, store hours, and user patterns. Five activities were derived from observations in the authors' field work: Eating, Shopping, Seeing, Doing, or Reading.

Magitti is unique in that it infers user activity from context and patterns of user behavior and, without its user having to issue a query, automatically generates place recommendations. Activity probability prediction is based on a combination of patterns observed across the user's demographic population and individual behavior pattern. The population patterns were derived from the data collected on Magitti's target demographic in fieldwork and from a Japanese Survey on Time Use and Leisure Activities [Japanese Statistics Bureau, 2006]. Individual user behavior models are learned over time by associating each venue in Magitti's database with one of the five activity modes that were observed in fieldwork, and modeling the frequency of each mode by tracking user behavior. For example, if a user visits a retail store, the system records that as being in the Buy activity mode; similarly for visiting a restaurant or café, (Eat), theater or museum (See), gym or park (Do), and reading of content on Magitti itself (Read). These recorded location visits create models of the user's individual activity preferences.

For a given user and context, the recommender computes the utility of

each content item by combining results from a variety of recommendation models (cf. Figure 3.5). For example, Magitti includes a Content Analysis module that performs plain text analysis of the content of calendar appointments, viewed documents, and messages to extract information about the user's plans. The extracted information is used to infer the probability of the user's interest in activities at current or future times. The scores of items that have previously been seen are reduced, providing diversity to the set of recommended items. When all items have been scored, the top four results are displayed as a list on the Main Screen, ordered by their total score.

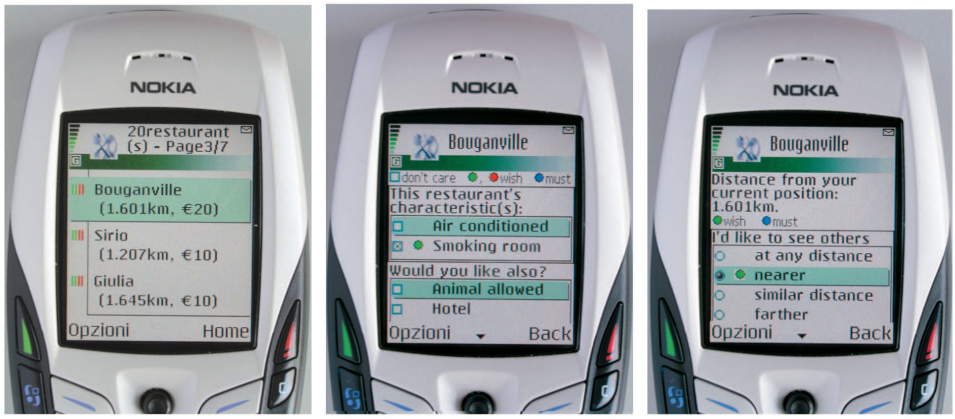
3.1.7 MobyRek

MobyRek is another academic-class context-aware tourist recommendation system [Ricci and Nguyen, 2007]. It lets users specify preferences for places such as hotels and restaurants, and it improves its recommendations over time. The system derives the user's long-term preferences, such as a preference for non-smoking rooms, from several recent interaction sessions between the user and the system and by letting users explicitly define a set of stable preferences (for example, the payment method). These stable preferences remain true throughout the sessions. In contrast, session-specific preferences, such as a desire to eat pizza, are transient. Session-specific preferences include both contextual preferences (i.e. space-time constraints) and product feature preferences (e.g. pizza restaurant, payment with credit cards).

MobyRek elicits user preferences through structured human-computer dialogs. A user's goal when participating in such a dialog is to find desired products, and the system's role is to help the user quickly and effectively find them. During a dialog, the system might either ask the user for a preference or propose a product. The user either answers the system's question or criticizes its proposal.

In particular, the system can raise a selective question to refine user preferences when the number of candidate products is unsatisfactory – for example, when the system found no items or too many items. When replying, the user can indicate, remove, or modify some conditions so that the system can retrieve a better result set (Figure 3.6). Over time, the system builds a user's long-term preference pattern.

When a recommendation session ends, whether successfully or not, the system retains it as a case. That way the system can exploit past user recommendation sessions in making new recommendations for that user and similar future use cases. For instance, suppose user A selected a particular hotel before travel and that user B previously selected the same hotel and a restaurant. The system would consider that restaurant to be a good choice



(a) Recommended restaurants

(b) Users can browse them...

(c) ... and critique them.

Figure 3.6: The MobyRek user interface [Ricci and Nguyen, 2007]

for user A. The session-specific preferences have the features of a restaurant selected in another travel by a user who made similar selections, in a similar contextual situation, with similar default preferences[12].

3.1.8 Crumpet

Crumpet is a EU project aiming at the "creation of user-friendly mobile services personalized for tourism" [Schmidt-belz et al., 2002]. The user can request information and recommendations about tourist attractions, restaurants and tours. The system provides pro-active tips when the user gets near a sight that might be of interest, supports interactive maps showing the position of the user as well as interesting sights. Its architecture foresees an external customization approach allowing the integration of various service and content providers via a dedicated interface.

Crumpet considers the following context types: location, device, network, and user preferences. Transformation of GPS coordinates to logical locations encompasses, in a first step, sending only significant location changes to the server and based on that, utilizing separate geo-coding services to infer addresses from the given coordinates in a second step. The logical device context is taken into account in terms of the device type determining, e.g., size of display, color depth, and raster vs. vector graphics capability. The logical network context is considered with respect to the quality of networking service (QoS) and the type of wireless connection that is available (WLAN

[12]This approach belongs to the category of Case-Based Reasoning.

or GSM). The device context and the network context together define how much and what kind of information should be sent to the user. For example, if the network connection is poor, it is not practical to send large quantities of data, such as movie clips. If the user is using a mobile phone with text-only capabilities, he cannot receive images even if the network connection is good. The user's interests representing the logical user context are learned dynamically by tracking user interaction. In Crumpet an adaptive user model learns user interests from the user's interaction with the system. When a user asks for more information about an object, this adds a small amount to the evidence that a user is interested in objects with these features, more than in others. If a user asks for more and more details about the same object, or even asks for directions to a site/restaurant, this adds a greater amount to the evidence that she or he is interested in such services.

Adaptations are performed by locating and querying suitable content and service providers and adapting the query outcome to the aforementioned context types. According to the user's position and interests a list of places of interest is generated and displayed.

3.1.9 Evaluation

In this section some major issues with the described approaches are pointed out, indicating possible areas for improvement. As a general observation, there are fundamental differences between the commercially available systems and the academic-class prototypes in this field. Accordingly, the evaluation is split in two parts.

Commercial platforms

Commercial platforms like Platial and Whrrl are gaining a lot of momentum, shaping a new class of services – *mobile social applications*. Their primary aim is the linking between users by imposing a virtual digital media and metadata layer on top of spatial data. They do not utilize automated personalization techniques – search is usually performed by manually selecting place categories, tags, and by entering keywords. The user has to manually go through written place reviews and decide which users to trust. This often imposes significant overhead and does not meet the need for quick decision support in mobile scenarios.

A further ramification of relying on explicit user comments and reviews is the introduced bias of information. Not everybody is willing or able to take the time and write some meaningful text about a recently visited place.

Moreover, users' opinions about a place might change over time. These aspects result in a biased set of stated opinions available on such platforms.

Finally, current location is typically the only context type used in commercially available systems, whereas long-term user preferences regarding places are not considered.

CitySense differs from those systems as it acquires and revises long-term user preferences based on movement history. These user profiles are used to present personalized views on popular spots in the city. CitySense's other distinguishing aspect is its focus on privacy issues. However, CitySense has no knowledge about real places such as clubs or restaurants – the user has to perform extra steps to get the desired information, i.e. performing an Internet search for places. The system has been designed for maximum usability by fully automating the data collection and aggregation process, abandoning the idea of an accurate identification of actual places the users have visited. In city areas with high place density this method is susceptible to errors and the resulting fuzziness might lead to distorted user profiles. CitySense is still a beta product. Currently, it is available only on the BlackBerry and only contains data for San Francisco, CA.

Academic Prototypes

CityVoyager is somewhat similar to CitySense in the way that it tries to create user profiles from movement history. However, the approach it takes differs as it considers established place locations and tries to recognize actual place visits. In [Yuichiro and Masanori, 2006], the place detection algorithm showed rather disappointing results – it gave only 31% true positives. Moreover, the place detection algorithm is tailored to shops only, considering only places that are represented as single points on the map. Places which exist continuosly in the two-dimensional space such as parks can not be considered by CityVoyager's algorithm. Nevertheless, this makes the algorithm very efficient for this particular class of places. However, authors also report that the estimation algorithm will inevitably have trouble when a user frequently visits two proximate shops.

Magitti employs a unique way of pro-active place recommendations based on activity prediction algorithms. The system infers interests and activities from models that are created and maintained without any user interaction. This includes text analysis of messages sent and received, websites and documents opened, and places visited in the past. However, nothing is said about the method for detecting place visits. Another issue is the requirement that all places must be classified manually into exactly one of the five activity categories. This makes the places database difficult to

maintain. The plurality of employed recommendation algorithms increases the model complexity, results in low transparency of recommendation results and lowers the trustworthiness. The lack of explanations or special visualization techniques for the four recommended items further rises questions regarding user acceptance.

MobyRek elicits user preferences through structured human-computer dialogs. This critique-based approach might seem appropriate for use cases such as trip and hotel planning. However, the time required to interact with MobyRek might be too long for quick everyday decisions regarding activities such as spontaneous going out, having a drink, relaxing, etc. Moreover, users are known for not giving very much feedback on the appropriateness of presented items, particularly not negative feedback (see the discussion in [Schwab and Kobsa, 2002]).

Crumpet allows the users to inspect their preferences model and modify it. The preferences are learned by analyzing user interaction with the system. Unlike CityVoyager, Crumpet decided to drop the idea to use user's movements to infer interests as the localization was found to be not precise enough to automatically determine the user's logical position (place) without doubt.

Furthermore, all described research-class systems rely on a pre-filled places database, whereas commercial systems like Qype entirely rely on users to maintain the places database. Academic city guides employ no mechanisms for considering user contributions regarding place locations in a collaborative way, making the systems highly dependent on 3rd party (commercial) content providers.

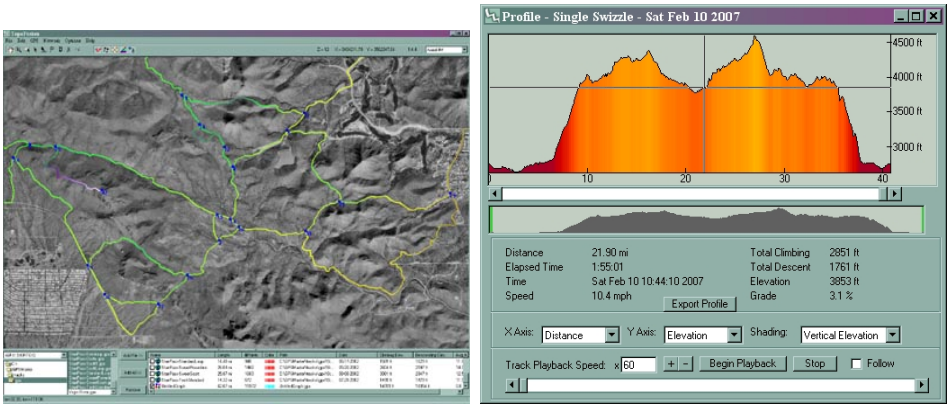
User preferences are either collected automatically by observing user interactions (Crumpet) and inferring place visits by employing some spatial place detection algorithms (CityVoyager), or manually through a sequence of questions (MobyRek). None of the approaches alone provided satisfactory results, leaving a semi-automatic approach for acquisition of user preferences an unexplored option.

Adaptations on the information level are offered by filtering and sorting items according to the user's preferences and different context types. On the presentation level, the user's current position is highlighted on the map and recommended objects of interest are displayed without any explanations or visual distinctions between them regarding their inferred relevance. CityVoyager's unique beeping approach yielded rather unsatisfactory results as it was found to be sometimes annoying and embarrassing in public [Yuichiro and Masanori, 2006].

3.2 Routes

With the elevated price of spatial data sets and the unavailability of data meeting the needs of specific user groups it is not surprising that the idea of collecting spatial data by voluntary contributors emerged. Mountain bikers were among the first communities to feel unhappy with the commercially available spatial data products: their cartography focuses on roads and much of the thematic information is useful especially to car drivers. As a consequence, several biker communities started to collect GPS track data and published their data through web portals such as the GPS-tour.info portal[13]. Bikers download tours that others have uploaded and use them for planning and navigating their own tour.

GeoSkating[14] is a similar project, aiming at the skating community. While skating, GPS position data is published directly to a server through a mobile phone. At the same time the skater can enrich GPS data with road surface ratings and by sending pictures and videos from the phone. The server will draw geographic maps showing road quality through coloring (red, yellow, green). Pictures and videos are presented on the GPS locations where they were captured. Skaters can also be followed in real-time over the map while skating.



(a) TopoFusion Main Screen (b) Track elevation profile

Figure 3.7: TopoFusion screenshots

[13] http://www.gps-tour.info
[14] http://geoskating.com/

(a) Several GPS tracks... (b) ... reduced to a single route segment!

Figure 3.8: TopoFusion track reduction

3.2.1 TopoFusion

In contrast to the aforementioned websites, collaborative acquisition and aggregation of spatial data constitutes a central concern for the digital trail library described by [Morris et al., 2004] and commercially available as TopoFusion[15]. TopoFusion is a fully featured GPS mapping program for the PC used to display, scroll and zoom over maps, while overlaying GPS tracks (routes) and supporting information on top of them.

The basic representation of a route is a GPS track log, recorded as recreators travel on routes. As users complete trips, the GPS track logs of their trips are voluntarily submitted to the central library. A major problem with real-world GPS data is that GPS track logs will overlap and intersect each other. TopoFusion's distinguishing feature is the ability to create aggregated route networks from a collection of overlapping and intersecting GPS track logs. Aggregation is dealt with at the level of geometrical information using algorithms that compute routes by averaging over many GPS tracks. TopoFusion identifies sections of tracks that are close and similar enough such that they are actually representations of the same route or road. The algorithm averages these segments to eliminate duplicates [Morris et al., 2004]. Figure 3.8 shows several GPS tracks (a) being combined to a single route segment (b). Since this thesis builds on TopoFusion's route aggregation algorithm, a comprehensive description is given in section 6.5.

[15] `http://www.topofusion.com`

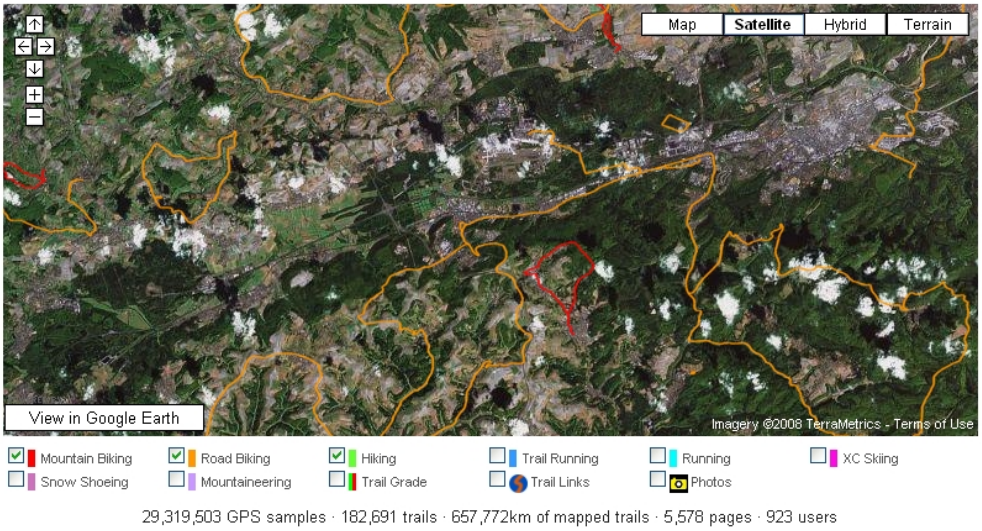


Figure 3.9: Trailguru screenshot

3.2.2 Trailguru

Similar to TopoFusion, the Trailguru project[16] aims to combine GPS track data from multiple trips to produce a comprehensive route database. However, the website reveals no information about the used aggregation algorithms. The results seem to be comparable to TopoFusion's, though. Trailguru has no PC or mobile client – user interaction is handled via the website only (cf. Figure 3.9).

Tracks are categorized to several predefined activity categories, such as Mountain Biking, Hiking, Running, and Skiing. Users can upload, share and explore routes and attached photos in Google Maps and Google Earth, engage in forum discussions and subscribe to feeds. Registered users can also create new routes by manually selecting route segments. Trailguru makes several statistics available to users, including total kilometers in the last weeks/months.

3.2.3 Evaluation

Services like GPS-tour.info and GeoSkating are basically used to generate multimedially annotated maps. They do not aggregate the collected spatial data.

An advantage of combining multiple submissions for identical route segments as employed in TopoFusion and Trailguru lies in the increased quality

[16] `http://www.trailguru.com`

of the data. GPS errors can be corrected by averaging the segments together. The result will be a better representation of routes. Since it is impossible to verify whether routes submitted actually exist, some sort of reliability measure using the number of submitted tracks that cover a segment has been suggested in [Morris et al., 2004] as well.

Even though systems like TopoFusion aggregate content and help avoiding information overload, they do not offer any personalized adaptations of shared content. This keeps the burden of analyzing the vast amount of uploaded content and selecting relevant routes on the user's side. Using long-term user preferences or other methods for personalization purposes is not explored by any of the mentioned route guides. Besides the user ID, any relation to the person who uploaded the route and their preferences is basically lost and must be manually explored, similar to commercial place guides like Platial.

There are no dedicated mobile client applications for TopoFusion or Trailguru – routes must be uploaded manually from GPS-enabled devices to the website. Browsing routes is also possible via the website only.

3.3 Overall Analysis

This chapter introduced and evaluated several applications and research prototypes dealing with places and routes. In a nutshell, the related approaches can be divided into three large groups:

- commercial systems dealing with places, with the primary aim at linking users,
- systems relying on 3rd party databases of pre-classified places dealing with personalized place recommendations, and
- systems for uploading and browsing routes, without significant personalization techniques.

Commercial systems dealing with places let users explicitly specify ratings or written reviews about the places they have visited. While such systems are effectively addressing the social and affectional aspects of sharing multimedia content and annotations about places (cf. also the discussion about motivations in section 2.6), their suitability for an unobtrusive and comprehensive acquisition of user preferences and collaborative place discovery seems to be limited.

The focus of route-oriented applications lies on the data collection part. To some extent, data is also aggregated to help reduce the burden on users

when browsing the database. However, a digital route library should account for the differences in preferences and information needs between users. Mobile scenarios characterized by limited user attention time demand further optimizations of the information flow besides spatial aggregation techniques, as offered by personalized recommendations.

Academic place recommendation systems, on the other hand, focus on such personalization techniques to avoid information overload. Commonly, these systems deal with the acquisition of long-term user preferences regarding places by observing the user's behavior. However, the fully automatic approach delivers unsatisfactory results when used without any additional user interaction, whereas approaches based solely on structured dialogs and explicit feedback have arguable effectiveness and limited potential for large-scale success.

Current place-oriented systems model places as single points in space (i.e. as "points" of interest). However, observed spatial behavior could also be used to derive spatial characteristics of places as well. While route-oriented systems such as TopoFusion aggregate GPS tracks to accurately describe routes, location data gathered from multiple visits might be aggregated in order to describe geographic shapes of places. More accurate information about place shapes could further improve the process of acquisition of user preferences and the assistance of users on the move.

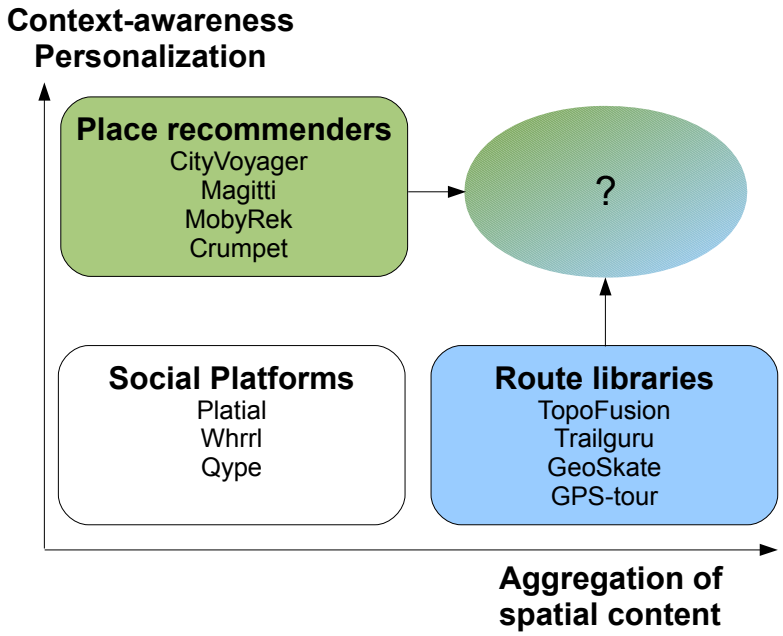


Figure 3.10: Positioning of related work categories

To the best of my knowledge there is no application trying to combine places and routes. For example, when planning a jogging or hiking route, users might be interested in the surrounding places, as well. Significant places such as castles, churches, or lakes might be exactly what a user is looking for when going for a run. Existing digital route libraries only deal with routes themselves, however.

Figure 3.10 summarizes the drawn conclusions about the analyzed state of the art, revealing a potential gap between the two related groups of approaches.

Chapter 4

Introducing Crumblr

This chapter describes the core concepts behind *Crumblr* – a novel mobile application dealing with places and routes, prototypically developed in the course of this thesis. First of all, a detailed introduction to the central goals and challenges addressed by Crumblr is given, describing the main contributions of the thesis from a conceptual point of view. This is followed by a comprehensive description of the developed prototype in the form of realistic user stories, relating them to the introduced concepts.

4.1 Core Concepts

This section provides a high-level overview of Crumblr's core concepts. The proposed concepts and ideas are motivated by the background developed in Chapter 2 and the analysis of related projects in Chapter 3. The overview of Crumblr's core goals and concepts prepares the more detailed discussion of use cases given in section 4.2.

4.1.1 Semi-Automatic Acquisition of Long-Term User Preferences

The acquisition and revision of stable, long-time user preferences regarding places and routes is not considered in today's commercially available mobile recommender systems. However, user profiles are indispensable when trying to implement personalization techniques. People develop personal preferences regarding places and routes while engaging in everyday activities. The proliferation of mobile Internet technology offers several ways to obtain detailed clues about the people's spatial preferences. As analyzed in the previous chapter, commercial systems such as Platial provide a platform for users

to share place reviews and to explicitly rate places. Undeniable benefit results from explicitly capturing place or route reviews *in situ* – very detailed, emotionally rich reviews can be captured this way. However, these reviews are likely to be not very representative as the number of required actions to post such reviews is very high and the time available in mobile scenarios is limited. This would very likely lead to only a small subset of users (presumably the "tech-savvy" ones) willing to perform the required steps to upload and share written reviews, leaving a user in need of place recommendations with a biased set of unstructured reviews to manually choose from.

Related academic work tries to capture stable personal preferences by either employing an implicit, fully automatic, observation-based approach or by explicitly engaging in direct dialogs with the user. Crumblr is positioned between these two extremes, trying to find a good balance between the unobtrusiveness of automated user observation and the accuracy of explicit user interaction. One of Crumblr's fundamental goals is to employ a novel semi-automatic approach to gather data about place visits and to derive long-term user preferences from this data. This approach should be considered complementary to the purely manual approaches employed by existing commercial systems.

The questions "how to observe the user" or "what aspects to observe" must be answered when trying to base a long-term user profile on user observations. Crumblr is based on the idea that a person's visits to places and routes in the physical world can be an implicit form of expressing preference. [Froehlich et al., 2006] investigated the relationship between explicit place ratings and implicit aspects of travel behavior such as visit frequency and duration of stay. The results showed that, first, sensor-triggered experience sampling is a useful methodology for collecting targeted information. Second, despite the complexities underlying travel routines and visit behavior, there exist positive correlations between place preference and features such as the visit frequency and duration of stay.

Crumblr employs a method somewhat comparable to CityVoyager's recognition of place visits and Magitti's activity estimation (cf. Chapter 3). Crumblr is designed to estimate long-term user preferences from user's past locations and activities. It builds upon previous work on extracting places from traces of user's movements gathered by GPS or other location sensing technologies. In its essence, this thesis proposes a novel place detection algorithm based on a combination of clustering methods and detection of GPS signal loss events. Crumblr periodically samples the user's position data via GPS. Crumblr's core concepts related to sampled GPS data are based on the following two observations:

- If several GPS position samples relate to the same logical location (e.g. a park), those samples are aggregated to one logical place visit – the aggregation is based on a clustering algorithm considering both spatial and temporal aspects. For example, when a person spends more than 15 minutes in one geographic region, this is recognized as a place visit.

- When a loss of the GPS signal has been observed (i.e. the GPS module is unable to obtain a position fix) and if the next observed GPS position is close enough to the last valid one, it is assumed that the person has entered and left a building. Crumblr will recognize this case as a visit to the building.

Specifically, the algorithms were designed to address the noisy and sparse nature of GPS data caused by inherent GPS errors and long data sampling intervals. The long GPS sampling intervals have been chosen in order to avoid quick battery depletion that can be observed on some of the newer mobile devices equipped with GPS. Based on the detection of place visits, comprehensive long-term preferences can be derived. A detailed analysis of Crumblr's algorithms dealing with extracting place visits from GPS data is given in Chapter 6.

To address the identified accuracy problems arising from a fully automated place recognition approach, Crumblr periodically presents the identified place visits on a map, giving the user a comprehensive overview of their past spatial activities. Additionally, Crumblr uses this "point of interaction" to let the user confirm the estimated activity that the user has performed at each place. Activity estimation is based on aggregated history data. After the inevitable initial cold start phase inherent in all systems relying on user-generated content, Crumblr should have collected enough data to recognize the performed activities with good accuracy.

Figure 4.1 summarizes the outlined approach employed by Crumblr for semi-automatic place and activity detection. After analyzing the collected trace of GPS data, Crumblr has recognized visits to the cafe Venezia and the Irish Pub. By examining history data, both places are mapped to the Bars & Pubs activity category. Next, these suggestions need to be verified by the user, increasing the data quality and reducing privacy issues as well. As a final step, data is uploaded to the server, where it is persisted and serves for future place and activity recognition. Ideally, the system has correctly recognized the past places and activities and the user only has to confirm the act of uploading the data to the Crumblr server. The quality of automatically recognized visits is improving over time, which is a consequence of the

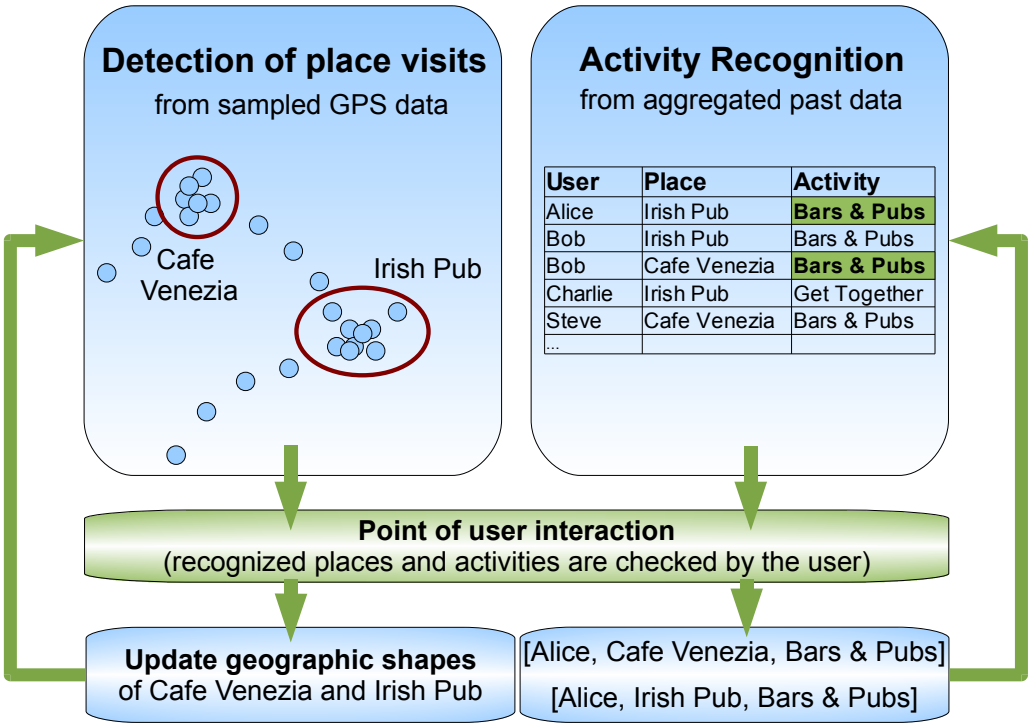


Figure 4.1: Place and activity detection process in Crumblr

utilized network effect. This aspect has already been outlined in Chapter 2 and is further described in the following subsection dealing with data aggregation. The feedback loops in Figure 4.1 express the improvement cycle in Crumblr.

Preferences regarding routes. In contrast to places, the acquisition of long-term preferences regarding *routes* is handled in a different way. Crumblr deals with recreational routes such as jogging, biking, and hiking routes. A constant, automatic observation similar to the recognition of place visits would lead to a large amount of useless data unsuitable to automatic filtering. Especially the start and end points of routes are better captured manually. Route visits are therefore not automatically observed in the background but instead require the user to explicitly start and stop recording the route.

Personal motivation. Systems utilizing user-generated content usually have to deal with a potential problem called "cold start". Specifically, the cold start problem implies that the user has to dedicate an amount of effort using the system in its blank state – contributing to the construction of the

spatial database and their user profile – before the system can start providing full functionality.

Besides letting the system learn more about the user, the explicit user interaction step performed prior to uploading the observed data also serves another purpose. As discussed in section 2.6, users are willing to share location information for both personal and social purposes. By giving the users an overview of their past places and activities, Crumblr is accounting for the personal motivation for reviewing and reminiscing in the past events. This aspect should also make the system usable in its blank state.

Privacy issues. *Semi-automatically* collected data suffers less from privacy issues than fully automated user tracking such as employed by CityVoyager. Recent research on sharing location information has shown that people are not willing to share their home or work location, as already outlined. By letting the user take action during/after the aggregation process and giving them the control over the collected data before uploading it to a central authority, privacy issues become less critical. Users always have the option not to share specific place visits. Moreover, related systems providing location-based services employ some sort of notification mechanisms to alert the user about interesting events or messages in the user's surroundings. In order to accomplish this, the mobile client needs to periodically reveal the user's location and identity. Crumblr has been designed to never upload user's location data to the server without user's full awareness.

Activities. Besides places and routes, the activities play an important role in Crumblr as they constitute a core component of the user's long-term preferences. However, to make this kind of data fully applicable to formal models, the concept of activities is formalized by introducing an activity taxonomy. Table 4.1 shows the taxonomy of depth two, which is currently utilized by Crumblr. The design of this taxonomy has been driven by two conflicting goals: to optimize the user experience by reducing the depth and the overall size of the taxonomy on the one side, and to include all currently observable activities common to popular social media sites such as Qype and Flickr on the other side[1]. Being a trade-off between these two goals, the resulting proof-of-concept taxonomy therefore makes no claim of being complete. Moreover, the child activities are not complete refinements of their parent

[1]The activity taxonomy has been created within the scope of both the Crumblr and Captchr projects. The *Captchr* project has been developed by Matthias Käppler in the course of his diploma thesis at the University of Kaiserslautern. The relation between the two projects is outlined in the next chapter.

Nightlife & Events	Sports & Wellness	Infotainment
Clubbing	Tennis	Museums
Bars & Pubs	Soccer	Theaters
Party	Basketball	Shows
Get Together	Water & Beach	Fairs & Exhibitions
Music Events	Biking	Conf. & Conventions
	Baths & Spas	Animal Parks
	Jogging	
	Fitness	
	Bar Sports	

Food & Drink	Entertainment	Outdoor & Travel
Chinese Food	Cinemas	Leisure in Nature
Mexican Food	Amusement Parks	Walking
Italian Food	Gaming & Gambling	Hiking
Turkish Food	Adult Entertainment	Sightseeing
Greece Food		
Fast Food		
Cafes		

Table 4.1: The activity taxonomy used by Crumblr

activities. Parent activities represent subsuming abstractions of their child activities.

4.1.2 Aggregation of Spatial and Contextual Data

If two people have visited the same place, it is highly unlikely that the GPS data captured during these visits will be identical. In order to effectively recognize where the user has been, data from multiple place or route visits must be aggregated. Furthermore, when designing an approach to recommend places and routes, spatial data alone is not sufficient. Effective data aggregation techniques are considered indispensable and constitute a core concept in Crumblr.

Route model

When considering routes, the aggregation of GPS tracks is based on previous work presented in TopoFusion (cf. Chapter 3). Like TopoFusion, Crumblr creates aggregated route networks from a collection of overlapping and inter-

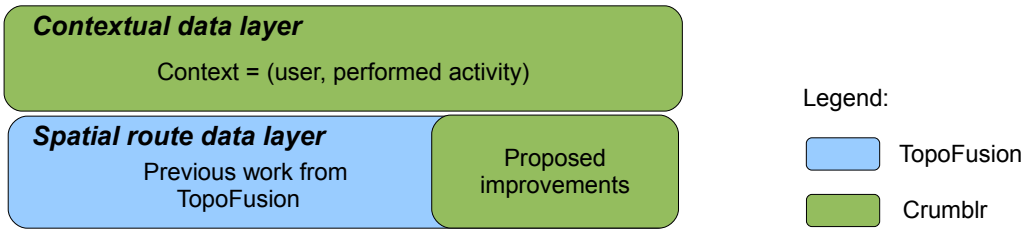


Figure 4.2: The two-layered route network model in Crumblr

secting GPS track logs. It identifies tracks that are actually representations of the same route or road and averages them to eliminate duplicates.

Several improvements to TopoFusion's aggregation algorithms are suggested and the route network model is extended with contextual data. The context elements captured for this purpose are the user ID and the activity performed on the route. This extension brings routes and places to the same conceptual level, making both entities suitable to similar recommendation models (cf. Figure 4.2). Uploaded GPS tracks are automatically aggregated in a network of routes (more precisely: route segments) that is stored on the Crumblr server. A detailed discussion about the network aggregation algorithms is given in Chapter 6.

Place model

Every place or route visit further describes the geographic properties such as the shape of the place or route in Crumblr. Information about the geographic shape of a place is used to improve the aforementioned visit recognition process. For example, when a user has visited a large place such as the Central Park in New York, a system that is aware of the park's geographic shape is able to correctly detect this visit with higher probability than a system that models a place as a single point in space. The latter place model is employed by all related projects, as already outlined in Chapter 3. The information about the geographic shapes is co-created by Crumblr's users – every uploaded place visit further describes the shape of the place. Figure 4.3 illustrates an example of the proposed method for updating place shapes. An existing place shape (green) is updated by a recognized place visit (orange), yielding the new "averaged" place shape (blue). Additionally, the effect of single updates on a place shape is limited by employing a specific weighting method. In theory, this approach should lead to the convergence of the computed shapes towards the real-world shapes of places. In practice, it is expected that this approach will significantly improve the suggestions

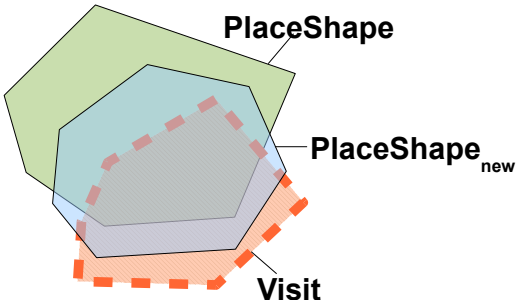


Figure 4.3: Old shape (green) and a new place visit (orange) result in new place shape (blue)

provided when recognizing place visits. A detailed discussion about the place shape aggregation algorithms is given in Chapter 6.

Aggregated spatial and contextual data is used to describe both the users and the spatial entities in Crumblr. By adding the notions of the user and the activity to the spatial model, additional value is derived from the aggregation of spatial and contextual data. On the one hand, every place or route visit provides additional clues about the user's interests. As already outlined, by aggregating this data over time, Crumblr estimates users' long-term interests and favorite places and routes. Similarities between users can be computed in order to yield personalized recommendations and to promote user interaction. On the other hand, every place or route visit further describes the places and routes as well. Besides the spatial characteristics, patterns in the contextual layer can be mined in order to reveal potentially interesting clues about further aspects of the place or route segment in question. Examples for such patterns are popularity trends, the distribution of visits over time, and the distribution of the performed activities.

To summarize this important and distinguishing idea behind Crumblr: the users describe the places and routes they visit, whereas in turn the visited places and routes describe the users as well.

4.1.3 Personalized Recommendations of Places and Routes

The previously stated goals, acquisition of long-term preferences and data aggregation, enable personalized recommendations of spatial content. To assist users on the move, information about places and routes is filtered according to the user's context. The user's current location, the purpose of use and the

user's movement history (long-term preferences) are the context dimensions used by Crumblr when recommending places and routes.

The recommendation process in Crumblr consists of two steps. First, the items (places or routes) are filtered according to the user's current location, an explicitly specified maximum distance to the current location, and the activity to be performed. Second, the filtered items are rated according to several components of the recommendation model. The filtered and rated items are returned to the mobile client, which is responsible for the visualization of items.

The recommendation models for places and routes are conceptually introduced below. The discussion assumes a general user of the system called "Alice" in need of place or route recommendations. The algorithms behind each component of the recommendation model will be described in more detail in Chapter 6.

Recommendation model for places

When recommending places for Alice, the following recommendation criteria are utilized in Crumblr:

- **Places preferred by *generally similar* users**: General user similarity is a concept realized by the Captchr system and will be discussed in section 6.6.1. Basically, if two users behave similarly with respect to the places they visit, the activities they engage in, and the time slots they are usually active in, these two users have high general similarity. Therefore, places preferred by users generally similar to Alice are rated high.

- **Activity similarity**: Places preferred by users similar with respect to the specified activity are rated high. For example, compared to Alice, the users having similar preferences regarding Pubs & Bars are considered a good choice for recommending other Pubs & Bars – their favorite places for Pubs & Bars are rated high, in this case.

- **Absolute popularity of places**: The total distribution of all users' visits in the region reveals the most frequented places with respect to the chosen activity. This popularity is absolute in the sense of neutrality to the user profiles attached to place visits – the place rating for each place depends on the total number of visits regardless of the users behind these visits.

Recommendation model for routes

By imposing a layer of contextual data, i.e. user identities and performed activities, on top of spatial data, the routes become applicable to various recommendation criteria as well. When recommending routes to Alice, Crumblr uses the following criteria to rate route segments in the aggregated route network:

- **Activity similarity**: Route segments preferred by users with similar preferences regarding the chosen route activity are rated high. For example, if Alice usually prefers the same jogging routes as Bob (maybe because they both have similar levels of fitness), Bob is considered similar to Alice regarding jogging. Bob's favorite route segments will be rated high, in this case.

- **Dedication to the chosen activity**: For each route segment, Crumblr calculates the percentage of the desired activity (e.g. jogging) compared to the total number of visits to this segment. The resulting rating for the segment is directly proportional to the calculated percentage. For example, a route segment might only be 26% dedicated to jogging, because there were other users who frequently used that route segment for other activities such as biking and hiking. One might want to avoid bikers and hikers on the road and therefore prefer other routes.

- **Distance to relevant places**: In addition to the specified route activity, Alice can specify what kind of places she would like to have along the routes. For example, while hiking, Alice might want to indulge in sightseeing. In this case, Crumblr's rating for a route segment suitable for hiking would correlate negatively with the segment's distance to sightseeing places[2].

- **Absolute popularity of route segments**: The total distribution of all users' visits in the region reveals the most frequented route segments with respect to the chosen activity. This popularity is absolute in the sense of neutrality to the user profiles attached to the segment visits – the segment rating for each place is directly proportional to the total number of visits to this segment.

[2]Crumblr does not explicitly categorize places or routes– an item's suitability for an activity is derived from the aggregated history data.

4.1.4 Adaptive Visualization

Adaptive visualization techniques need to be employed on the mobile client to enable quick decisions, addressing the limited time available for user interaction in mobile environments. Therefore, in addition to filtering techniques, Crumblr adapts the visual representation of recommended items to their estimated relevance to the user – the lower the estimated relevance, the more transparent the item appears on the map display. As the user can quickly spot the most relevant items on the map, this technique promises to accelerate the user's decision process.

Additionally, a structured visualization approach consisting of several individually selectable layers on the geographic map is chosen as a means to explain the estimated relevance of recommended items to the user. Instead of trying to combine the individual ratings in a single, less meaningful number, Crumblr lets the user individually choose from the rating criteria. Each layer reflects a single component of the recommendation model (e.g. "distance to relevant places", "preferred by similar users"). Furthermore, textual explanations of the estimated relevances are constructed on the server and displayed on the mobile client. The explanations are driven by the nature of the utilized component of the recommendation model. For example, when recommending favorite places from users having similar "Cafes" preferences, the identities of the most similar users are displayed on demand, promoting further interaction and socializing between users as well.

By combining visualization techniques with textual explanations, the complex recommendation model is expected to become transparent to the user, increasing the system's trustworthiness and overall acceptance. For example, Magitti does not provide any kind of explanation for the recommended items and completely hides the components of the recommendation model. A user study around Magitti showed that the users miss explanations and perceive the application as not trustworthy [Bellotti et al., 2008].

4.1.5 Summary

Figure 4.4 provides a visual presentation of Crumblr's life cycle. The users' spatial behavior is semi-automatically captured via mobile clients. It is aggregated and stored on the server, enabling personalized recommendations of shared content. After getting recommendations, the users visit places and engage in activities, implicitly closing the feedback loop.

Crumblr makes no claim of implementing the best approach to address all the outlined challenges – the main goal of this project is a prototypical implementation of the proposed ideas on a novel platform for mobile devices,

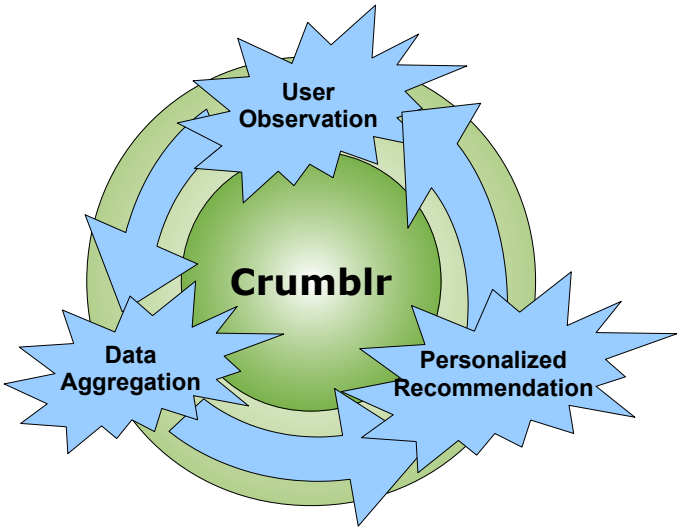


Figure 4.4: Crumblr's life cycle

Google Android.

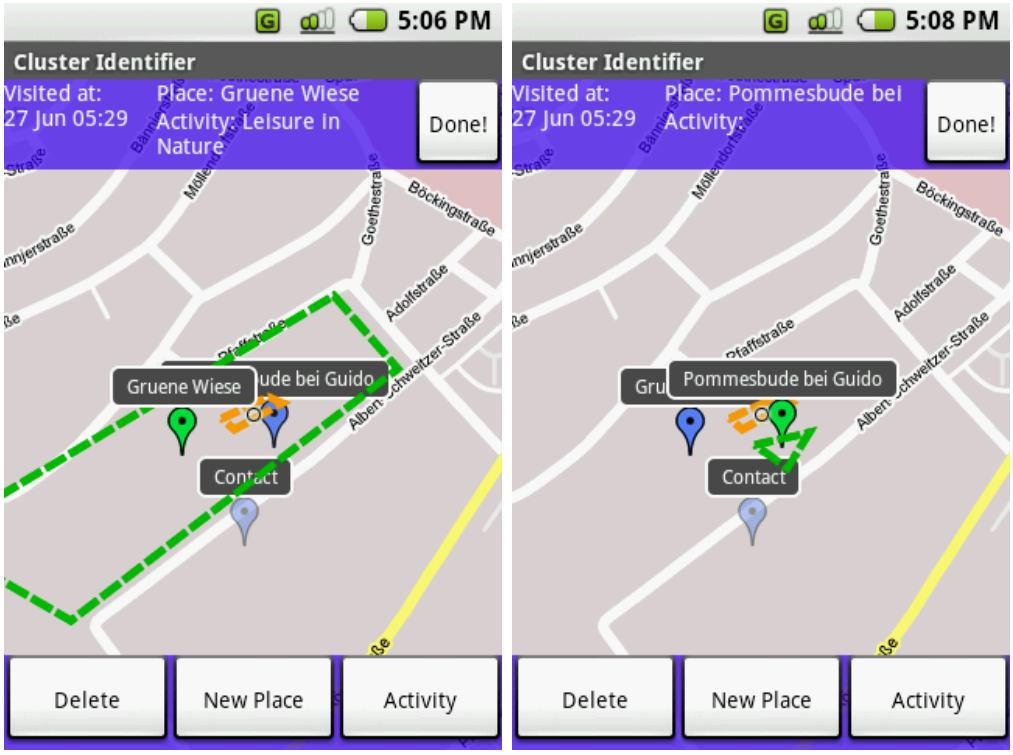
The project's name, "Crumblr", is based on the term "bread crumbs": similar to dropping a bread crumb trail as the user travels across the landscape, a GPS trace of sampled locations is collected. Crumblr analyzes these bread crumb trails in order to extract objectively observable local knowledge in form of preferences with respect to places and routes. By building a descriptive model of the involved entities (users, activities, places, routes), personalized recommendations can be provided in order to assist users on the move.

4.2 Use Cases

After the core concepts have been introduced, this section demonstrates Crumblr's user interface and use cases via user stories. A discussion about the implementation-related aspects such as the system's architecture, the technologies utilized, and the algorithms employed 'behind the scenes' is given in the following chapters.

4.2.1 Semi-Automatic Recognition of Place Visits

Alice has decided to follow a friend's advice and give Crumblr a try. After installation, the application spawns a background service which is constantly monitoring Alice's location via the integrated GPS module. Crumblr does



(a) Crumblr has recognized a visit to "Gruene Wiese".

(b) Of course, the user can select another nearby place if Crumblr's recognition was wrong.

Figure 4.5: Selecting the correct place

not require network connectivity when collecting location data. The recognition of place visits is executed while the device's battery is charging, under the condition that a reasonable amount of location data has been collected.

After a few days the application notifies Alice that it needs her attention – the Crumblr Visit Recognition Service has completed its work and a small notification icon has been displayed on the status bar. This service is the part of Crumblr responsible for automatically recognizing visits to places. After clicking on the notification icon, the Cluster Identifier (CI) dialog is launched, as shown in Figure 4.5(a). The purpose of the CI dialog is to guide the user through the semi-automatic visit recognition process (cf. section 4.1.1). By using the navigation pad on her mobile device, Alice can tap through the recognized place visits. If she wants to delete a visit (i.e. she does not want to share it), she can tap on the "Delete" button on the screen.

In the upper part of the screen, details about the selected place visit are

displayed. A list of visit timestamps[3], the title of the associated place and the performed activity are provided to the user so that they are always aware of the data that will be uploaded to the Crumblr server.

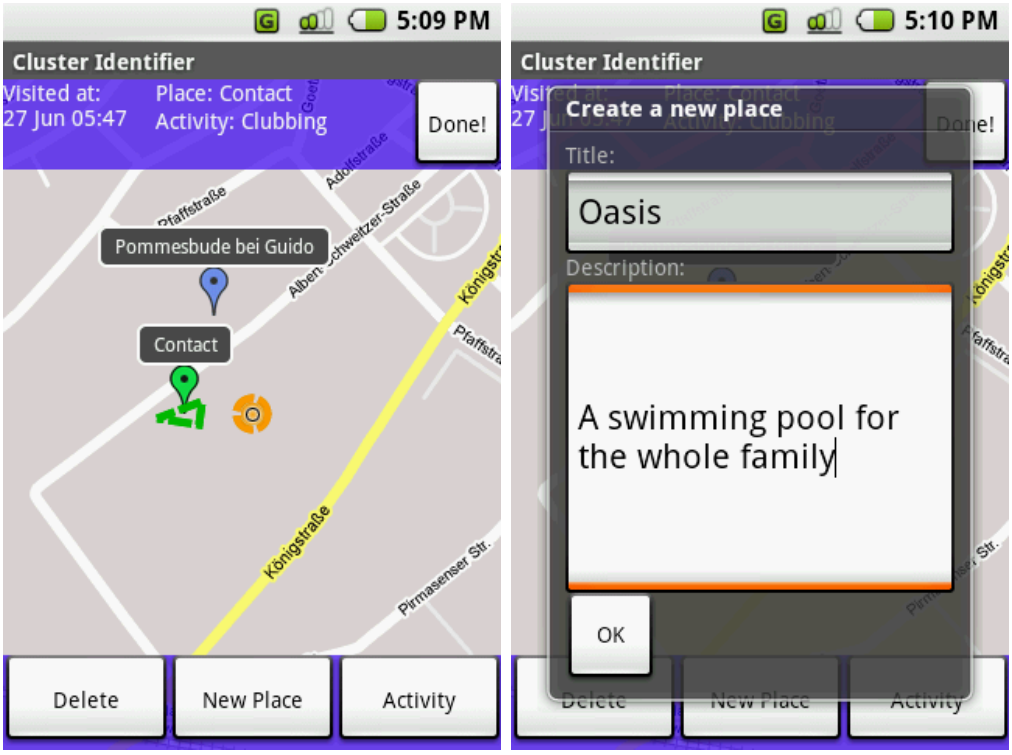
For each recognized place visit the system draws the shape of the region corresponding to the visit on the map (dashed orange lines in the center of the screen) and suggests a place for it. It is up to Alice to either approve or correct this suggestion by either tapping to the next place visit (by pressing "up" or "down" on the nav-pad) or selecting another place (by pressing "left" or "right" on the nav-pad). The currently associated place is displayed in green color, consisting of the balloon-like place marker and a dashed contour line, similar to the place visit itself. The place contour helps users estimate the geographical shape of places such as parks or larger areas, whereas simple markers fail to do so effectively. For example, the shape of the park "Gruene Wiese" is a lot larger than the shape of "Pommesbude bei Guido" (Figures 4.5(a) and 4.5(b)).

Moreover, the system adjusts the opacity of the place markers according to the distance between the place region and the estimated visit region. The closer the place region is to the visit region, the more opaque the icon appears. This measure is supposed to accelerate the process of selecting the right place by letting the user focus on the most probable places. Consider again the example in Figure 4.5 – even though the place marker for "Pommesbude bei Guido" is closer to the recognized place visit region (orange lines), the visit region is completely contained in the region of "Gruene Wiese". This makes the latter the best candidate place for the visit in question. There is a third place in the screenshot, "Contact" – its opacity is very low because Crumblr considers it not being very probable for the currently selected visit. Alice has indeed visited "Gruene Wiese" and does not need to change the suggested place.

By pressing "up", the dialog takes Alice to the next recognized place visit (Figure 4.6(a)). This time, "Contact" is considered the best place candidate and is therefore selected. Crumblr was wrong in this case, however, and Alice cycles through the other surrounding places. There is also an option to create a new place by tapping the corresponding button on the screen, being exactly what Alice wants to do in this case. In Figure 4.6(b), Alice provides the title and a short description of the new place "Oasis", which is automatically associated with the currently selected place visit (cf. Figure 4.7(a)).

Besides the visited place, the system tries to recognize the activity Alice has performed for each recognized visit. Alice has, again, two choices – ei-

[3]Crumblr recognizes and stores multiple visits to the same place.

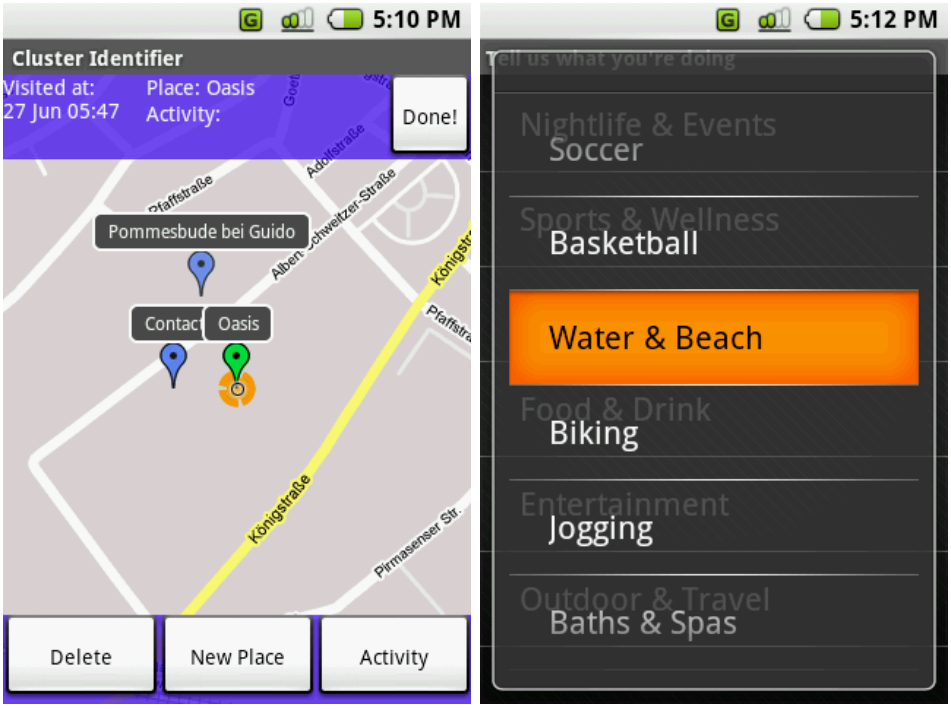


(a) If Crumblr's database is missing a place... (b) ...the user can easily create a new one.

Figure 4.6: Creating a new place

ther the system's guess is right or Alice has to select the correct activity by tapping the "Activity" button. The latter action spawns a new list dialog to select an activity from the predefined taxonomy (cf. Figure 4.7(b) and Table 4.1). The activity selection dialog has been carefully designed to make this process as effortless as possible – reducing the maximum number of actions to go through all available activities was the primary goal here. For example, with only two clicks Alice is able to select "Water & Beach" from the activity taxonomy (cf. Figure 4.7(b)).

Alice is satisfied with the place visits and activities recognized by Crumblr. Furthermore, Alice has discovered several nice places during the last few days. Being confident about Crumblr's visit recognition capabilities, she is glad that the process of sharing her personal preferences and interesting places with the community is only a matter of few clicks. After reviewing her visits, Alice taps on the "Done!" button in the upper right corner and the provided data is sent to the Crumblr server. In the meanwhile, the background process on the mobile device continues to observe Alice's movement.



(a) After creating a new place...

(b) ...the user selects the performed activity.

Figure 4.7: Selecting the activity for the new place

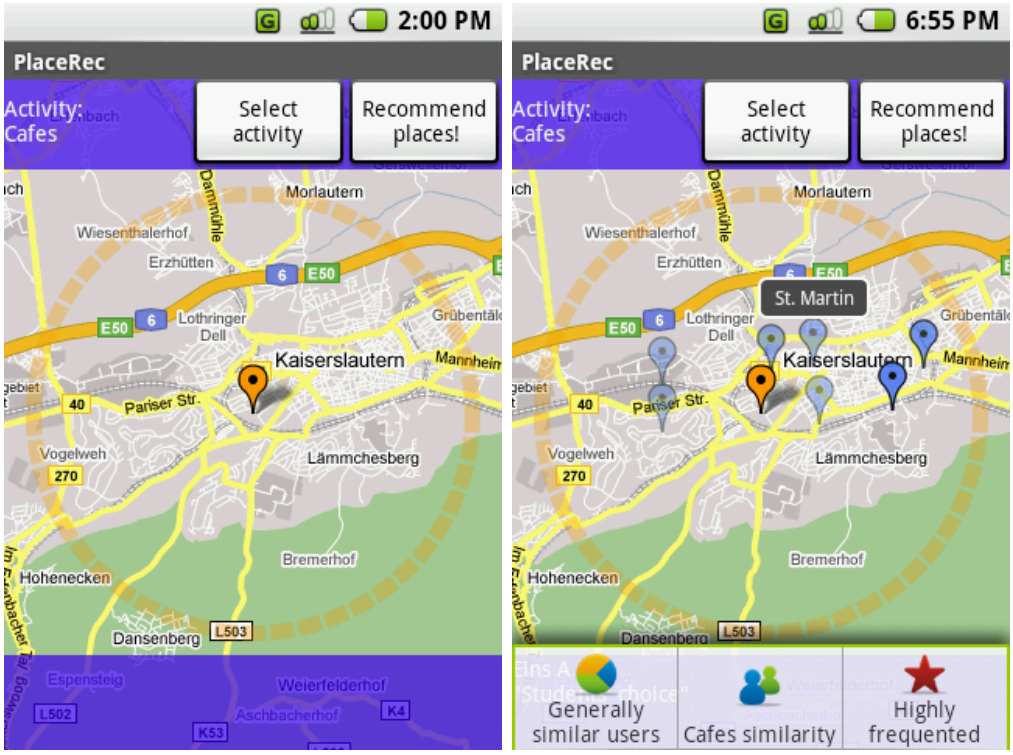
After a few weeks (depending on how active Alice is) Crumblr will have collected enough data and the described process will be initiated again.

4.2.2 Route Recording

To capture routes (or visits to routes), Crumblr requires the user to explicitly start and stop a "Route Recording" mode. After stopping the Route Recording mode, Crumblr presents the recorded route on a map screen. Alice only needs to provide the activity (as described in Figure 4.7(b)) and everything can be uploaded to the server. Since the routes are automatically aggregated on the Crumblr server, no further user interaction is required.

4.2.3 Recommending Places

Since Crumblr is aware of Alice's current location, Alice can get location-based place recommendations for an activity she selects. Initially, when opening the Place Recommendation dialog, Alice looks at the screen in Fig-

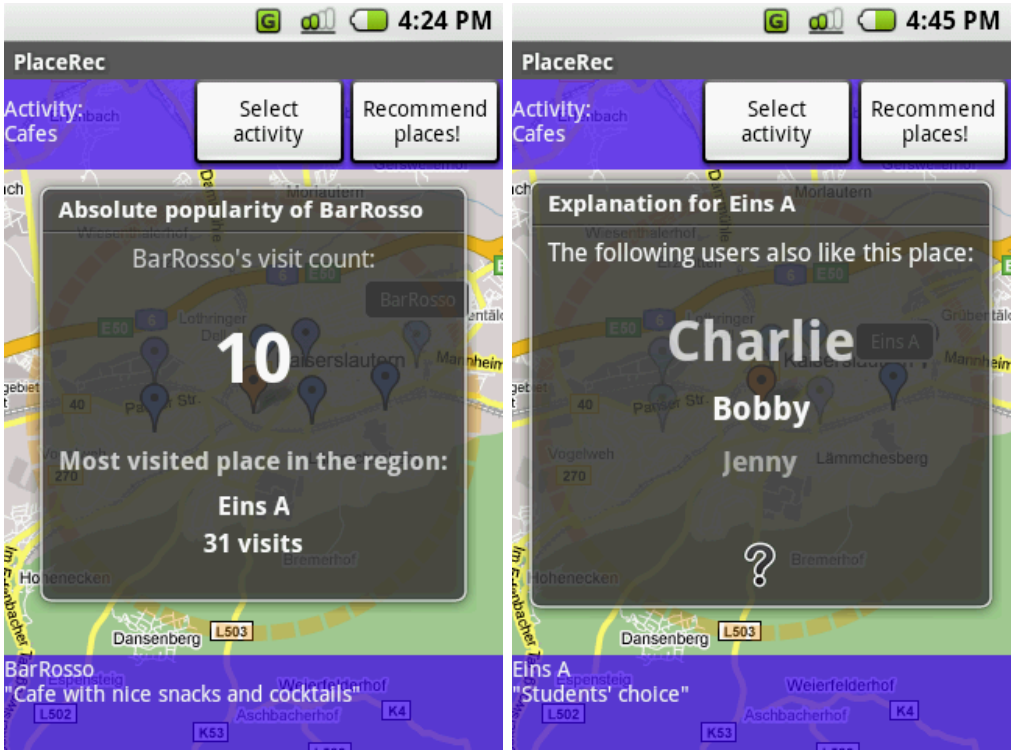


(a) Place Recommendations dialog (b) Cycling through recommended places

Figure 4.8: Place recommendations

ure 4.8(a). Alice can press "up" to increase or "down" to decrease the radius of interest around her current location. Crumblr will only consider places that fall within this radius. It is displayed as a dashed orange-colored circle around Alice's current position.

After setting the radius of interest and selecting the desired activity "Cafes" (a process already described), Alice presses the "Recommend places" button. After a few seconds, seven place recommendations are retrieved from the Crumblr server and displayed as depicted in Figure 4.8(b). Places considered less interesting for Alice are displayed with a low opacity. By using the navigation pad, Alice can cycle through the recommended places. Each place is displayed as a balloon-like marker, with the place title shown above the selected marker and a brief description in the lower part of the screen. For example, Figure 4.8(b) shows Alice having selected the place "BarRosso" – its opacity is medium, indicating a moderate relevance to Alice. The opacity of a place depends on the currently active layer of the recommendation model. To view and choose from all available layers (i.e. recommendation criteria), Alice can press the "Menu" button on her



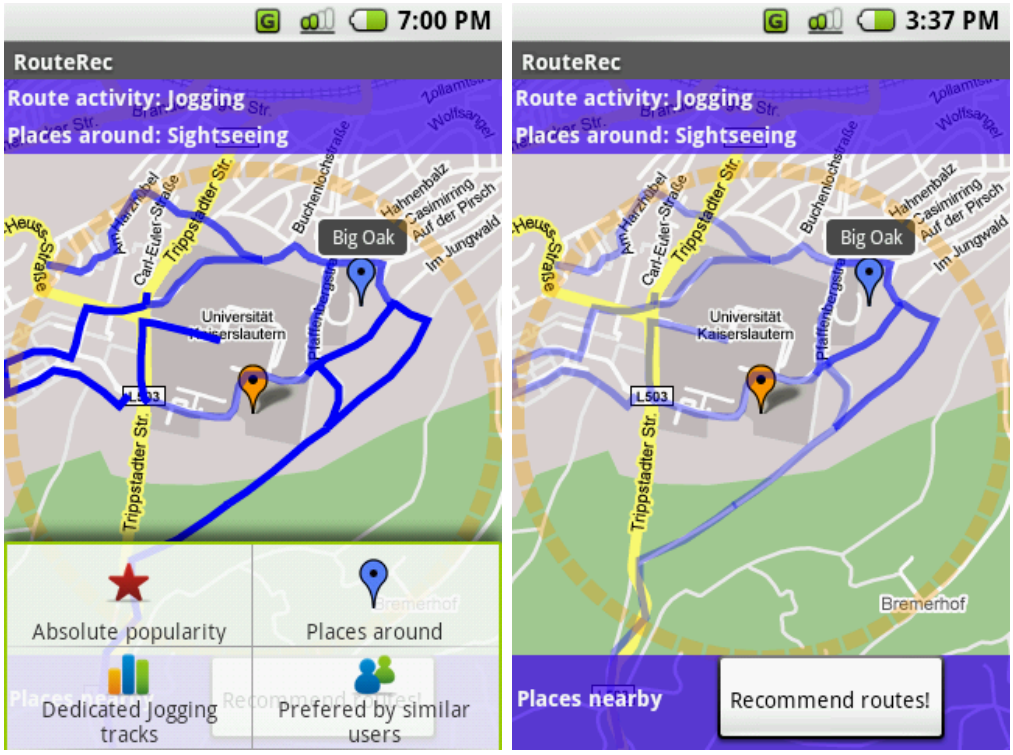
(a) Explanation for BarRosso's low popularity

(b) Reasons for Eins A's high "general similarity" rating

Figure 4.9: Place explanations

device. This lets a small menu slide into the screen, showing the available recommendation criteria (cf. the lower part of Figure 4.8(b)). Exactly one layer can be active at any time – by changing it, the opacity of the places changes as well, reflecting their estimated relevance corresponding to the currently active component of the recommendation model. Crumblr fosters the serendipity factor by setting a low opacity level for places that Alice has already visited. This way, places unknown to Alice are visually emphasized.

As in all map-based dialogs in Google Android, Alice can zoom and pan over the map by using gestures with her finger (i.e. by performing drag-and-release moves). By pressing the "center" button, Alice can see an explanation of the estimated relevance calculated by Crumblr. For example, by selecting the layer "Absolute popularity" and requesting explanations for the the low popularity rating of "BarRosso", a dialog is displayed as shown in Figure 4.9(a). Apparently, "BarRosso" is significantly less frequented than "Eins A". Figure 4.9(b) shows the explanation for the place "Eins A" and the "General user similarity" component of the place recommendation model.



(a) Recommendation model menu

(b) "Sightseeing" places around "Jogging" routes

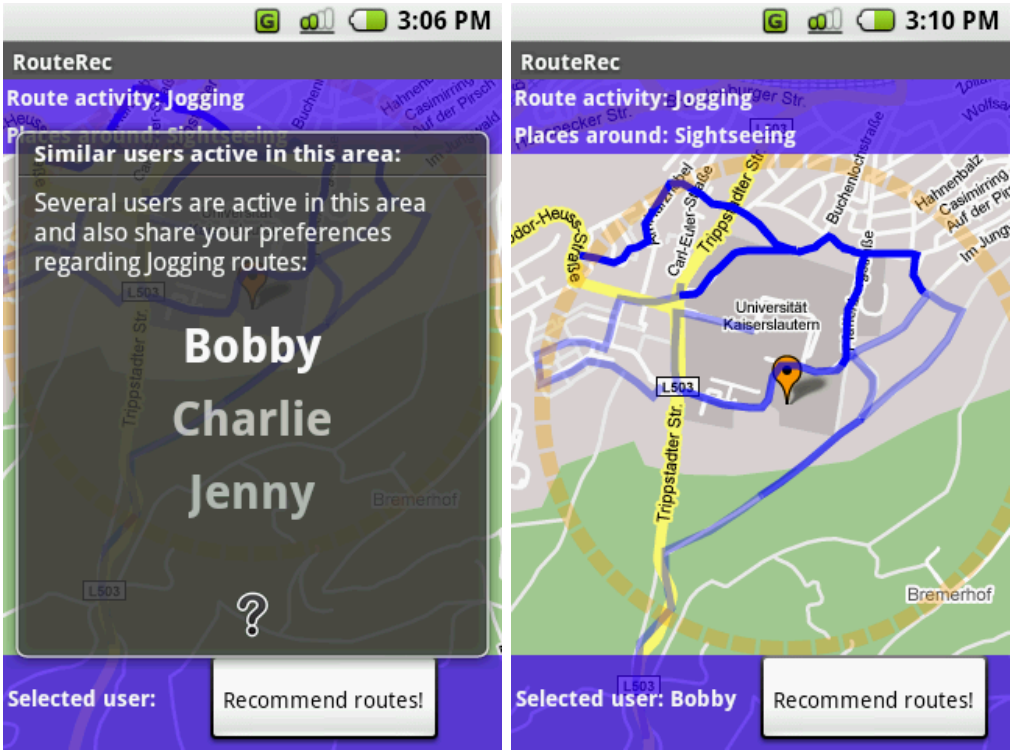
Figure 4.10: Route recommendations and recommendation layers

According to the displayed explanation window, the user Charlie behaves rather similarly to Alice (indicated by the opacity of his name), and he also visited this place more often than any other user (indicated by the font size). The small font size for Jenny expresses her low visit frequency to "Eins A" when compared to the other visitors, while the low opacity of the font relates to Jenny's low general similarity to Alice. By clicking on the question mark symbol in the lower part of the dialog, the rationale behind the approach of scaling the font size and opacity is displayed.

A comprehensive discussion of the place recommendation model and the used explanation approaches is given in section 6.6.

4.2.4 Recommending Routes

The user interaction in the Route Recommendation dialog is similar to the Place Recommendation dialog described in the previous section. However, due to the differences in the recommendation models described in section



(a) Selecting a similar user (b) Viewing Bobby's favorite routes

Figure 4.11: Explaining "Preferred by similar users" ratings

4.1.3, other input parameters must be obtained from Alice in order to retrieve interesting routes from Crumblr.

Alice adjusts the radius of interest and presses the "Recommend Routes" button to select the the desired route activity and the activity describing the places Alice wants to have near the routes. After pressing "OK" and waiting a few seconds for the server to process the request and return the route segments, the recommendation results are displayed as shown in Figure 4.10. Crumblr retrieves the part of the route network that lies in the region specified by the radius of interest. Similar to the Place Recommendation dialog, there is exactly one layer selected describing one component of the recommendation model. Figure 4.10(a) shows the pop-up menu for the available components of the route recommendation model. Each segment's opacity level is adjusted to the calculated rating, corresponding to the currently active layer. By switching the active layer to "Places around", Alice can see which route segments are close to "Sightseeing" places (Figure 4.10(b)). The places are shown as blue markers in order to provide a better explanation for the estimated relevance and the resulting opacity of each segment. By

clicking on the place marker, its title is displayed.

Figure 4.11(a) shows the result of selecting the layer "Preferred by similar users": a list of the most similar users who were active in the given region is displayed. The varying opacity expresses the calculated "Jogging" similarity between Alice and the other users. By clicking on the question mark symbol in the lower part, the rationale behind the approach of scaling the opacity is displayed. Let us assume that Alice is interested in the favorite routes of the most similar user in the region, Bobby. By clicking on his name, Bobby's preferred route segments are emphasized as shown in Figure 4.11(b). Alice is able to cycle through the favorite routes of other users in a similar fashion.

A comprehensive discussion of the route recommendation model and the explanation approaches is given in section 6.6.

The consistent use of opacities and separate layers for both places and routes is supposed to increase the overall comprehensibility and usability of the mobile client. By switching layers, the users can choose from different, personalized views on the aggregated, shared content.

Summary

This chapter covered the core aspects of Crumblr from a conceptual point of view. It finally provided a comprehensive demonstration of the implemented prototype, showing how the presented concepts map to the user experience on the mobile device. The next chapter discusses aspects related to the technological foundations and system design.

Chapter 5

Technological Foundations and System Design

After the main concepts of Crumblr have been introduced and the user interface has been described in detail, this chapter turns to the aspects related to system design and technological decisions. First, the utilized technologies are described in detail, as the underlying frameworks set the scope Crumblr was designed within. Second, the top-level solution architecture and the software architecture of Crumblr's client and server are discussed. In a nutshell, this chapter outlines Crumblr's technological foundations and building blocks before turning to its core algorithms in the next chapter.

5.1 Technological Foundations

This section discusses the rationale behind the main technological choices that have been made during the project. Google Android and the Grails framework are the two core platforms that enabled a rapid development process and made it possible to develop a fully functional Crumblr prototype within time constraints inherent in a diploma thesis project.

Google Android

Section 2.5 briefly introduced some of the mobile platforms that are likely to play a major role in the mobile computing world in the years to come. Table 5.1 presents an evaluation matrix having two dimensions – the mobile platforms that have been evaluated for the project's purposes, and the evaluation criteria that were derived from project characteristics and constraints such as development skills, needed platform services, programming

Evaluation criteria	iPhone	Google Android	LiMo	Windows Mobile
SDK available	yes	yes	no	yes
Openness of the platform	limited	yes	yes	limited
Java API	no	yes	no	yes
Devices available	yes	no	yes	yes
Large, healthy community	yes	yes	no	yes
Full-stack software	yes	yes	no	yes
Background services supported	no	yes	yes	with MDIP3
Modularization & interoperability	bad	good	?	bad
Cost	free	free	free	depends on tools used
Platform maturity	immature	immature	immature	mature

Table 5.1: Evaluation of existing mobile platforms, partially based on [Kumparak, 2008]

languages, available documentation, etc.

The support for background services (i.e. processes running in the background without a UI) was considered critical for Crumblr since the client software needs to operate continuously without user interaction. This is the main reason why the iPhone SDK was not the best choice for Crumblr, although Apple's platform is gaining a lot of momentum among developers recently. Android is the only platform having no real devices available, yet – the first Android-enabled devices will be launched by the end of 2008. Nevertheless, developers can deploy and test their applications on an emulator, coupled with a solid plug-in for the popular Eclipse development environment.

Compelling arguments for Android are its Java-based SDK, the full-stack platform, the flexibility and richness of the programming interfaces, and the fact that it is open and supported by leading companies in the mobile industry. The Java-based SDK allows for leveraging existing knowledge, libraries,

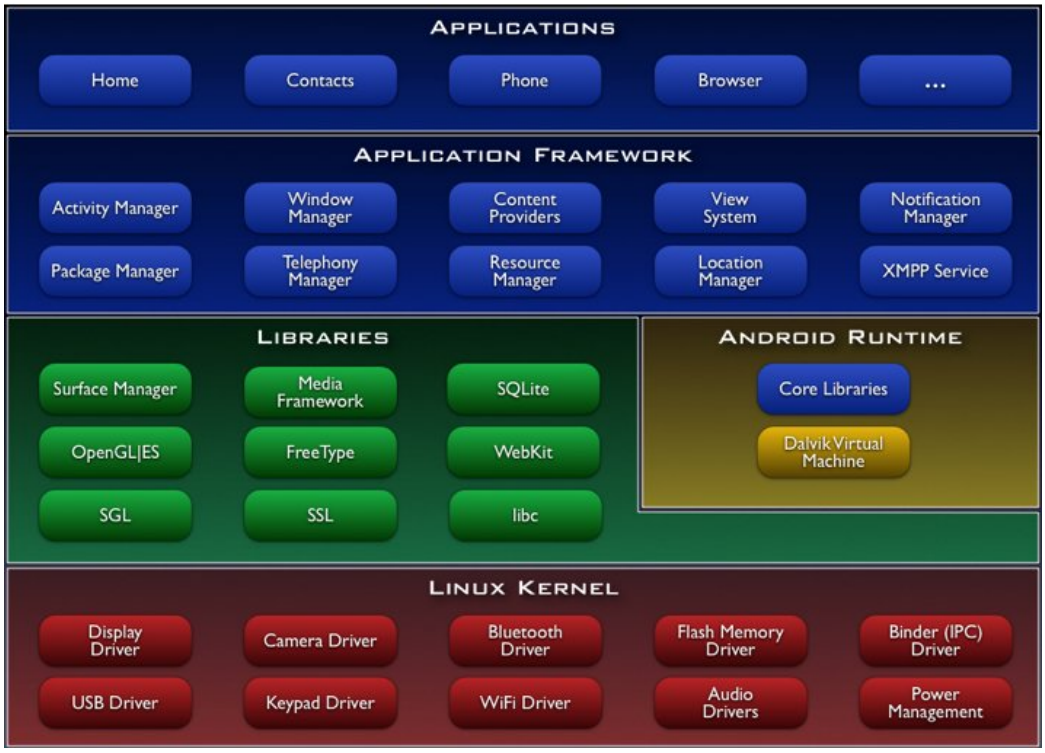


Figure 5.1: The Google Android platform [Google, 2007]

and tools centered around the Java programming language.

Figure 5.1 shows the major components of the Android platform. Android is a software stack for mobile devices that includes a Linux-based operating system, middleware, and key applications.

Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Even though Android does not provide a Java virtual machine (JVM), the Android SDK provides the tools and API's necessary to begin developing applications on the Android platform using the Java programming language. The Dalvik VM runs compiled Java classed by transforming them into the .dex format understood by the Dalvik VM. Going up the stack, Android includes a set of C/C++ libraries used by various components of the Android system. These capabilities are exposed to developers through the Android application framework. Underlying all applications is a set of services and systems, including a rich and extensible set of GUI widgets. Finally, Android ships with a set of core applications including an email client, SMS program, calendar, maps, browser, contacts, and others. All applications are written using the Java programming language. Some of

Android's most innovative aspects result from its key design principles which are described next [Google, 2007].

Reusability

Developers have full access to the same framework APIs used by the core applications. The application architecture is designed to simplify the reuse of components; any application can publish its capabilities and any other application may then make use of these capabilities (subject to security constraints enforced by the framework). This same mechanism allows the user to replace components.

Interoperability and Loose Coupling

The Android platform provides a powerful message-oriented middleware, aiming at loose coupling and seamless interoperability between application components. This architectural feature is based on *Intents*. An Intent is a simple message object that represents an "intention" to do something. For example, if an application wants to display a web page, it expresses its "Intent" to view the URI by creating an Intent instance and handing it off to the system. The system locates some other piece of code (in this case, the Browser) that knows how to handle that Intent, and runs it. Intents can also be used to broadcast interesting events (such as a notification) system-wide.

Long-Running Background Tasks

Crumblr's aforementioned need for long-running background tasks on the mobile client is appropriately addressed by Android's *Services*. A Service is a process that runs in the background. Other components "bind" to a Service and invoke methods on it via remote procedure calls. *Notifications* are a convenient mechanism for alerting the user of something that needs their attention – for example, after Crumblr's clustering process has finished recognizing place visits. A Notification is a small icon that appears in the status bar. Users can interact with this icon to receive information.

The described evaluation of available platforms and the aforementioned unique aspects of Google's platform make Android the mobile environment of choice for Crumblr's mobile client.

Groovy

Groovy is an object-oriented programming language for the Java Platform as an alternative to the Java programming language. It is a dynamic language with features similar to those of Python, Ruby, Perl, and Smalltalk. It can be used as a scripting language for the Java Platform. Groovy uses a Java-like curly bracket syntax which is dynamically compiled to Java Virtual Machine bytecodes and which works seamlessly with other Java code and libraries. Most Java code is valid Groovy syntax and can be used dynamically as a scripting language.

> *Whereas the Java language has won over an entire generation of programmers with its commitment to exactitude and extensiveness, Groovy heralds a new era of programming on the Java platform, one defined by convenience, expedience, and agility.* [Glover, 2004]

Code written in Groovy is easy to read and maintain, and reduces scaffolding code when developing web applications. A plug-in for the Eclipse development environment exists as well. The Crumblr server has been entirely written in the Groovy programming language.

Grails

Grails[1] is an open-source, rapid web application development framework that provides a full-stack programming model based on the Groovy scripting language. Built on top of Spring, Hibernate, Sitemesh, and other "best of breed" Java frameworks, Grails is intended to be a high-productivity framework by following the "convention over configuration" paradigm, providing a standalone development environment and hiding much of the configuration detail from the developer.

Grails provides a powerful and consistent persistence framework, easy to use view templates using JSP and GSP (Groovy Server Pages), dynamic tag libraries to create custom web page components, and good Ajax support. The Grails framework takes away the need to add configuration in XML files. Instead, Grails uses a set of rules (conventions) while inspecting the code of Grails-based applications. For example, a class name which ends with "Controller" (for example BookController) is considered a web controller. The domain model in Grails is persisted to the database using GORM (Grails Object Relational Mapping), which uses Hibernate behind

[1] http://grails.org

the scenes. Because of the dynamic nature of Groovy and the conventions of Grails, there is almost no configuration involved in creating persistent domain classes. Among its most interesting features is Grails support for automatic marshaling of Java objects to JSON[2] or XML and vice versa, and its excellent support for Web Services and HTTP request parsing. Excellent literature for getting acquainted with Grails is the book *Getting Started with Grails* from Jason Rudolph [Rudolph, 2007].

To conclude, opting for Grails as the web development framework for the Groovy programming language was an obvious choice, considering the benefits outlined above. Crumblr's entire server infrastructure is based on Grails.

5.2 Solution Architecture

Figure 5.2 depicts the top-level solution architecture of the system, including the organization of utilized technologies and frameworks. The current prototype is a client-server based application consisting of one server and many instances of the mobile client. The prototype is entirely based on open source software.

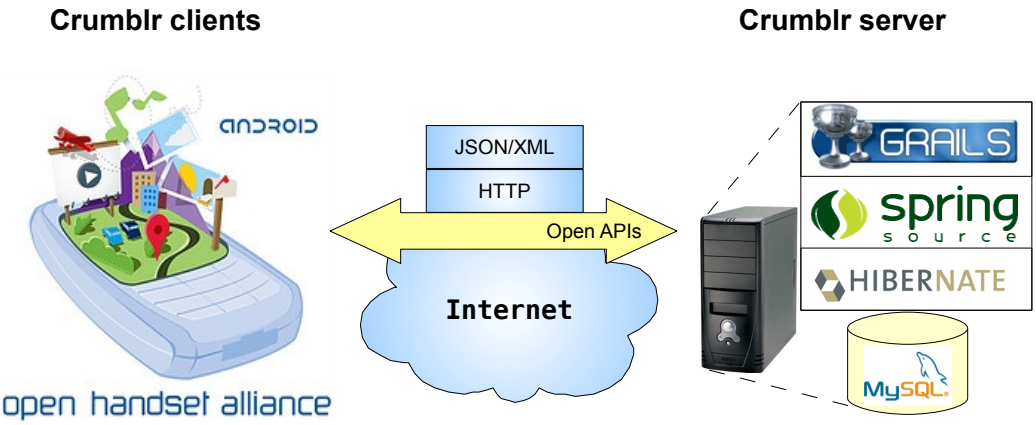


Figure 5.2: The solution architecture

While the mobile client has been built on Google Android, the server is running as a web application based on the Grails framework. A MySQL database is used for persisting user data and shared content, which is exchanged via JSON messages transmitted by HTTP. All interfaces offered by

[2]Java Script Object Notation

the server utilize established standards, making Crumblr's server infrastructure open and reusable by other applications and services.

Having outlined the "big picture", the discussion now turns to Crumblr's client and server parts.

5.2.1 Client Architecture

Figure 5.3 presents the client's high-level architecture. The client is structured in three parts – the Android Activities, Android Services, and a Resource Layer. Each part's structure and main responsibilities are described below.

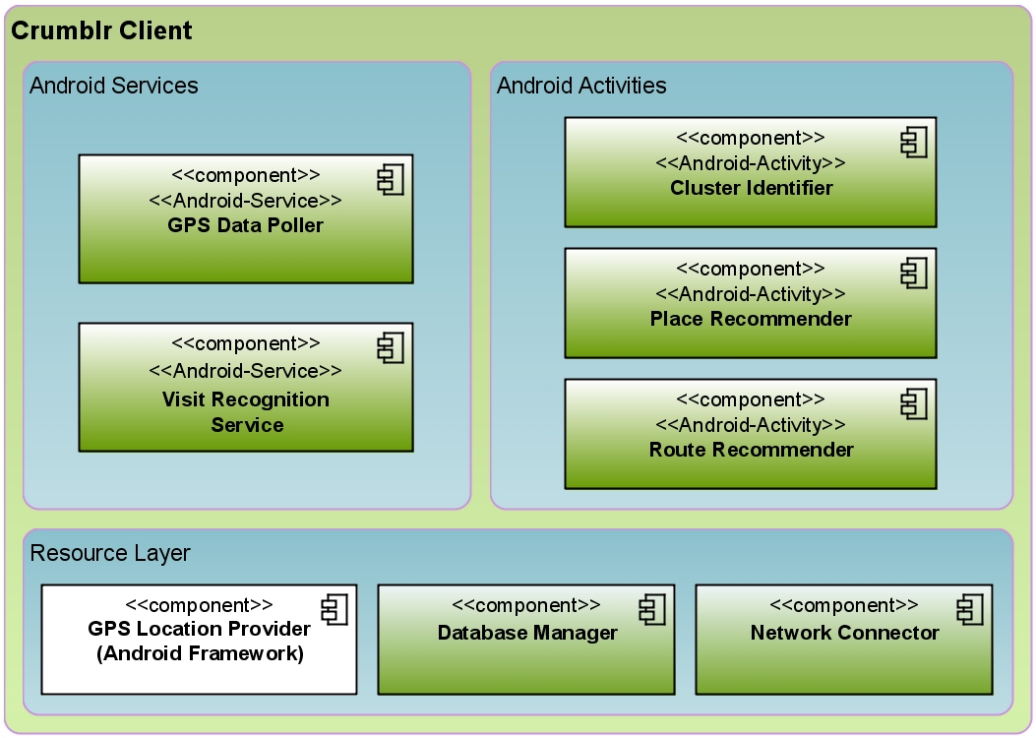


Figure 5.3: Crumblr client architecture – Crumblr's components are green-colored.

The Resource Layer consists of several components that are used by both the Services and Activities. The *Location Provider* component is part of Android's Application Framework. It is basically a wrapper for the GPS module. The *Database Manager* is responsible for managing persistent data such as user settings, the collected GPS data, and the

recognized place visits. The database used by Android is SQLite3, representing the main data synchronization point for the used Services and Activities. The *Network Connector* module is used by those parts of the client that require communication with the server. The Connector module utilizes asynchronous HTTP communication, allowing for non-blocking callback mechanisms when communicating with the server.

The Services part contains two Android Services – the GPS Data Poller and the Visit Recognition Service. The *GPS Data Poller* has been designed to continuously collect GPS position data and store it directly in the database via the Database Manager. The *Visit Recognition Service* gets activated by the Android platform when the phone's battery is charging. This approach is feasible since the visit recognition can be performed anytime. Upon activation, the Visit Recognition Service uses the Database Manager to retrieve any stored and unprocessed GPS data. This data is fed into Crumblr's algorithm for detecting place visits. The visit recognition algorithm is a background task which spawns a system notification upon completion. By clicking on the notification icon, the Cluster Identifier Activity is launched, which is described below.

The Activities part contains several components responsible for user interaction. An Android Activity is usually a single screen in an application. Crumblr's Android Activities use the Database Manager to store and load application and user data. The network communication is handled by the Network Connector. The *Cluster Identifier* is the component responsible for managing the process of semi-automatic recognition of place visits. It presents a visual summary of the recognized place visits and activities on a map-based view. As described in section 4.1.1, the user is given control over the recognized places and activities, before uploading the data to the server via the Network Connector. The *Place Recommender* and the *Route Recommender* are Android Activities responsible for fetching place and route recommendations from the server and displaying them to the user. The visual appearances of both Activities have been described in detail in section 4.2.

5.2.2 Server Architecture

The architecture of the Crumblr server is depicted in Figure 5.4. It is based on a classic layered architectural style adapted to the Grails terminology,

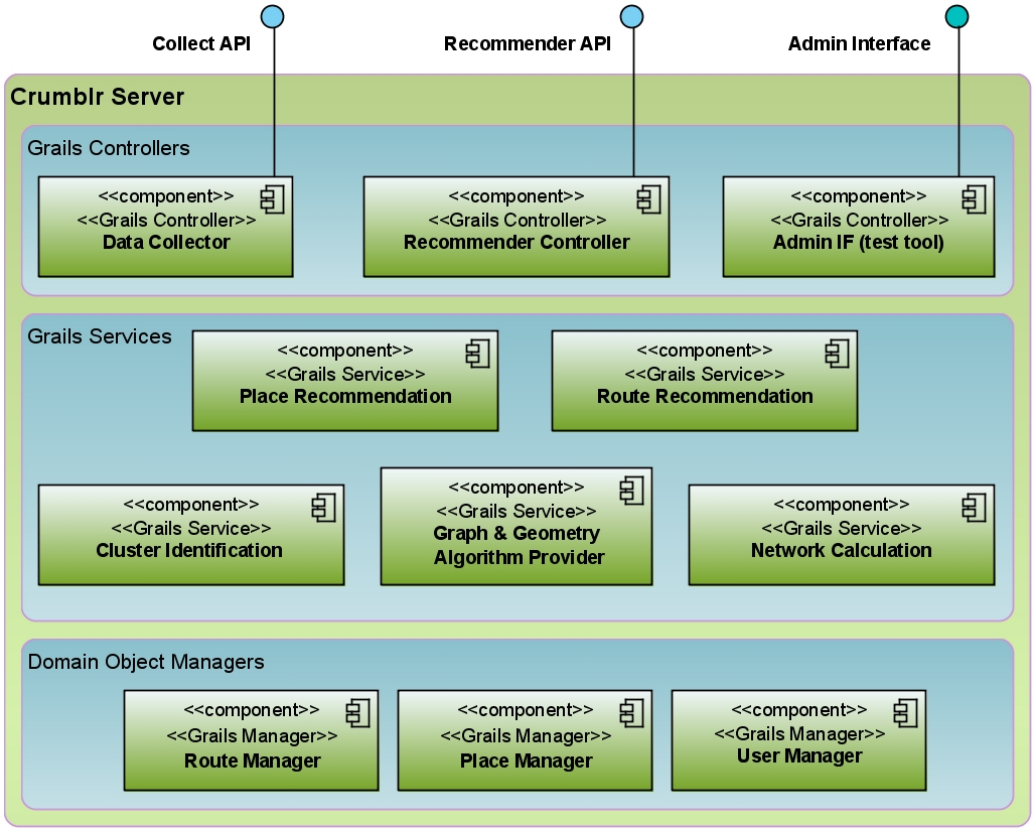


Figure 5.4: Crumblr server architecture

consisting of Grails Domain Object Managers, Grails Services, and Grails Controllers.

The Domain Object Managers are responsible for managing the persisted data model, including spatial and contextual data discussed in previous chapters. The most important entities (users, routes, places) are managed by their corresponding *Managers*. In a nutshell, the managers offer basic persistence-related operations to the upper layers.

The Grails Services Layer consists of several components encapsulating Crumblr's core application logic and algorithms. The *Cluster Identification* service is responsible for associating place visits to places. It is also responsible for estimating performed activities and updating place shapes. The *Network Calculation* service implements Crumblr's route aggregation algorithms for maintaining a library of route segments. Both the Cluster Identification service and the Network Calculation service make use of the *Graph & Geometry Algorithm Provider*, which

encapsulates various basic algorithms needed by the former services in an extensible way. The *Place Recommendation* service and the *Route Recommendation* service are responsible for executing the recommendation models described in section 6.6. As they both heavily rely on algorithmic tasks, they utilize the Algorithm Provider service as well.

The Grails Controllers Layer contains the components responsible for accepting incoming HTTP requests, authentication, input validation, calling the requested services, and finally rendering the response to the client. There are two main categories of interfaces offered by the Controllers Layer: the *Collect API* (for uploading data about visits to places or routes) and the *Recommender API* (for requesting recommendations). Figure 5.4 shows a special *Admin Interface* and its controller that have been implemented for internal testing purposes. The Admin Interface has become a full-fledged tool for a flexible management of test scenarios; it will be briefly described in section 5.2.5.

5.2.3 Component Interaction Flows

Having outlined the responsibilities of components on both the client and the server side, an example of a component interaction flow is given in Figure 5.5. The figure illustrates the interaction flow describing Crumblr's semi-automatic visit recognition use case[3]. When charging the mobile device's battery, the Android operating system spawns an instance of the Visit Recognition Service (VRS). Upon activation, VRS fetches all available GPS position data from the Database Manager (step 1) and executes the algorithm for extracting place visits from GPS traces (step 3). It finally stores the data describing the recognized visits to the database and notifies the user about the completion (steps 4 and 5). Eventually, the user will open the notification and launch the Cluster Identifier Activity (CI) (step 6). The CI Activity fetches the recognized visits from the Database Manager (steps 7 and 8) and retrieves the suggested places and activities from the server for the recognized visits (steps 9 and 10). Specifically, when suggesting places and activities, the Cluster Identification service (server) utilizes the Graph & Geometry Algorithm Provider to calculate the most likely places and activities for each visit (not shown in figure). After selecting and confirming the places and activities (steps 11–13), CI uploads the content to the server.

[3]The example has been chosen due to its relative complexity – component interactions for other use cases are easily derived from the static description of the components' responsibilities and are therefore omitted.

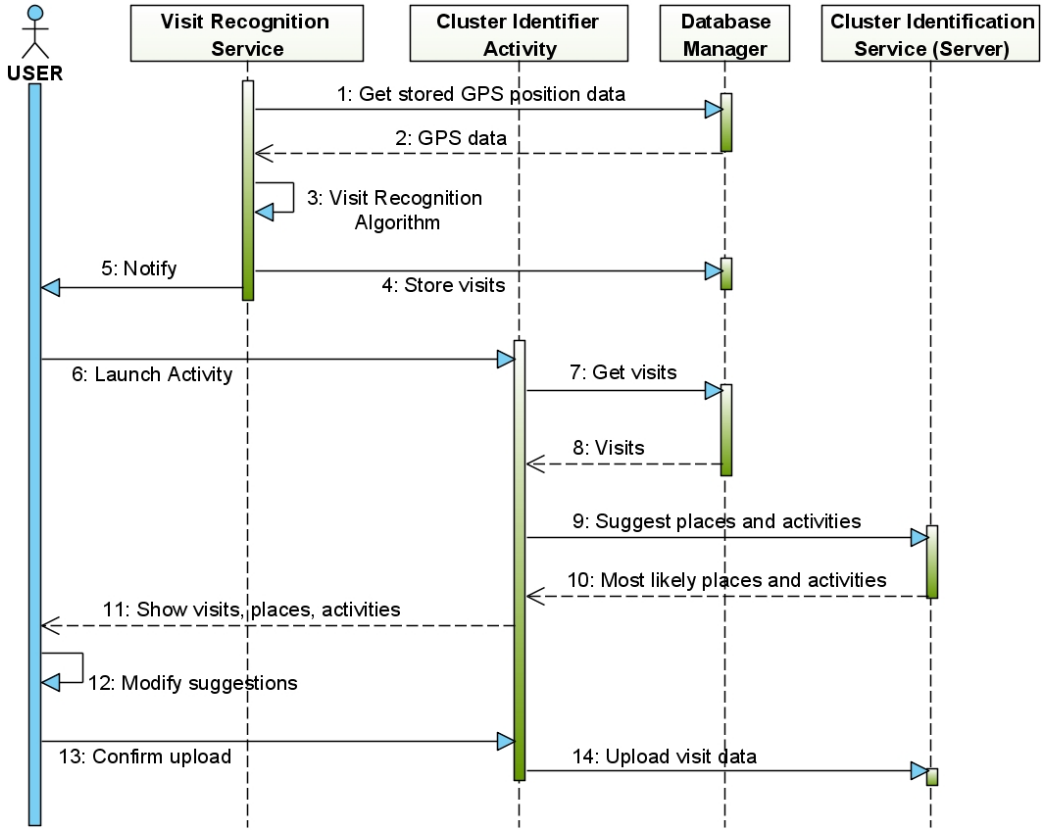


Figure 5.5: Crumblr's semi-automatic visit recognition process

Finally, the Cluster Identification service (server) updates the place shapes according to the uploaded visit data (not shown in figure).

5.2.4 Captchr

The *Captchr* project has been developed by Matthias Käppler in the course of his diploma thesis at the University of Kaiserslautern [Käppler, 2008]. Captchr is a mobile micro-blogging platform that takes proven Web 2.0 paradigms from the desktop PC and adapts them to mobile devices. By letting users explicitly attach additional contextual data to blog entries, such as the performed activity and current mood, Captchr exploits this contextual data to build long-term user profiles. At its core, Captchr leverages these profiles to compute social neighborhoods and to promote interaction between its users. A more detailed description of Captchr's notion of social neighborhoods is given in section 6.6.1.

Among Captchr's core domain entities are the users, the places, and the

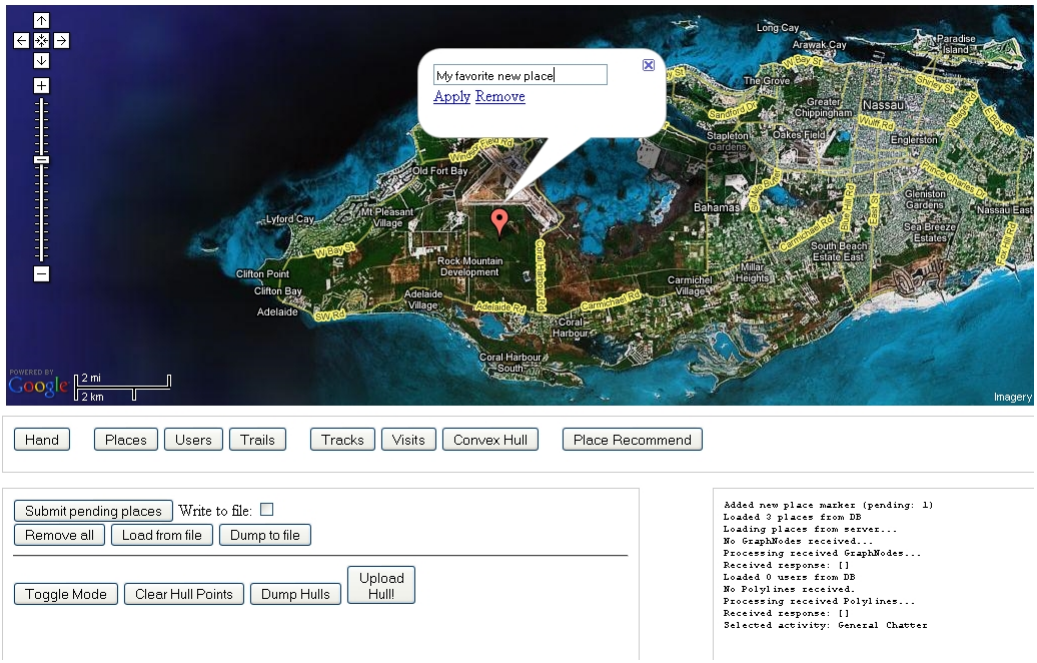


Figure 5.6: Crumblr's Admin Interface: creating a new place

performed activities. To benefit from this fact, Captchr and Crumblr share functionality. On the server side, the Domain Object Managers have been jointly developed in order to provide functionality needed by both systems. Similarly, the Resource Layer on the client side comprises functionality used by both client applications. On a higher level, a part of Crumblr's place recommendation model makes use of Captchr's User Similarity Service (cf. section 6.6.1). Several other ways of interaction between Captchr and Crumblr are thinkable – a selection of ideas about bringing the two systems closer together is given in Chapter 7.

5.2.5 Admin Interface

The "Admin Interface" component (cf. Figure 5.4) offers a HTML/JavaScript-based interface to several parts of Crumblr's server functionality. Initially developed for validating the implemented services and domain object managers, the Admin Interface has grown to a powerful tool for flexible test case management. Figure 5.6 shows a typical screenshot of the Admin Interface. It consists of three main areas: the Google Maps view in the upper part, the control buttons and input fields in the lower left part, and the status messages in the lower right part. The Admin Interface enables the

creation, modification, storing, and loading of complete data sets including users, places, routes, and visits. Moreover, the implemented geometric algorithms, graph reduction methods, and parts of the recommendation model can be extensively tested as well. Screenshots showing more of Admin Interface's functionality can be found in Appendix C.

5.3 Design Aspects

This section discusses some of the key non-functional aspects that further characterize a system's design, i.e. considerations regarding security, performance, and usability.

5.3.1 Security

Being open, web-based, and service-oriented, Crumblr has been designed with security considerations in mind. Crumblr employs HTTP Basic Authentication and a Grails plug-in on the server side for enforcing authorization rules. All users must authenticate themselves before accessing any protected functionality on the server. Confidentiality and data integrity are provided by HTTPS/SSL, which will be natively supported by the final version of Google Android. However, it should be emphasized that security aspects were not among the primary development concerns. Security threats including forgery of information, denial of service attacks, or valid but disruptive data currently present challenges to many typical Web 2.0 applications, including the prototypical implementation of Crumblr.

5.3.2 Performance

Network effects (cf. section 2.1) usually consume massive system resources. Essentially, successful platforms relying on user contributions are permanently under friendly distributed denial of service attacks (DDoS). After reaching a certain level of popularity, such platforms are faced with *scalability* and *availability* challenges. Scalability is a desirable property of a system, a network, or a process, which indicates its ability to either handle growing amounts of work in a graceful manner, or to be readily enlarged. Availability is the proportion of time a system is in a functioning condition [Bondi, 2000]. Common solutions to these challenges typically involve high hardware and software costs. Scalability and availability were not among the primary development concerns, as they do not represent urgent issues for the prototype.

5.3.3 Usability

Chapter 2.2 identified several challenges an application designed for mobile devices should address: small size, short interaction times, and limited power. Crumblr's user interface has been designed to account for these considerations:

Limited space: On Crumblr's map-based views, only the most important items are shown at any time. When associating place visits with the Cluster Identifier dialog, only the most likely places for each recognized place visit are displayed. This reduces the perceived information overload. The displayed place and route recommendations are visually structured by using varying opacity levels and a layer-based organization of recommendation models. On demand, more detailed explanations are shown in new dialog windows.

Limited time: Considering short interaction time, Crumblr's algorithms have been designed to automatically estimate the visited places and performed activities; this way, the required user actions when sharing place visits are significantly reduced. As the aggregated data grows over time, the correctness of estimated places and activities should improve.

User input: To account for the size and time limitations, the complete basic functionality in all Crumblr's dialogs is accessible via the touch screen and the navigation pad. Furthermore, the design of the activity taxonomy was driven with usability considerations in mind as well. Due to the low taxonomy depth, the user is able to select any activity with at most two input actions.

Battery impact: In order to minimize the impact of the long-running, resource consuming Visit Recognition Service on the mobile client's battery, the VRS is configured to get activated by the Android platform when the battery is being loaded.

Chapter 6

Algorithms and Models

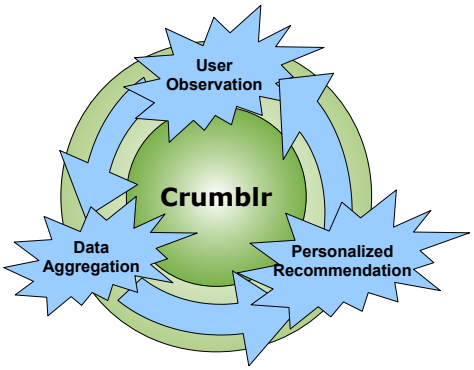


Figure 6.1: Crumblr's life cycle model

Having described the system's design, this chapter provides a detailed analysis of the employed algorithms for semi-automatic collection of user data, aggregating spatial and contextual data, and recommending places and routes. The discussion follows the life cycle model outlined in section 4.1.5 (cf. Figure 6.1). For each analyzed algorithm class, related work is presented first before outlining the approach employed by Crumblr.

6.1 Extracting Place Visits From GPS Data

As already described in section 4.1.1, Crumblr aims at recognizing place visits from sampled GPS data. Basically, place visits can be extracted from GPS data by looking for drop-outs of the GPS signal or by recognizing regions where many consecutive location measurements are clustered together.

6.1.1 The comMotion Recurring GPS Dropout Algorithm

Early work on extracting place visits from GPS data used loss of signal to infer the location of indoor places. One of the first systems dealing with detecting place visits is comMotion [Marmasse and Schm, 2000], utilizing a device that constantly takes GPS readings. The fact that most buildings are GPS-opaque was exploited advantageously, permitting a simple mechanism for learning the locations of buildings. If the GPS signal is lost, this is interpreted as a significant cue, namely that a building has been entered. The system maintains a history of readings, and when the signal has been lost within a given radius on three different occasions the application infers that this location (building) is interesting. Once a location has been discovered and accepted by the user, the user can attach a to-do list to it – a prototypical example is associating a shopping list with a supermarket.

The requirement for the user to go to a place at least three times before it can be recognized was found to be overly restrictive [Hightower et al., 2005]. Moreover, since the indicator for identifying a location is the loss of the GPS signal, only indoor locations can be found. Some outdoor places, such as parks or sidewalk cafes, may not cause GPS signal loss, and thus cannot be discovered. Conversely, in so-called "urban canyons" between tall buildings or in tunnels, GPS signals are often weak and unreliable, which could trigger false positives. Furthermore, the use of a fixed radius for delimiting places may be problematic: the size and shape of places like a cafe, one's home or place of work can vary widely [Zhou et al., 2004]. Finally, the algorithm does not deal with capturing place shapes or identifying the places the user has visited.

6.1.2 k-Means Clustering

Multiple GPS measurements in the same location do not necessarily yield the same coordinates due to errors and variations in the measured phenomena, i.e. GPS signal travel times. Thus, the places where the user spends considerable time appear as clusters of locations in the GPS traces rather than as single points. Identifying densely clustered regions from the trace is basically a clustering problem.

There are two basic types of clustering algorithms [Kaufman and Rousseeuw, 1990]: partitioning and hierarchical algorithms. *Partitioning algorithms* partition a set D of n objects into a set of k clusters. k is an input parameter for these algorithms, i.e some domain knowledge is required which unfortunately is not available for many applications. The

partitioning algorithm typically starts with an initial partition of D and then uses an iterative control strategy to optimize an objective function providing a quality measure for the clusters. *Hierarchical algorithms* create an iterative hierarchical decomposition of D. The hierarchical decomposition is represented by a *dendrogram*, a tree that iteratively splits D into smaller subsets until each subset consists of only one object. In such a hierarchy, each node of the tree represents a cluster of D. In contrast to partitioning algorithms, hierarchical algorithms do not need k as an input. However, a termination condition has to be defined indicating when the decomposition process should be terminated [Kaufman and Rousseeuw, 1990].

k-means is a popular clustering method belonging to the class of partitioning algorithms. It minimizes an error term which is the sum of squared distances from each point to its cluster center. In formal notation, the error term to be minimized is

$$E = \sum_{i=1}^{k} \sum_{x \in C_i} d(x, m_i)^2 \tag{6.1}$$

where k is the number of clusters, m_i is the center of cluster C_i, and $d(x, m_i)$ is the Euclidean distance between a point x and m_i. The algorithm initially assigns all points to a predefined number of clusters k randomly. Then it iterates over each point, finds the cluster center nearest to that point, and assigns the point to the cluster that the center belongs to. This iteration is repeated until the error term 6.1 is below a given threshold.

Evaluation. Generally, k-means clustering has several drawbacks for detecting place visits [Zhou et al., 2004]:

- It needs to know the number of clusters beforehand. This number could be difficult to determine in general. It could, for example, be approximated by looking for a "knee" in the graph plotting the error term 6.1 against the number of clusters k.

- All points are included in the clusters, which makes the results sensitive to noise. A single noisy location reading far from other points can significantly pull a cluster center towards it, since the squared-distance error term 6.1 heavily weights distant outliers.

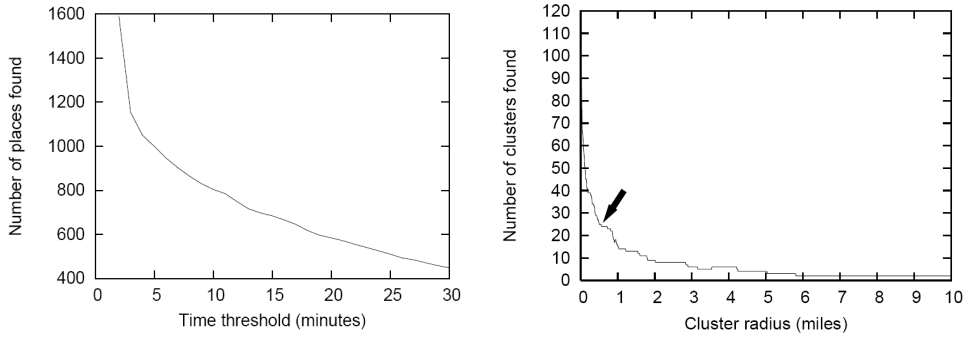
- The k-means algorithm is non-deterministic: the final clustering depends on the initial random assignment of points to clusters.

- k-means favors circularly shaped places. However, points from a student's location history on a university campus might easily form an irregular shape.

Ashbrook and Starner's k-Means Variant

In [Ashbrook, 2002], Ashbrook and Starner logged position data only while the user was traveling at speeds greater than one mile per hour. Since they were mostly interested in locations where the users spent time, rather than how they got there, the authors looked for time gaps in the data that indicate that the user stopped moving. The same time gaps also occur when the GPS receiver cannot find any GPS satellites, such as when the user enters a building. Whenever a point is found that has more than a certain time t between it and the previous point, it is concluded that the point marks a place visit.

In order to decide what value to choose for t, Ashbrook and Starner plotted the number of places found for many values of t on a graph (cf. Figure 6.2(a)) and looked for an obvious point at which to choose t. The authors decided on ten minutes as an amount of stopping time t that users might reasonably consider significant. Due to GPS errors multiple visits to the same



(a) Number of place visits found for varying values of t

(b) Number of locations found as cluster radius changes

Figure 6.2: Varying time thresholds and radii [Ashbrook, 2002]

place will be separated by only a few meters. For this reason, Ashbrook and Starner created clusters of place visit points using a variant of the k-means clustering algorithm. The basic idea is to take one point p and a predefined radius r. All the points within this radius of the point p are marked, and the mean of these points is determined. The mean is then taken as the new center point p', and the process is repeated. This continues until p' stops changing. Once the mean no longer moves, all points within the radius r are

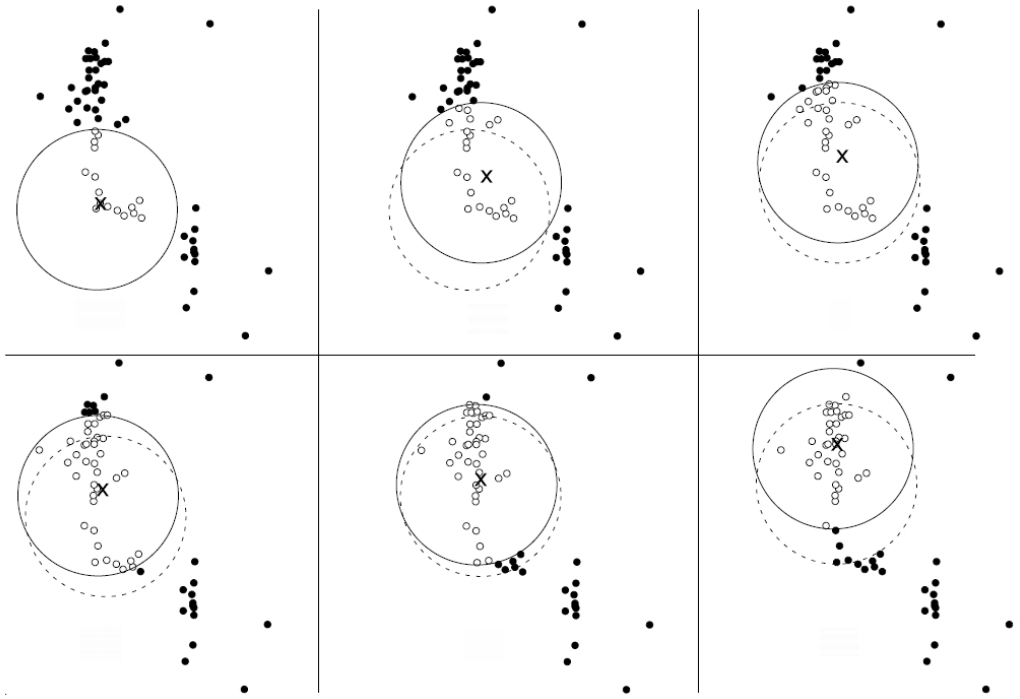


Figure 6.3: Illustration of the location clustering algorithm [Ashbrook, 2002]

placed in a cluster and removed from consideration. The procedure repeats until no points remain and the algorithm terminates with a collection of clusters describing places. An illustration of this is shown in Figure 6.3. The X denotes the center of the cluster. The white dots are the points within the cluster, and the dotted line shows the location of the cluster from the previous step. In the last step (lower right), the mean has stopped moving, so all of the white points will be part of this location [Ashbrook, 2002].

As already mentioned, the radius r needs to be predefined. By making the radius too small, the algorithm will end up with only one point per cluster. On the other hand, if the radius is too large, unrelated places would be grouped together, such as home and the grocery store. In order to find a good value for the radius, the clustering algorithm was run several times with varying radii. The results were plotted on a graph as depicted in Figure 6.2(b). The arrow denotes a knee in the graph, indicating the radius value just before the number of clusters begins to converge to the number of places [Ashbrook, 2002].

Evaluation. When finding places, Ashbrook and Starner's initial approach considered a point as a place visit if it had time t between it and the previous point. This basically means that places would be detected when the user exited a building and the GPS receiver re-acquired the signal. The improved algorithm published in [Ashbrook and Starner, 2003], however, registers a place when the signal is lost, and so is not dependent upon signal acquisition time.

While not having the problem of choosing the number of clusters, this specific variant of k-means clustering still has significant drawbacks for the purpose of capturing place shapes. First, stationary data about place shapes is lost by logging position data only when the user is moving. Second, by only looking for time gaps between two points larger than t, the algorithm cannot recognize urban canyons between tall buildings, resulting in false positives and reduced precision. Finally, the fixed radius r is a very hard constraint, considering the different sizes and shapes of places.

6.1.3 Accumulative Clustering

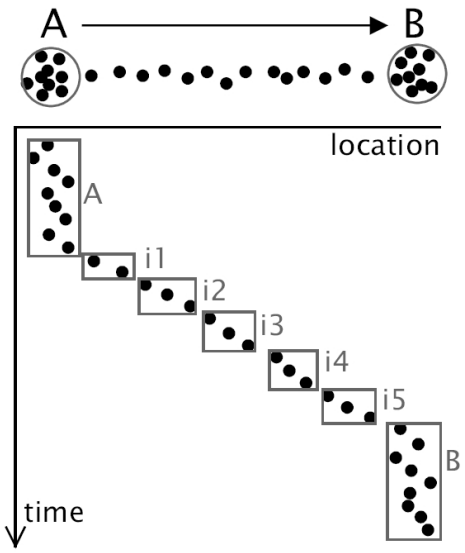


Figure 6.4: An illustration of Kang et al.'s accumulative clustering algorithm [Kang et al., 2004]

The well-known clustering algorithms such as k-means have problems when dealing with noisy GPS data and arbitrary place shapes. Kang et al. designed an algorithm that extracts places by applying accumulative clustering on traces of location data [Kang et al., 2004]. It takes a stream of timestamped coordinates derived from any location system as input and performs clustering and merging simultaneously.

Specifically, the algorithm compares each incoming coordinate with previous coordinates in the current cluster – if the stream of coordinates moves away from the current cluster then a new one is formed. Figure 6.4 illustrates this process. The algorithm takes as input a distance threshold d. Suppose that a user Alice moves from place A to place B. While at place A, her location coordinates are close together, i.e. within the distance d of each other, and so belong to

one cluster A. As Alice moves toward place B, her coordinates move away from cluster A, and a few small intermediate clusters are generated, clusters $i1$, $i2$, $i3$, $i4$, and $i5$. A short time after arriving at place B, cluster B is formed. If a cluster's time duration is longer than some threshold t, the cluster is considered to be a significant place. Therefore, clusters A and B are considered significant places while the smaller clusters are ignored [Kang et al., 2004].

When a cluster is added to the set of significant places, the algorithm checks the merging condition: if the cluster's centroid is within $d/3$ of an existing place, the cluster is merged with that place; otherwise it is added as a new place. A merging threshold smaller than d is sufficient because the distance between the clusters' centroids tend to be smaller than the maximum distance between individual coordinates.

Evaluation. Unlike other clustering algorithms that require offline clustering of complete location traces, the accumulative clustering algorithm from Kang et al. computes clusters incrementally as new location estimates are generated. The algorithm has been evaluated with 700 hours of real trace data: the algorithm extracted most place visits successfully [Kang et al., 2004]. However, the high frequent data sampling (1 reading per second) could potentially degrade the performance and consume a significant amount of battery from resource-limited mobile devices. Moreover, signal loss events indicating visits to indoor places cannot be detected.

6.1.4 Density-based Clustering

Density-based clustering uses the density of local *neighborhoods* of points [Ester et al., 1996]. Density-based clustering uses a notion of connectivity of that neighborhood, whose points eventually form a cluster. Each cluster has a considerably higher density of points than areas outside of the cluster. An example of density-based clustering is shown in Figure 6.5.

DJ-Clustering is a density-based algorithm proposed in [Zhou et al., 2004]. There are two parameters used to define density in DJ-Clustering: *Eps*, the radius of a circle, and *MinPts*, the minimum number of points within that circle. For each point, its neighborhood is calculated: the neighborhood consists of points within distance *Eps*, under the condition that there are at least *MinPts* of them (cf. Def. 6.1.1). If no such neighborhood is found, the point is labeled as noise. If a neighborhood is found and no neighbor is in an existing cluster, the points are added to a new cluster. If a neighbor is in an existing cluster c, the neighborhood is joined with c (cf. Def. 6.1.2). Since

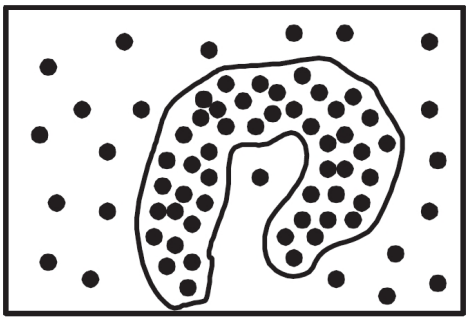


Figure 6.5: A density-based approach forms a cluster where point density is high [Ester et al., 1996]

DJ-Clustering serves as a basis for the algorithms discussed in the following sections, a pseudo-code of DJ-Clustering is outlined in Algorithm 1.

Algorithm 1 DJ-Clustering from [Zhou et al., 2004]

```
1: while there is an unprocessed point p from sample S do
2:    Compute the density-based neighborhood N(p) wrt Eps, MinPts
3:    if N(p) is empty then
4:       Label p as noise
5:    else if N(p) is density-joinable to an existing cluster then
6:       Merge N(p) and all the density-joinable clusters
7:    else
8:       Create a new cluster C based on N(p)
9:    end if
10: end while
11: return All clusters C_i
```

Definition 6.1.1 (Density-based neighborhood) Let S be a set of location points and $p \in S$. Let $H(p)$ be the candidate neighborhood of p defined as:

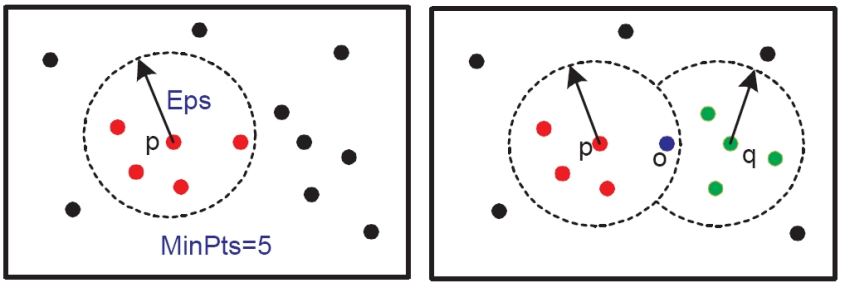
$$H(p) = \{q \in S | dist(p, q) \leq Eps\}$$

$H(p)$ is called the density-based neighborhood N of the point p, denoted by $N(p)$, if

$$|H(p)| \geq MinPts$$

where Eps is the radius of a circle around p which defines the density, and $MinPts$ is the minimum number of points required in that circle. An example for a density-based neighborhood is depicted in Figure 6.6(a).

Definition 6.1.2 (Density-joinable) Let $N(p)$ and $N(q)$ be the density-based neighborhoods of points $p, q \in S$, respectively. $N(p)$ is density-joinable to $N(q)$, denoted as $J(N(p), N(q))$, if there is a point o such that both $N(p)$ and $N(q)$ contain o. A density-joinable relation is illustrated in Figure 6.6(b).



(a) Density – $N(p)$ (b) Density-joinable

Figure 6.6: Density-based join concept [Zhou et al., 2005b]

Eps and *MinPts* combined determine the density of the neighborhoods and thus the size and shape of the clusters. To discover smaller clusters and a larger number of clusters, one can decrease both parameters. To discover personal places from GPS data, *Eps* may be set to approximate the uncertainty in GPS readings, e.g. to 20 meters [Zhou et al., 2004]. Values for *MinPts* range depend on the data sampling interval; higher values mean that clusters must be more dense in order to be formed. In general, this will have the effect of increasing the precision of the discovered places while decreasing the recall.

Evaluation. In [Zhou et al., 2004], the advantages of density-based algorithms over k-means clustering approaches are summarized as follows:

- DJ-Clustering allows clusters of arbitrary shape,
- robustly ignores outliers, noise, and unusual points,
- it is easier to choose reasonable parameter values that do not depend on the user, and
- it provides deterministic results.

However, an evaluation in [Hightower et al., 2005] revealed a weakness in DJ-Clustering due to its indifference to the time dimension. Its goal to account for large regions results in a lack of sharp demarcation between place

visits. Consider the following case: user Alice visits a sidewalk cafe C_1 in the morning; in the evening, she visits another cafe C_2 that is right next to C_1. DJ-Clustering will detect two clusters and merge them, since they are close to each other and thus density-joinable.

6.1.5 Density-based Temporal Clustering

In [Zhou et al., 2005b], two extensions of DJ-Clustering were proposed: join-based temporal clustering (TDJ) and TDJ with relaxed temporal constraints (R-TDJ). The algorithms treat time as the third dimension and take a threshold for the time elapsed between two points as the third input, $deltaT$, along with Eps and $minPts$.

TDJ's only difference to DJ-Clustering is an additional condition for neighborhoods: the points in the neighborhood set must lie within the temporal window defined by $deltaT$. This way, two clusters describing visits to *two* closely neighbored places are not merged to one big cluster (as in DJ-Clustering). TDJ's pseudo-code can be illustrated by adding the temporal constraint $deltaT$ on line 2 in Algorithm 1.

However, TDJ does not recognize repeated place visits with short duration of stay. For example, a person may drive through the same fast food restaurant a couple of times during a day. However, because the person only stays there for a short period of time, TDJ does not accumulate enough location readings to meet the $minPts$ constraint to form a cluster. To handle this situation, a relaxed temporal constraint strategy called R-TDJ was designed. This strategy considers points that satisfy the spatial constraint but fall in a much larger temporal window $rDeltaT$. The window is supposed to be large enough to capture a different visit to the same place. Using the above example, suppose there is not enough points in the morning visit to form a cluster. The R-TDJ strategy will relax the temporal constraint to count the evening visit points against $MinPts$. That is, the points from both the morning visit and the evening visit will be counted together against the $MinPts$ constraint, so that a morning cluster will be formed. The morning cluster will not be merged with the evening cluster.

Greater $rDeltaT$ values will lead to a more relaxed temporal constraint, thus find a larger number of less repetitive visits. A typical value could be set as 24 hours for daily repetitive events, or 168 hours for weekly ones.

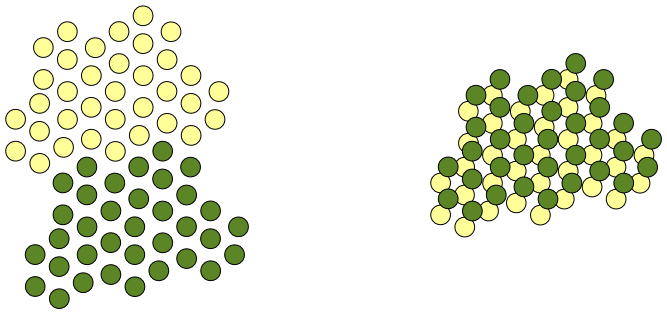
To improve the performance of the described density-based algorithms (DJ-Clustering, TDJ, R-TDJ), some temporal pre-processing is performed. First, GPS receivers return not just latitude and longitude, but also a speed, estimated by the distance traveled between consecutive readings. Zhou et al. exploited this information to eliminate GPS readings with speeds greater

than 0, in contrast to the already described approach by Ashbrook and Starner. This removes many GPS readings collected while driving, which are considered uninteresting. Second, a GPS reading was eliminated if it was within a small distance of the previous reading. This reduces the amount of data that needs to be processed by the algorithm, therefore speeding it up. Another reason these almost-stationary GPS readings were eliminated is that the authors had "an intuition that indoor places and outdoor places should be represented by similar sets of points" [Zhou et al., 2005b].

Evaluation. While designing an appropriate algorithm for Crumblr, a number of issues with TDJ and R-TDJ have been identified:

First, the aforementioned temporal pre-processing steps for TDJ remove all points with speeds greater than 0. This makes it impossible to consider the last valid point before entering a building and losing the GPS signal.

Second, the removal of a GPS reading that was within a small distance of the previous reading is disadvantageous when trying to capture shapes of places, especially if this (unspecified) distance threshold is too low. It is also not clear whether the filtering of close points has an effect on fulfilling the $MinPts$ constraint.



(a) Case A (b) Case B

Figure 6.7: Two cases revealing weaknesses in TDJ and R-TDJ

Finally, consider the scenarios depicted in Figure 6.7. The yellow and the green points are location readings, taken at two separate days. In Figure 6.7(a), Alice has visited a cafe C_1 on Monday; on Tuesday, Alice has visited cafe C_2 which is spatially very close to C_1. In Figure 6.7(b), Alice has visited the same cafe, once on Saturday and once on Sunday. Table 6.1 outlines the issues with TDJ and R-TDJ when applied to the depicted scenarios.

While Case B in Table 6.1 could be fixed by simply merging the two clusters, Case A would remain an issue.

	Case A	Case B
TDJ	Correctly recognizes two visits to two places.	Does not recognize two visits to the same place. Due to the hard temporal constraint $deltaT$, the two clusters are not merged.
R-TDJ	Due to the relaxed temporal constraint $rDeltaT$ and because both clusters are density-joinable, the points from the second visit count to the first visit, and vice versa. Two large clusters are formed – points from different places are not separated.	Even though the relaxed temporal strategy correctly recognizes visits to the same place, the clusters are not merged. This would leave the user with two clusters logically belonging to the same place.

Table 6.1: Issues with TDJ and R-TDJ

Moreover, R-TDJ's relaxed temporal strategy was designed to detect personal *paths* from GPS traces, consisting of place visits. However, the place visits in R-TDJ do not have to be "significant" in Crumblr's sense: even passing a place P that has been clustered beforehand would yield a new visit to this place, due to the relaxed temporal constraint. This is not desirable for Crumblr's purposes, as its aim is to capture longer, significant visits to places.

6.1.6 Crumblr's Approach

Based on the evaluation of related algorithms, Crumblr's approach builds on the TDJ algorithm, modifying and extending it in several ways. Specifically, Crumblr skips TDJ's pre-processing steps and treats GPS signal losses separately. Furthermore, it does merge clusters, similarly to the accumulative clustering approach from Kang et al. described in 6.1.3.

Definition 6.1.3 (Temporal predecessors) Let p and q be two points from a set of GPS readings S, with $\text{timestamp}(p) < \text{timestamp}(q)$. p is the temporal predecessor of q if

$$\nexists r \in S : \text{timestamp}(p) < \text{timestamp}(r) < \text{timestamp}(q)$$

Algorithm 2 Crumblr's Place Visit Recognition Algorithm

Require: $\forall p_i, p_j \in S : i < j \Rightarrow timestamp(p_i) < timestamp(p_j)$

for all points $p_1, p_2 \in S$, where p_1 temporal predecessor of p_2 **do**
  if time_difference$(p_1, p_2) \geq deltaT \wedge$
  distance$(p_1, p_2) < k * \text{speed}_{p_1} * \text{time_difference}(p_1, p_2)$ **then**
    Mark p_1 as a visit to an indoor place
  end if
end for

Execute TDJ with two modifications:
– skip temporal pre-processing
– *do* merge clusters by applying Kang et al.'s merging condition

for all recognized clusters C_i **do**
  for all two consecutive points p_1, p_2 from C_i **do**
    if time_difference$(p_1, p_2) \geq newVisitT$ **then**
      Store timestamp(p_1) as a time of visit for cluster C_i
    end if
  end for
end for

return Clusters C_i with a list of visit timestamps for each cluster

Definition 6.1.4 (Visits to indoor places) Let p_1 and p_2 be two temporal neighbors from a set of GPS readings S. p_1 describes a visit to an indoor place if the following conditions are true:

$$\text{time\_difference}(p_1, p_2) \geq deltaT \tag{6.2}$$

i.e. the signal must have been lost for at least $deltaT$ minutes, and

$$\text{distance}(p_1, p_2) < k * \text{speed}_{p_1} * \text{time\_difference}(p_1, p_2) \tag{6.3}$$

i.e. the user must not have traveled further than the estimated maximum distance based on the speed at p_1 and the elapsed time time_difference(p_1, p_2), multiplied with a constant $k \in (0, 1)$.

Crumblr sets the sampling interval $T_{sampling}$ to a high value, i.e. 1 minute, to reduce battery drain. The constant k in equation 6.3 can be set to any real value between 0 and 1. The Crumblr prototype uses $k = 0.3$. Figure 6.8 illustrates the intuition behind the heuristic for detecting indoor visits, outlined in Definition 6.1.4. The green points represent GPS readings p_i, with p_1

being the last reading obtained before entering a building. If speed_{p_1} is not explicitly available from the GPS module, it can be estimated by considering the reading before p_1. Let p_0 be the point that was acquired before p_1, i.e. its temporal predecessor. The speed at the point p_1 can be estimated with the quotient $\text{distance}(p_0, p_1)/\text{time\_difference}(p_0, p_1)$. According to the example,

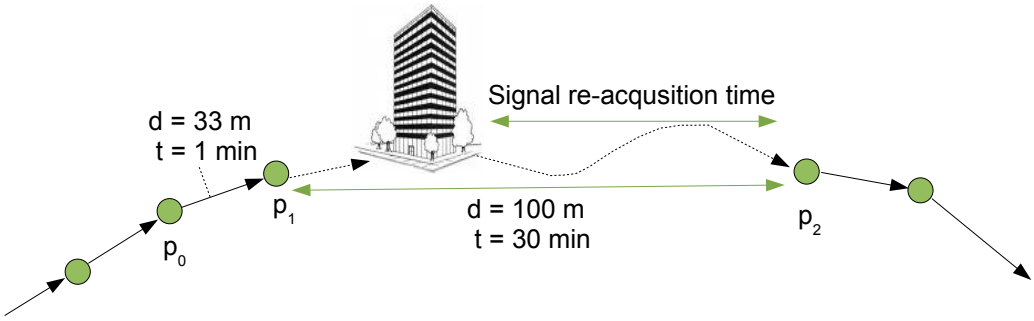


Figure 6.8: The intuition behind Definition 6.1.4

the user's estimated speed at point p_1 is 2 kilometers per hour. Since the reading p_2 was acquired 30 minutes after p_1 and $\text{distance}(p_1, p_2) = 100m$, then the conditions in Definition 6.1.4 are fulfilled: $\text{distance}(p_1, p_2) = 100m$ is less than $0.3 * 2kmh * 0.5h = 300m$. Basically, the latter value expresses how far the user could have gone assuming constant speed, multiplied by a factor k to account for a possible non-linear movement pattern before re-acquiring the signal (depicted by the curved line pointing from the building to p_2).

Algorithm 2 outlines the approach to detect place visits employed by Crumblr. In a first step (lines 1 – 5), the GPS trace is analyzed for loss of signal events. While the comMotion approach interpreted three signal loss events as a place visit, Crumblr proposes spatial and temporal constraints to eliminate urban canyons such as tall buildings or tunnels. The constraints are outlined in Definition 6.1.4. The equation 6.3 should make the approach independent of the time needed to re-acquire the GPS signal.

After detecting visits to indoor places, in a second step the GPS trace is analyzed for visits to outdoor places in form of clusters (line 6). For this purpose the TDJ algorithm is modified so that it *does* merge clusters that belong to the same place. The proposed merging condition is defined as follows: if the cluster's center is within $mergeT$ of an existing cluster's center, the two clusters are merged; otherwise it is added as a new place. The introduction of a merging threshold $mergeT$ is accounting for the issues outlined in Table 6.1 – by controlling the merge process, the algorithm is able to differ between

Case A and Case B. Crumblr sets $mergeT = Eps$.

Finally, in a third step, the clusters are analyzed for the number of different visits. The basic idea is to compare every pair of temporal neighbors in a cluster C_i and look for "jumps in time", indicating a new visit to this place (lines 7 – 13). The time difference between two consecutive points needs to be greater than the threshold $newVisitT$, which is set to 60 minutes. The simple assumption here is that if the user returns to a place after more than 60 minutes, Crumblr will interpret this as a new visit. This is important since the frequency of visits provides significant cues about a user's place preferences (as already discussed in section 4.1.1). An example of this procedure is illustrated in Table 6.2. The identified time jumps are presented as bold characters.

Timestamp	Latitude	Longitude
13:34:23 - 18 May 2008	7.4885340	34.89877456
13:35:23 - 18 May 2008	7.4885144	34.82654433
13:36:23 - 18 May 2008	7.4880054	34.89812466
...	...	...
19:12:15 - 18 May 2008	7.4880998	34.89833226
19:13:15 - 18 May 2008	7.4885364	34.89877666
10:30:23 - 20 May 2008	7.4885144	34.89841067
...	...	...

Table 6.2: An example of a set of points comprising a cluster. The cluster contains several time jumps since it is a result of multiple merge operations.

Results. To accurately evaluate Crumblr's proposed algorithm, a substantial amount of GPS trace data needs to be collected as well as ground-truth about the places people actually visited. The data collection task requires substantial effort as well as technical expertise to diagnose and fix any problems. Furthermore, the data is inherently sensitive, making it challenging to recruit volunteers. Therefore, due to resource limitations, some artificial test cases were constructed instead.

Typical scenarios were constructed with the Admin Interface (cf. section 5.2.5) and the data was fed into Crumblr's visit recognition algorithm. The complete description of the test scenarios and the obtained results are given in Appendix A. In summary, the first results are very promising: Crumblr filters out urban canyons, recognizes visits to indoor and outdoor places, detects multiple visits to the same place and demarcates closely neighbored places. However, a scenario not too far from reality was constructed that is

not interpreted correctly by Crumblr's visit recognition algorithm (cf. Figure A.3 in Appendix A). Nevertheless, the results still need to be quantified and generalized in a comprehensive user-based study, which can only be subject of future work (see also the final remarks in Chapter 8).

Computational Complexity. The computational complexity of Crumblr's visit detection algorithm can be analyzed in three steps, according to the outlined parts of the algorithm. Initially, as depicted in Algorithm 2, the set of GPS readings must be ordered by timestamps. Usually, this is already the case since the GPS readings are stored in the database in the order they were obtained. Therefore, each two consecutive points are temporal predecessors. Thus, detecting indoor visits can be done in $O(n)$.

Next, the R-TDJ algorithm is executed to compute the neighborhood of a point. This can be done in $O(n^2)$. Another major cost is the join computation for each point's neighborhood with existing clusters. This can be done in $O(n^2)$ as well.

Finally, extracting the visit timestamps for each cluster is done in $O(n)$ since the points are already ordered by timestamps.

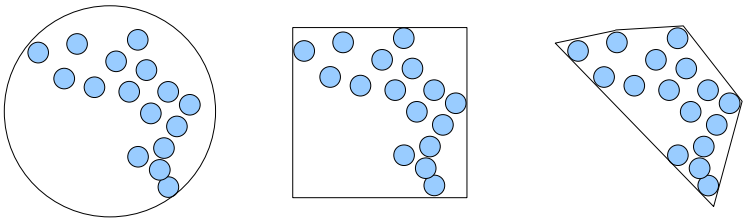
Overall, the complexity of the algorithm is $O(n^2)$. It can be further optimized by using a database that utilizes a spatial index, for example R-Trees [Guttman, 1984]. Performance tuning is out of the scope of this thesis, however. Moreover, since the algorithms are supposed to run on the Google Android platform, some remarks regarding performance tuning must be made. The Google Android platform provides a powerful notification architecture, enabling applications to react to system events such as "loading battery". The process running the visit recognition algorithm could be triggered by such an event. In this case, the overall complexity of $O(n^2)$ is acceptable for Crumblr's proof-of-concept purposes.

6.2 Place Shapes

When designing a location-aware system dealing with places, a fundamental question is:

> *How do people's understandings of* place *relate to the representations of physical locations used by computational systems?*

In [Zhou et al., 2005a], an empirical study was carried out to answer this question. The authors equipped 28 participants with GPS-enabled devices, which logged their position data for three weeks. Subsequently, the authors



(a) Circle (b) Rectangle (c) Convex hull

Figure 6.9: Approximating a set of points by geometric shapes

conducted semi-structured interviews, asking the subjects to map each visited place to one of the four shape categories – *Dot*, *Multi-Dots*, *Region*, *Path*. It was found that many places come in more complex shapes than points. In particular, the shapes *Region* and *Path* were common across subjects. Twenty-one percent of all places observed in the study were of these more complex shapes. In order to account for this result, Crumblr's place model employs a generic approach – every place is described either as a single anchor point or a region. Crumblr deals with *Paths* (i.e. routes) in a separate way, as already discussed.

When trying to represent places as regions instead of single points, several options are imaginable. The following considerations played a role when the Crumblr place model was designed. First, the input for the model is a set of clustered GPS points. Each cluster contains multiple points describing the place shape. The place model should therefore offer a suitable representation for such a set of points. Second, a trade-off must be chosen between a high amount of information (expressiveness) and computational/storage efficiency.

The idea to use simple geometric shapes such as circles and rectangles was considered inappropriate because of the loss of information due to coarse granularity. Precisely storing every point of each cluster would be too inefficient. A data structure achieving a good balance between the described criteria is the *convex hull*. This term is commonly used for the boundary of the minimal convex set containing a given non-empty finite set of points in the plane. Figure 6.9 illustrates and compares the described approaches to represent a set of points.

Furthermore, consider two places which are very close to each other, e.g. two downtown cafes. Figure 6.10 compares the capability of the three structures to demarcate the two places described by two sets of GPS readings (blue and orange). Obviously, the convex hull is better suited to describe

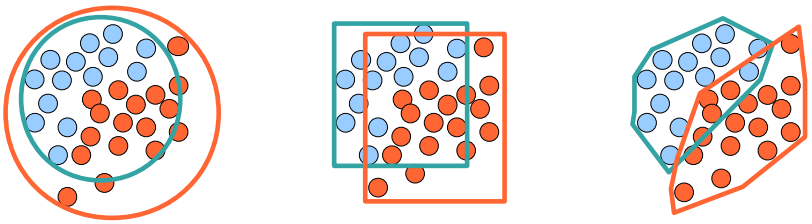


Figure 6.10: Place demarcation capability

complex place shapes and to differentiate between them in dense city areas. Simple geometric shapes such as circles or rectangles are too coarse-grained and too sensitive to outliers for this purpose. To improve the recognition of places a user has visited, such precise information about place shapes is very helpful. More details about Crumblr's method to associate places to visits will be given in section 6.3.

In computational geometry, numerous algorithms are proposed for computing the convex hull of a finite set of points, with various computational complexities. The list of algorithms includes "Jarvis march", "Graham scan", "Divide and conquer", and "Quick hull". Since the place shapes have to be managed and stored on the server, storage and computation efficiency is a significant factor when choosing a convex hull algorithm. However, since the algorithms are exchangeable, the ease of implementation was considered slightly more important for the proof-of-concept prototype.

6.2.1 Calculating Convex Hulls

Graham scan, published in 1972 by Ronald Graham [Graham, 1972], is a fairly sophisticated and very efficient algorithm having $O(n\log(n))$ complexity. Being also fairly easy to implement, it has been chosen for Crumblr's purposes. A brief outline of the algorithm is given below.

Let C be a list of points in a Cartesian coordinate system. First, the point with the lowest y-coordinate P is found. Next, C is sorted in ascending order of the angle they and the point P form with the x-axis. P itself is moved to the last position in C.

The algorithm proceeds by considering each point in the sorted list in sequence. For each point, it is checked whether moving from the two previously considered points to this point is a "left turn" or a "right turn". If it is a right turn, this means that the previous point is not part of the convex hull and should be removed from consideration. This process is continued for as long as the set of the last three points is a right turn. As soon as

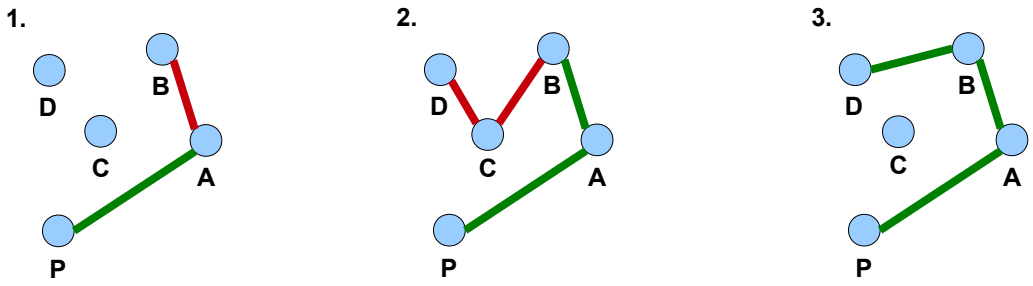


Figure 6.11: Graham scan

a left turn is encountered, the algorithm moves on to the next point in the sorted list. The process will eventually return to P, at which point the algorithm is completed and the list now contains the points on the convex hull in counterclockwise order.

Figure 6.11 illustrates the algorithm. The point P is the point with the lowest y-coordinate. As shown in "1.", A to B is a "left turn", so the algorithm proceeds to the next point in the array, C. In "2.", the turn from B to C is a left turn, as well. However, the next turn, from C to D is a right turn. The algorithm detects this and discards the previously chosen segments until the turn taken is left (B to D in this case).

6.2.2 Updating Place Shapes

As described in section 4.1.2, the place shapes are collaboratively created by Crumblr's users. Every uploaded place visit provides additional data about the actual place shape. After Crumblr's place visit detection algorithm has clustered the GPS data, Graham scan is performed to calculate the convex hull of each cluster. Next, for each convex hull, Crumblr estimates the place A this visit probably belongs to (a process described in section 6.3). After the user has confirmed the place, the place shape needs to be updated. The following discussion describes the design of the proposed method for updating place shapes based on convex hulls.

Simple methods

Let S_{old} be the set of points describing the old convex hull of the place A, and V the set of points describing the convex hull of a cluster representing a visit to A. The resulting new place shape S_{new} could be calculated as follows:

$$S_{new} = GrahamScan(S_{old} \cup V) \tag{6.4}$$

i.e. the union of S_{old} and V is taken as input for Graham scan, resulting in a new convex hull. Figure 6.12 illustrates this process.

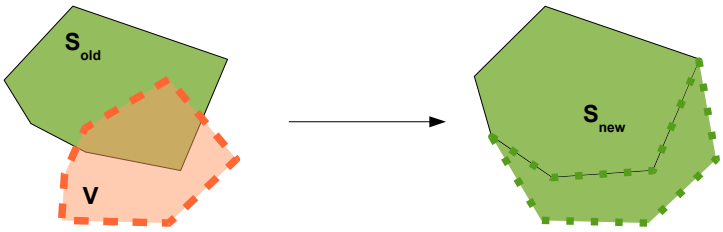


Figure 6.12: Old shape S_{old} and a new place visit V result in new place shape S_{new}

Another possible variant could be a simple union of the old shape with the convex hull of the cluster without performing Graham scan, i.e. $S_{new} = S_{old} \cup V$. The computation of S_{new} would be more complex, with basically no improvement in precision. Both introduced variants have one decisive drawback: the shape gets bigger and bigger over time, since every update can only enlarge the shape. However, Crumblr's stated goal is to design a converging method that stabilizes the shape over time, achieving a good balance between accuracy and efficiency.

Limiting the impact of single updates

As already mentioned in section 4.1.2, the effect of a single visit to the previously calculated shape is limited by employing a weighting method. The basic idea of the proposed weighting method for shape updates is the following: the more users already have contributed to the place shape, the less a new place visit should impact the shape update.

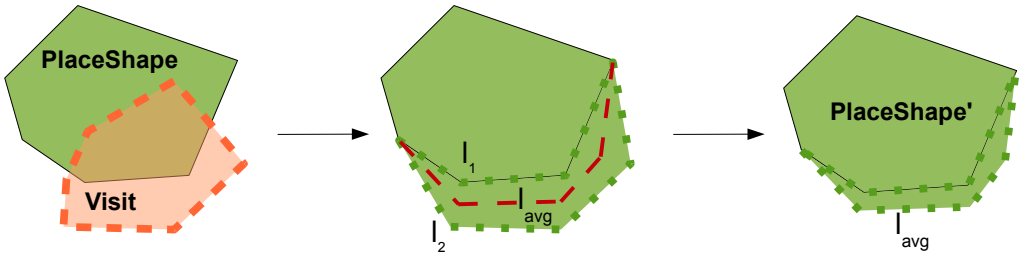


Figure 6.13: An approach to limit the impact of single updates

Consider a possible approach illustrated in Figure 6.13, where *PlaceShape* describes the convex hull of a given place shape and $Visit$ represents the convex hull of a place visit. In a first step, the two convex polygons are merged using Graham scan. Now consider the lines l_1 and l_2. l_1 is the line describing the old edge of *PlaceShape*, whereas l_2 resulted from performing Graham scan. To reduce the impact of $Visit$, the geometric average of l_1 and l_2 is computed and assembled into the resulting line l_{avg}. The resulting polygon is reduced to $PlaceShape'$ as illustrated in the last step.

The averaged polyline is determined as follows. Assume polyline l_1 contains more points than polyline l_2 (if this is not the case, reverse them) and let m be the number of points in l_1. The closest point in polyline l_2 to each point in l_1 is found. This produces m pairs of points, where points in l_2 can appear more than once, while points in l_1 appear only once. The geometric average of each of these pairs is computed and assembled into the resulting polyline. This polyline represents the average of l_1 and l_2 and has as much information as possible (since there are more points in l_1). Furthermore, when the geometric average of a pair of points is computed, the number of previous shape updates can be used as weight for the points of l_1. This way, the impact of single updates tends to decline over time.

This method still has the same disadvantage as the simpler variants – it only allows the place shape to grow. In the worst case, if the first visit to place A led to the shape *PlaceShape* being positioned very far away from the true shape, there is no way to ever correct the shape by transforming or moving it to the "true" shape. Therefore, a more flexible mechanism is needed to allow the shape to shrink in some directions as well.

ShapeUpdate – a flexible approach

Now consider the steps in Figure 6.14. The roles of *PlaceShape* and $Visit$ are now reversed. Again, in the first step, the two convex polygons are merged using Graham scan. Next, the lines l_1 and l_2 are chosen as illustrated in Figure 6.14. l_2 now represents the new edge added to $Visit$ after performing Graham scan, while l_1 is the old edge of $Visit$. After computing the geometric average l_{avg}, the resulting polygon is reduced according to l_{avg} to obtain $Visit'$, as depicted in the last step.

$PlaceShape'$ from Figure 6.13 represents the growth of the place shape, whereas $Visit'$ expresses how the place shape should shrink. In the final step, $PlaceShape'$ and $Visit'$ are combined. Figure 6.15 illustrates the approach taken here: by overlaying the two shapes we obtain an intersection region, representing the final $PlaceShape_{new}$.

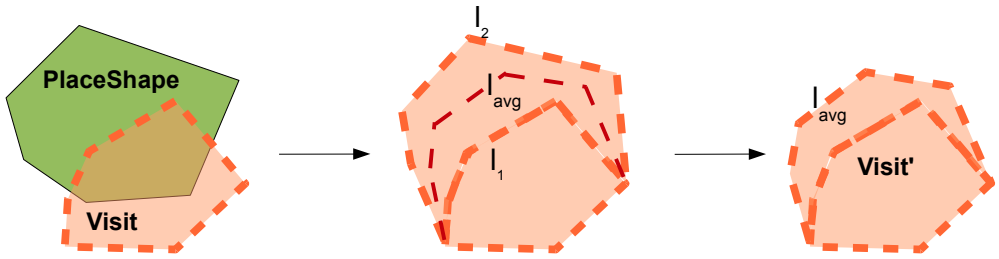


Figure 6.14: Reversing the roles of *PlaceShape* and $Visit$

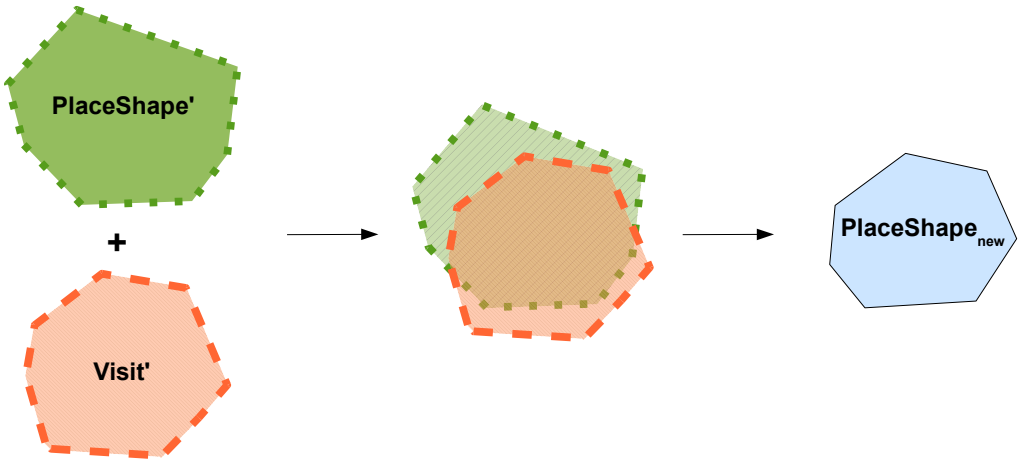


Figure 6.15: Combining $PlaceShape'$ and $Visit'$ to obtain the final shape $PlaceShape_{new}$

Figure 6.16 summarizes this method, called *ShapeUpdate*, by plotting the initial shapes, *PlaceShape* and $Visit$, with the outcome of the described method, $PlaceShape_{new}$. Intuitively, the resulting shape represents a weighted average of the two input shapes. The weights in the given example are equal, i.e. *PlaceShape* has been obtained from a single previous visit. Appendix B contains more complex examples, demonstrating the effectiveness of *ShapeUpdate*.

Loss-of-Signal Visits

Crumblr's place visit detection algorithm deals with loss-of-signal events separately, as already described. Due to the long sampling interval $T_{\text{sampling}} = 1$ min employed by Crumblr, loss of information occurs when trying to capture visits to indoor places (i.e. buildings). Consider the following worst case: if

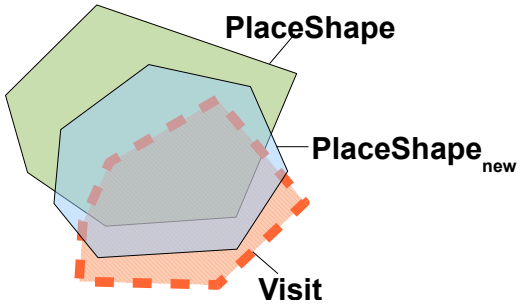


Figure 6.16: Final result of the *ShapeUpdate* method

the user enters a building right before the next GPS sample is requested from the GPS module, the previously obtained GPS coordinate p could be too far away from the entered building to be used directly as an estimate for the location of the building. Consequently, it should not directly be taken into consideration when updating the place shape of the building, making it a good candidate for the *ShapeUpdate* method as well. By applying *ShapeUpdate* on single points describing loss-of-signal events which are spread around the building, the resulting polygon should be 'drawn' towards the building's real location.

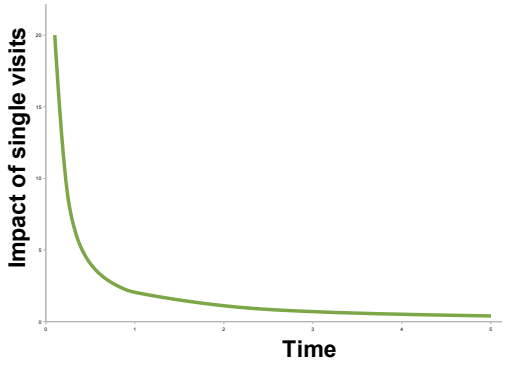


Figure 6.17: The expected impact of single updates over time

Results. As already mentioned, the process of updating the place shapes should converge towards a stable state. Because of the nature of convex hulls and the way updates are considered when calculating the new shape, the impact of updates to a shape will decline over time and converge to zero (Figure 6.17). Additionally, a stop threshold for the changes could be set, which would prevent any further insignificant updates to the shape. This

threshold, however, conflicts with the issue of timeliness, which is discussed in section 7.2. The proposed *ShapeUpdate* method can be implemented efficiently as its building blocks are based on efficient algorithms, while keeping a high level of informative content about place shapes.

The *ShapeUpdate* method can have a rather adverse effect: it has a lasting tendency to increase the number of points that define *PlaceShape*. However, a simplification step could be performed after each update, which would eliminate points that are almost collinear to its two neighbored points.

6.3 Associating Visits to Places

For each recognized place visit, Crumblr estimates the place this visit probably belongs to. A high accuracy relieves the user from having to select another place for a recognized visit. Crumblr employs a custom distance function to determine the likeliness that a visit belongs to a certain place.

As already discussed, a place visit is either described as a single point (loss of GPS signal) or a convex hull (clustered points). Let $Visit$ be such a set of points with cardinality $|Visit| \geq 1$. Place shapes defining a region are an optional part of the place model, as already mentioned. Therefore, a place can either be described by a single point or by a convex hull as well. Let $PlacePoints$ be the set of points describing a place P, with $|PlacePoints| \geq 1$. To determine the likeliness that a visit described by $Visit$ belongs to the place P described by $PlacePoints$, the distance between the two sets of points needs to be calculated. Intuitively, the likeliness of a place should be inversely proportional to the calculated distance to the visit region.

If at least one of the point sets consists of a single point, the distance calculation is straightforward. A simple method is to calculate the shortest distance between this single point and all points in the other set.

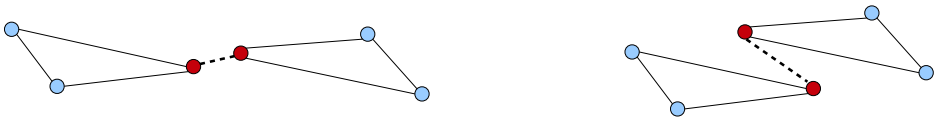
6.3.1 Shortest Distance

When talking about distances between two polygons A and B[1], a common method is to calculate the shortest distance between any point of A and any point of B. This *minimin* function can be formally defined as

$$D(A, B) = \min_{a \in A} \min_{b \in B} d(a, b) \tag{6.5}$$

[1]The following discussion also applies to the case when A or B consist of only two points.

where $d(a, b)$ is any metric between two points. This definition of distance between polygons is unsatisfactory for Crumblr, however. Consider the two polygons in Figure 6.18(a), for example. The triangles are close to each other considering their shortest distance, shown by their red vertices. However, when talking about polygons representing place shapes and visit regions, a small distance between two polygons should mean that no point of one polygon is far from the other polygon. In this sense, the two polygons shown in Figure 6.18(a) are not very close, as their furthest points are rather far away from each other. In summary, the shortest distance is independent of polygonal shapes. Another example is given in Figure 6.18(b), showing the



(a) Not considering the whole shape (b) Not considering the positions

Figure 6.18: Deficits of shortest distance between polygons

same two triangles, but in different positions. Obviously, the shortest distance concept carries low informative content, as the distance value *increases* when the polygons are moved towards each other.

6.3.2 The Hausdorff Distance

In spite of its apparent complexity, the *Hausdorff* distance captures the previously described subtleties which are ignored by the shortest distance. Named after Felix Hausdorff (1868–1942), Hausdorff distance is the maximum distance of a set to the nearest point in the other set [Rote, 1991]. More formally, Hausdorff distance from set A to set B is a *maximin* function, defined in equation 6.6.

$$h(A, B) = \max_{a \in A} \min_{b \in B} d(a, b) \tag{6.6}$$

It should be noted that Hausdorff distance is asymmetric, which means that in most cases $h(A, B)$ is not equal to $h(B, A)$. This general condition also holds for the example of Figure 6.19, as $h(A, B) = d(a1, b1)$, while $h(B, A) = d(b2, a1)$. This asymmetry is a property of maximin functions, while minimin functions are symmetric.

A more general definition of Hausdorff distance is denoted as

$$H(A, B) = \max\{h(A, B), h(B, A)\} \tag{6.7}$$

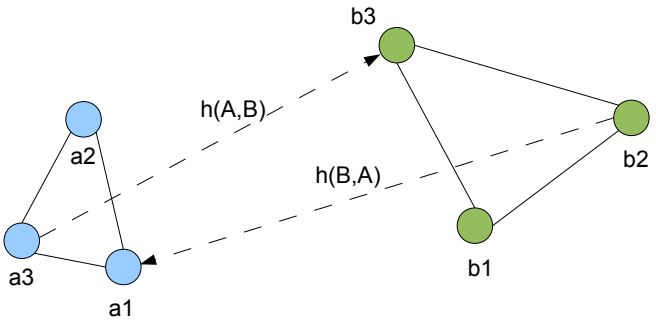


Figure 6.19: Hausdorff distance on point sets

which defines the Hausdorff distance *between* A and B, while equation 6.6 applies to Hausdorff distance *from* A to B (also called directed Hausdorff distance).

6.3.3 Crumblr's Approach

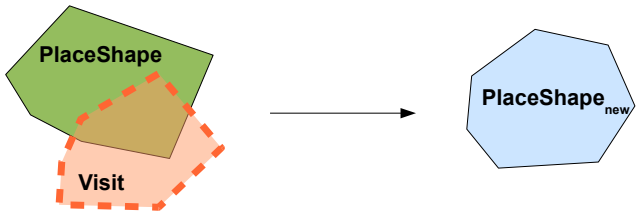
Initially, Crumblr utilized the Hausdorff distance to estimate the place likeliness. However, the Hausdorff distance does not provide meaningful results if the polygons intersect each other or if one is contained in the other. Special considerations are necessary to detect and deal with these cases. To tackle this problem, Crumblr proposes a custom approach to likeliness estimation. The basic idea is to look for the place shape S which would change the least after updating it with *ShapeUpdate*.

In general, the likeliness for a place P and visit $Visit$ can be calculated as follows:

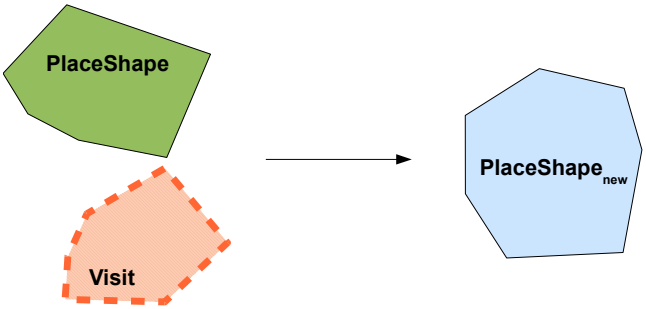
$$Likeliness_{P,V} = \frac{Surface(PlaceShape)}{Surface(ShapeUpdate*(Visit,\ PlaceShape))} \tag{6.8}$$

where *Surface* is the surface area of a polygon, and *ShapeUpdate** differs from the described *ShapeUpdate* method by not applying the weighting method. In other words, likeliness depends on how much $Visit$ would modify the place shape *PlaceShape* by applying the *ShapeUpdate** method. Crumblr exploits domain knowledge here: If $Visit$ is completely contained in the polygon $PlacePoints$, i.e. if $ShapeUpdate*(Visit, PlaceShape) = PlaceShape$, then the place P is considered an excellent candidate and the likeliness in equation 6.8 gets the maximum value 1. The Hausdorff distance method for estimating the likeliness, on the other side, would return a value less than 1, since the Hausdorff distance would be greater than 0, in this case. If the

two polygons intersect each other or if $PlacePoints$ is contained in $Visit$, the proposed approach provides an elegant solution by considering the calculated impact of a possible shape update.



(a) High likeliness



(b) Low likeliness

Figure 6.20: Examples for Crumblr's approach to estimating place likeliness

Results. Figure 6.20(a) shows an example for high estimated likeliness. The surface of the resulting convex polygon would be only slightly larger than the surface of the place shape. Figure 6.20(b) depicts a case where a visit region having no intersections with the place region results in a low likeliness because a shape update would significantly enlarge $PlaceShape$.

The proposed approach offers the following advantages over distance-based approaches. First, the proposed likeliness calculation accounts for the intuitive long-term tendency that the impact of single visits to existing shapes should be minimized. Second, the method is very efficient since calculating the surface area of a convex polygon and performing the $ShapeUpdate*$ algorithm can be implemented very efficiently.

6.4 Activity Estimation

As described in section 4.1.1, Crumblr estimates the performed activity and suggests it to the user by examining aggregated history data. The aggre-

gated history data comprises a set of place visits. A place visit is modeled in Crumblr as follows:

$$\text{PlaceVisit} = (\text{place, user, setOfActivities})$$

where setOfActivities is a set of all activities that were performed by the user at that place combined with the count for each activity. For example, a place visit history might look like outlined in Table 6.4:

Place	**User**	**Activities**
BarRosso	Alice	{(Cafes, 4)}
BarRosso	Bob	{(Mexican Food, 3), (Cafes,2)}
BarRosso	Charlie	{(Cafes, 7)}
Kung Fu Wok	Bob	{Chinese Food, 4)}
...	...	...

Table 6.3: An example of a place visit history

After executing its place visit detection algorithm and recognizing that the user U has visited the place P, the following two approaches are imaginable for estimating the performed activity based on history data:

1. For all place visits to P, sum up the activity counts for each activity and pick the one with the highest count.

2. For all place visits to P *made by* U, pick the activity with the highest count.

For example, when estimating the activity for the place BarRosso and user Bob, the first approach returns "Cafes" and the second "Mexican Food".

Crumblr first tries to apply the second approach – if there were no previous visits from user U to the place P, Crumblr falls back to the first approach. The basic idea here is that, in some cases, different people tend to perform different activities at a place. By favoring the second approach, the activity is estimated in a personalized way. If the first approach also returns no data, the user has to pick the activity from the complete activity taxonomy[2], however.

[2]This approach is favored over a list of ranked recommendations in order not to confuse the user with an unfamiliar ordering of activities.

6.5 Aggregating Routes

This section is organized in three parts. First, it outlines the work in [Morris et al., 2004] that serves as a foundation for Crumblr's route aggregation approach. Second, improvements to Morris et al.'s work are suggested. Third, the contextual layer briefly outlined in section 4.1.2 is explained in detail.

6.5.1 Graph Reductions by Morris et al.

One of the most fundamental problems in forming a database of routes is that due to errors inherent in the GPS system itself, two GPS tracks taken from the same trail will not exactly be the same. A second problem arises from the fact that routes overlap and intersect with other user-submitted routes. Given a set of possibly overlapping and intersecting GPS tracks, the general problem is to form a network of routes. The input to the problem is a set of GPS tracks S (cf. Figure 6.21(a)), where each GPS track is a polygonal line. The desired output is a planar graph G with vertices V and edges E, where each vertex represents a route junction and each edge represents a route segment as a polygonal line (cf. Figure 6.21(b)). The output graph G has the following properties [Morris et al., 2004]:

- No duplicate representations of any physical route.
- Where duplicates exist, the resulting edge is the geometric average of all duplicates present in the input.
- A vertex exists only where an actual route junction exists.

The first step is to build an initial graph representing all intersections among the members of S. The intersections represent vertices of the graph, splitting the tracks into separate polylines at all intersection points. Each section of a split polyline corresponds to an edge in the graph.

The task of graph reduction is to find portions of the graph that are close enough to be considered the same trail. Each operation performed reduces the graph in some way, bringing it closer to the desired solution. The following are the reductions proposed in [Morris et al., 2004].

Parallel Reduction

A parallel reduction takes two *parallel edges* in the graph and reduces them to a single edge. Edges are parallel in the graph if they connect the same two vertices. Parallel edges are replaced by a single edge if the polylines associated with the two edges are sufficiently similar. Similarity is determined

(a) Route of intersecting and overlapping GPS tracks

(b) Completely reduced graph

Figure 6.21: Example of the employed route aggregation method

by computing the Hausdorff distance, $H(A, B)$, between the two polylines A and B (see also section 6.3.2). If $H(A, B)$ is below a threshold value, *rThresh*, the reduction is performed. The averaged polyline to replace the parallel edges is determined as already described in section 6.2.2.

If *rThresh* is chosen to be larger than the GPS error present in the data, pairs of polylines collected by traversing a single physical trail will reduce to a single trail. Since the GPS error is almost always insignificant compared to the distance between unique trails, the value for *rThresh* can be chosen appropriately. Typical values of *rThresh* range from 20 to 60 meters (depending on the quality of the data) as physical trails are almost always (in the Hausdorff distance) much further apart [Morris et al., 2004].

(a) Two overlapping jogging routes in a park

(b) Parallely reduced graph

Figure 6.22: Example of the employed route aggregation method

Figure 6.22(b) gives an example of a parallel reduction and the resulting averaged polyline.

Serial Reduction

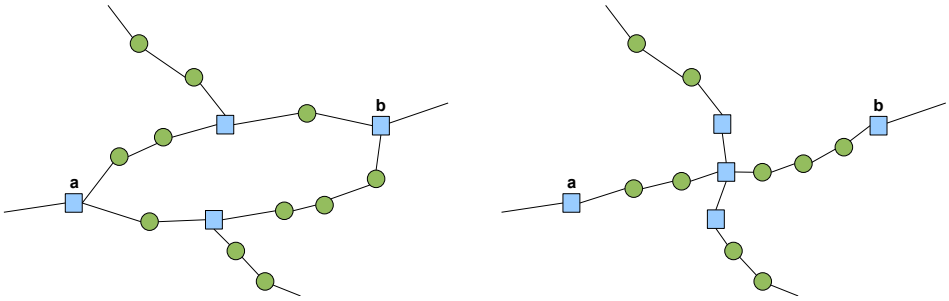
A serial reduction eliminates a vertex of degree two (having only two outgoing edges). Vertices of degree two can be created as a result of applying other graph reductions. If a vertex in the graph has only two edges it cannot be a trail intersection – therefore, it can be safely deleted. An example is shown in Figure 6.22(b): after multiple parallel reductions several nodes of degree two are created, depicted by blue rectangles. They can be safely removed from the graph by merging the connected edges.

Face Reduction

Faces of a graph are regions bounded by some edges of the graph, as shown in Figure 6.23(a). As GPS tracks intersect each other while traveling on the

same trail, many small faces are formed due to GPS errors, as depicted in Figure 6.21(a).

A face in the graph is reduced if all of its components are sufficiently close. A measure of closeness is determined by first finding the two vertices of the face that are the furthest away from each other. Let these vertices be a and b. Two polylines are then formed with a and b as the common endpoints by concatenating the polylines corresponding to the edges of the face. The result is a pair of parallel edges. These two polylines are evaluated for similarity using the same Hausdorff distance method as in parallel reductions (cf. section 6.5.1). The same threshold value, $rThresh$, is also utilized. Figure 6.23(a) shows a face of degree four, with the vertices of the graph shown as blue boxes. The points describing the polylines (each edge of the graph has a corresponding polyline) are shown as green circles. Vertices a and b are the pair of vertices that are furthest apart; they are used to form two polylines to determine if the face should be reduced.



(a) A face of degree 4 (b) Reduced face

Figure 6.23: Example of face reduction

When a face is reduced, the average polyline R is computed in the same manner as described in parallel reductions. Although a single polyline is produced, all data from the face is being incorporated in it. All the edges from the face (and their corresponding polylines) are deleted from the graph and R is inserted connecting a and b. Let d be the degree of the face being reduced, i.e. the face contains d vertices. Although a and b are connected to R, the other $d-2$ vertices are not. The remaining $d-2$ vertices are connected to R by determining the point on R that is closest to each of the vertices individually. R is split appropriately at each point found to be closest to one of the $d-2$ vertices. New vertices are inserted at each split point with two edges corresponding to split polyline segments from R and an edge connecting the split point to the corresponding vertex in the $d-2$ set. The polylines connecting the new vertices to $d-2$ vertices consist of only

two points: those of the two vertices they are connecting. Any of the $d-2$ vertices may have a split point in common and in this case the split vertex has edges to any $d-2$ vertices that share it. The result of this process is illustrated in Figure 6.23(b).

Face reductions usually also produce vertices of degree two, as in Figure 6.23(b). This makes them a suitable input to the aforementioned serial reductions. Note that a parallel reduction is simply a face reduction of degree two. They have been detailed separately to ease description; they are also treated differently when analyzing the overall graph for possible reductions.

Face detection. Morris et al. do not discuss methods for detecting faces. A large number of uploaded GPS tracks tends to create many faces in the resulting graph. Therefore, an efficient method for detecting faces is needed. Furthermore, not every detected face will be reduced, i.e. not every face's components are sufficiently close.

Classic algorithms for detecting cycles (or more specifically: faces) use depth-first search (DFS) to solve this problem [Cormen et al., 2001]. However, the output of such algorithms is usually a complete enumeration of all faces in a given graph $G = (V, E)$. Crumblr employs a custom iterative breadth-first search (BFS) based approach for simultaneously detecting faces and processing them. The approach can be expressed as follows. Let v be any vertex in V. By performing BFS with the node v as starting point, a cycle c is eventually detected, if G is cyclic. Due to the nature of BFS, c has to be a face. The face detection triggers a separate method for checking if the detected face should be reduced (i.e. if its components are sufficiently close). If the check was positive, the face is reduced immediately and the graph G is modified. Subsequently, control is returned to the face detection algorithm, which restarts BFS at a vertex $z \neq v$. The algorithm terminates after all faces have been either reduced or ignored.

6.5.2 Suggested Improvements

This thesis builds on the described graph reductions and suggests some improvements and extensions, which are outlined below.

Weighting Method for Sequential Operation

Although Morris et al. originally proposed the graph reductions as a one time aggregation of GPS tracks, in [Morris et al., 2004] it has been noted that the described operations can be run repeatedly with new data. However, Morris et al. did not explicitly deal with the problem of a single track's

(a) Edges fulfilling the relaxed constraints

(b) After applying parallel reduction

Figure 6.24: Parallel reduction with relaxed constraints

impact on the network. Crumblr suggests a weighting method to account for the "maturity" of the established route network. For example, when a polyline $line_{new}$ describing a new GPS track needs to be averaged into the network by performing a parallel reduction of $line_{new}$ and a polyline from the network $line_{network}$, then $line_{network}$ should be weighted proportionally to the number of tracks that contributed to it. The same weighting method is employed by the *ShapeUpdate* method (cf. section 6.2.2) in order to reduce the impact of single visits to place shapes – the method's possible issues with timeliness are discussed in Chapter 7.

Faces emerge "near" the intersections that result from integrating new GPS tracks into the graph G. By focusing the aforementioned BFS-based face detection process on vertices that represent the calculated intersections, the face detection process performs significantly better.

Parallel Reduction with Relaxed Constraints

Parallel reductions in [Morris et al., 2004] are applied to edges that connect the same vertices, i.e. having two vertices and sharing them both. However, Crumblr proposes to relax this constraint and to apply parallel reduction to edges both having one vertex and sharing the single vertex. Figure 6.24

illustrates the proposed extension to parallel reduction.

Broken Junction Reduction

Consider the case depicted on the left hand side in Figure 6.25. Let the distance between vertices a and b be below *rThresh*. The graph reductions described so far are unable to further reduce this constellation of vertices and edges. However, the edges to the left and right of the vertices a and b

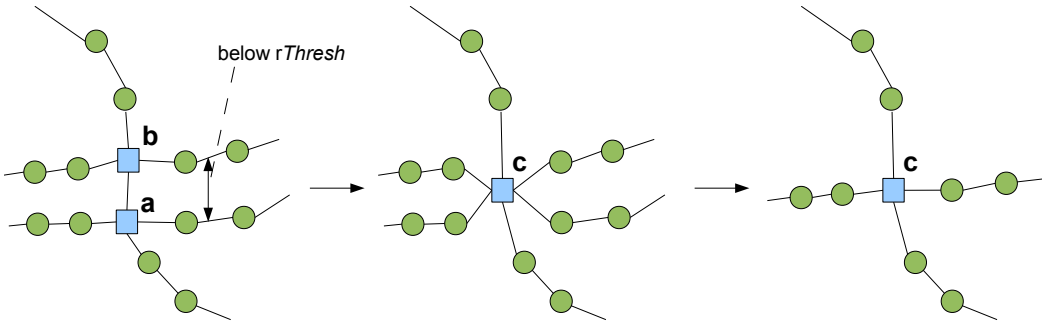


Figure 6.25: Broken junction reduction

could be reduced in a parallel manner since they are close enough. To reduce this "broken junction", the following reduction method is proposed: if two vertices a and b are directly connected and their distance is below *rThresh*, a new vertex c is created, inheriting all edges from a and b (middle part of Figure 6.25). This enables the global graph reduction procedure to continue simplifying the graph structure by finally applying parallel reduction, as illustrated on the right hand side in Figure 6.25.

Evaluation

Morris et al. have proposed an effective solution to the problem of forming a database of routes. However, they provided a solution to the spatial part of the problem only, while the semantic part, i.e. the integration of conflicting thematic data, has remained unexplored. In [Matyas, 2007], an approach has been presented that uses Semantic Web technologies – formal ontologies – to describe spatial and semantic aggregation methods. More specifically, the authors employ a rule-based language that allows to express spatial and semantic preconditions for the application of aggregation methods. Such a rule could be: "If there is a river between two uploaded GPS tracks, they should *not* be reduced by applying Morris et al.'s parallel reduction". If the data is checked positively against all defined rules then the specified graph

reduction method is to be applied. Due to the lack of detailed information about the landscape (such as rivers, lakes, or bridges), such approaches are currently out of the scope of Crumblr and can only be part of future work.

6.5.3 Enriching the Network with Contextual Data

In order to enable personalized route recommendations, some additional data is attached to each segment in the network. When a user uploads a track, he also specifies the performed activity (e.g. "Jogging"). Crumblr attaches this contextual data to the track, i.e. the user's identity and the performed activity, before aggregating it into the network. Each route segment is related to the set of tracks that contributed to it, where each track is associated to the performed activity and the user who uploaded the track.

When performing graph reductions, the newly created route segments inherit the contextual data from the segments they have been created from. For example, when applying parallel reduction to segments p_1 and p_2, the data about users and activities is transferred to the resulting segment p_{avg} before p_1 and p_2 are finally removed from the network.

6.6 Recommending Places and Routes

In order to provide an intelligent mechanism to filter out the excess of available information and to provide users with the prospect to find out items that they will probably like according to their logged history of prior behavior, recommendation systems have been extensively adopted by both research and e-commerce applications. At the core of recommendation systems are prediction algorithms which aim to calculate the probability that a user will "like" an item. Recommendation algorithms are classified into *content-based* and *collaborative filtering* based, while hybrid techniques have been proposed as well. The coverage of related approaches that have also been applied to recommendation systems, such as Bayesian networks, clustering, and Horting, is out of the scope of this thesis; [Schafer et al., 1999] provides a detailed taxonomy and examples of recommendation systems. A brief overview of the most common recommendation techniques is given below, before discussing Crumblr's approach in detail.

Content-Based Filtering

The basic idea of content-based filtering is to express the content of each item in a form that can be objectively evaluated, and filter out items whose

content does not match the user's preferences. The most commonly used method for expressing content is the *feature vector* method. According to this method, each item is described in the form of a vector v_{data} consisting of values for a set of features. For example, in the case of text data, features are defined as the frequency with which several keywords appear in the text. The preferences of each user are also expressed as a vector v_{user} using the same set of features. If v_{data} is "similar" to v_{user}, the probability that the user will like the item is considered high. Thus the item is recommended to the user. Most content-based systems are intended only for recommending text-based items, since appropriately expressing the content for other types of data such as places is difficult to automatize.

User-Based Collaborative Filtering

Unlike traditional content-based filtering systems, such as those developed using information retrieval or artificial intelligence technology, filtering decisions in collaborative filtering are based on human and not machine analysis of content. Collaborative filtering recommends items that were given high *ratings* by a number of users with similar preferences as the user who requested the recommendation [Goldberg et al., 1992]. Ratings capture preferences of a user to a specific item or item category. They are collected by either prompting the user by the system's interface for an explicit rating (e.g. a number from 1 to 10) or by implicitly deriving a rating from observed user actions (e.g. online buying behavior).

The biggest advantage of collaborative filtering over content-based filtering is that collaborative filtering requires no previous knowledge about the content of the data, and thus can be applied to any type of data, regardless of content. It does not depend on machine analysis of content and it has the ability to implicitly filter based on complex and hard to represent concepts, such as taste and quality.

Item-Based Collaborative Filtering

There is a variation of collaborative filtering called item-based collaborative filtering [Sarwar et al., 2001]. In this approach, instead of calculating similarities between users, similarities between items are calculated. Items which show high similarity with the items that the user has given high ratings to are recommended. The main idea here is to analyze the user-item representation matrix to identify relations between different items and then to use these relations to compute the prediction score for a given user-item pair. The intuition behind this approach is that a user would be interested in items

similar to the items the user liked earlier and would tend to avoid items similar to the items the user did not like earlier. Amazon.com employs a variant of item-based collaborative filtering, for example.

In [Sarwar et al., 2001], item-based collaborative filtering has been experimentally evaluated on a large movie database consisting of several thousands of users and movie ratings. Experimental results showed that these techniques tend to produce much faster recommendations, since they do not require to identify the neighborhood of similar users when a recommendation is requested. The item-based scheme has been shown to provide better quality of predictions than the classic user-user scheme; however, the improvement was found to be not significantly large.

Similarity

There is a number of different ways to compute the similarity between users or items. Among the most common methods are cosine-based similarity and statistics-based similarity (Pearson's correlation coefficient, Jaccard's similarity coefficient). Mathematically, cosine vector similarity sim_{cos} of two vectors x, y is defined as the dot product of these vectors divided by the product of their magnitudes:

$$sim_{cos}(x, y) = cos(\theta) = \frac{x \cdot y}{\|x\|\|y\|} = \frac{\sum_{k}^{n} x_k y_k}{\sqrt{\sum_{k}^{n} x_k^2 \sum_{k}^{n} y_k^2}} \tag{6.9}$$

A comprehensive discussion on similarity measures is out of the scope of this thesis. Among other similarity measures, cosine similarity has some desirable properties which make it a well suited candidate for Crumblr's purposes. First, it is insensitive to the length of the vectors – it only considers the structural similarity, i.e. the angle, between the vectors. Second, it produces values in $[0, 1]$ that can directly be interpreted as the degree of similarity, with 0 being the lowest degree and 1 the highest one.

Explanations

It seems reasonable to provide explanation facilities for recommendation systems such as collaborative filtering systems. Previous work with another type of decision aid – expert systems – has shown that explanations can provide considerable benefit [Miller and Larson, 1992]. A detailed background about the benefits of explanations cannot be given here. Some of the

benefits provided are justification, education, user involvement, and acceptance [Herlocker et al., 2000]. User acceptance is of special importance for applications relying on user contributions, as it determines the adoption rate, the effectiveness, and the success of such applications. It is my personal belief that user-based recommendations are easier to explain than item-based recommendations, because the former utilize a more trustworthy, human-centered basis for explanations – other users. Moreover, going beyond pure item recommendations, user-based recommendations also foster interactions between users. Todays most successful platforms like Amazon.com and Digg[3] effectively utilize such approaches. For platforms like Crumblr, user-based recommendations and explanations are expected to significantly increase the system's acceptance.

Analysis

Filtering approach	Performance	Quality	Explainability
Content-based	++	−	++
Item-based collaborative	+	+	−
User-based collaborative	−	+	+

Table 6.4: An analysis of recommendation approaches applied to spatial entities such as places and routes

Table 6.4 outlines a qualitative comparison of the described approaches applied to the domain of places and routes. Although content-based filtering alone does not seem sufficient for such concepts like places and routes, it certainly makes sense to filter content according to some simple features such as the geographic location. Since Crumblr is a proof-of-concept prototype, differences in performance are considered less important than the explainability of the approach. Therefore, advantage is given to user-based collaborative filtering over item-based approaches. Several components of Crumblr's recommendation model employ collaborative filtering to recommend places and routes to users.

Crumblr's Recommendation Process

As already stated in section 4.1.3, Crumblr's recommendation process consists of two steps:

[3] www.digg.com

1. Content-based filtering of items (places or route segments) according to the user's current location and the activity to be performed. The suitability of an item I for a given activity act_P is determined by examining the history of visits to this item; the item I is considered suitable for act_P if there has been a sufficient number n of visits to I with activity act_P. Currently, for simplification purposes, n is set to 1.

2. Rating the filtered items according to several components of the utilized recommendation model.

The rest of this section deals with the second step, i.e. the components of Crumblr's recommendation model for places and routes, respectively.

6.6.1 Place Recommendations in Crumblr

This section provides a detailed coverage of the recommendation model for places, conceptually introduced in section 4.1.3.

The following scenario provides an example that will be used throughout the section. Consider the place visits given in Table 6.5 and imagine

		Places						
		Venezia	**Glockencafe**	**Extrablatt**	**Eins A**	**BarRosso**	**21 Lounge**	**St. Martin**
Users	**Alice**	3	3	1	0	0	0	0
	Bobby	0	2	13	10	2	1	1
	Charlie	14	3	1	20	3	3	0
	Jenny	2	0	10	2	3	16	2
	Steve	12	6	2	0	2	1	15

Table 6.5: An example for place visits in Kaiserslautern. The cell values express the number of visits for the given user and place, considering the activity "Cafes" only.

the following scenario: Alice is new in town, has already visited a few cafes (Venezia, Glockencafe, Extrablatt) and wants to try out some new places. As already mentioned, Crumblr first filters the places according to the user's current location and the activity to be performed ("Cafes"). Second, the

places are rated according to several components of the utilized place recommendation model (cf. section 4.1.3), which will be described in detail next.

General Similarity

Crumblr utilizes a service offered by the Captchr system called *UserSimilarityService*. Captchr hereby defines user similarity as a function of two user's long-term profiles. In Captchr a long-term user profile is modeled as a quadruple of the user's activities, absolute spatial behavior, relative spatial behavior, and temporal behavior. A more formal discussion of the concept of user profiles can be found in [Käppler, 2008] and shall therefore not be discussed here in detail. Instead, the components of the user profile are informally explained below:

- User's activities are represented as vectors in n-dimensional activity space[4] with elements representing the frequency of occurrence for the given activity and user.
- Absolute spatial behavior refers to what places a user has been to, including the total number of visits and the user's explicit ratings for each of these places.
- Relative spatial behavior yields information about the distances a user moves, with respect to the user's home place.
- Temporal behavior is derived by analyzing how often a user has been active in a certain time slot of a day. Currently, four different time slots are distinguished: morning, afternoon, evening, and night.

A formal similarity model for computing social neighborhoods has been proposed in [Käppler, 2008] and implemented in the Captchr prototype. Each component is modeled as a feature vector, making it suitable for traditional approaches to measuring similarities such as the cosine similarity or the Jaccard's similarity coefficient. In a nutshell, the similarity between two users A and B is calculated in Captchr by summing up the weighted partial similarities over the components of these profiles. The total user similarity therefore has a value in $[0, 1]$.

Considering the example from Table 6.5 again, let $PlaceVisitors = \{\text{Bobby, Charlie, Jenny, Steve}\}$ denote the set of all users who visited the places $AllPlaces = \{$Venezia, Glockencafe, Extrablatt, Eins A, BarRosso, 21

[4]Currently there are 41 activities, transient activities not counted, thus $n = 41$.

Lounge, St. Martin}. Let us further assume that Captchr has calculated the similarities between Alice and the users from *PlaceVisitors* as given in Table 6.6. Denote this vector of "general user similarities" as sim(Alice).

	Bobby	**Charlie**	**Jenny**	**Steve**
General similarity to Alice	0.89	0.68	0.25	0.27

Table 6.6: User similarities between Alice and *PlaceVisitors*, as calculated by Captchr

Generally, Crumblr minimizes the rating for places that have previously been visited by Alice, providing diversity and fostering serendipitous recommendations. Therefore, let us only consider the places that Alice has not visited yet[5], *NotVisited*. For each place from *NotVisited*, the "generally similar users" component of the recommendation model tries to answer the question:

Is this place mostly visited by users like me?

It does so by calculating the match between Alice's profile and the weighted "average" user for each place. Examine Table 6.7: for each place P, the columns represent the place profile vector v_P, whose components express each user's share of the total place visit count. For example, 83% of total

	Eins A	**BarRosso**	**21 Lounge**	**St. Martin**
Bobby	31%	20%	5%	6%
Charlie	63%	30%	14%	0%
Jenny	6%	30%	76%	11%
Steve	0%	20%	5%	83%

Table 6.7: Users' share of the total visit count for each place

registered visits to St. Martin were made by Steve, 11% by Jenny, and so on. By multiplying each user's visit share with the user's general similarity to Alice and summing up the products, Crumblr calculates the final rating $GenSim_P(Alice)$ for a place P and user Alice:

$$GenSim_P(Alice) = sim(\text{Alice})^T * v_P$$

For example, the general similarity rating for St. Martin is calculated as follows:

$$[0.89, 0.68, 0.25, 0.27]^T * [0.06, 0, 0.11, 0.83] \approx 0.31$$

[5] At least according to the knowledge available to Crumblr at that time.

Finally, the general similarity ratings for all places in *NotVisited* are given in Table 6.8.

	Eins A	**BarRosso**	**21 Lounge**	**St. Martin**
$\text{GenSim}_P(\text{Alice})$	0.72	0.51	0.35	0.31

Table 6.8: Alice's calculated general similarity ratings for places in *NotVisited*

In summary, the general similarity rating of a place P tells Alice how she matches with the estimated average visitor of P. A rating of 0.72 for the cafe Eins A can be interpreted as very good. Ratings are not normalized since Crumblr wants to reflect the true score it has calculated, expressing how Alice's general profile compares with the average visitor.

Activity Similarity

Alice might perceive the "general" similarity as too general – she might think that "Cafes" preferences alone matter more to her than the overall observable user behavior. Two persons that are generally similar in terms of Captchr's general similarity concept might have different tastes regarding "Cafes". To account for this, Crumblr provides the "activity similarity" component of its recommendation model. This approach is based on user-based collaborative filtering.

Like many traditional user-based collaborative filtering approaches, Crumblr first calculates the similarity between two users by employing cosine vector similarity. It does so by considering only visits to "Cafes". A user's visit vector for "Cafes", $\text{visit}_{\text{Cafes}}$, is a vector in a p-dimensional space, where p is the number of *AllPlaces* and each component expresses the visit frequency for the corresponding place. For example, based on Table 6.5, Steve's visit vector for "Cafes" is $[12, 6, 2, 0, 2, 1, 15]$. To calculate the activity similarity $sim_{\text{Cafes}}(\text{Alice, U})$ between Alice and any other user U, their visit vectors for "Cafes" are used as input for cosine similarity (cf. equation 6.9).

Table 6.9 summarizes the computed activity similarities between Alice and the users in *PlaceVisitors*. The row of this table can be interpreted as a vector $sim^{\text{Alice}}_{\text{Cafes}}$. Apparently, Charlie and Steve have tastes very similar to Alice regarding "Cafes". Crumblr therefore considers Charlie and Steve as a good source of information when rating places in *NotVisited*. In this case, it tries to answer the following question:

	Bobby	Charlie	Jenny	Steve
"Cafes" similarity to Alice	0.33	0.82	0.36	0.95

Table 6.9: Activity similarities between Alice and *PlaceVisitors*, as calculated by Crumblr

> *What are the other favorite "Cafes" of users that usually prefer the same "Cafes" as I do?*

Having calculated the activity similarities between Alice and the other users, the next step in collaborative filtering is to pick out several users with the highest similarity values (*nearest neighbors*). With similarity values of 0.82 and 0.95, Charlie and Steve should intuitively be included in the set of nearest neighbors, *NN*. Bobby and Jenny, on the other hand, are considered not sufficiently similar and are removed from consideration.

Finally, classic collaborative filtering tries to predict the rating that Alice is going to provide for an item (place). The prediction is based on the nearest neighbors' ratings concerning the place and their similarity with the active user. Since Crumblr does not collect explicit ratings[6], the ratings must be implicitly derived from the observed user behavior captured in Table 6.5. Considering a user U, the more often U has visited a place P compared to other places, the higher the rating for P should be. An intuitive approach is to pick the U's most often visited place P_{max} and normalize the other places' visit count by P_{max}'s visit count. This leads to a rating interval $[0, 1]$, with 1 being the highest rating. Table 6.10 summarizes the derived ratings for Charlie and Steve, $r_{Charlie,P_i}$ and r_{Steve,P_i} respectively.

	Eins A	BarRosso	21 Lounge	St. Martin
$r_{Charlie,P_i}$	1	0.15	0.15	0
r_{Steve,P_i}	0	0.13	0.07	1

Table 6.10: Users' derived place ratings for places in *NonVisited*

Crumblr predicts the rating p_{Alice,P_i} Alice is going to give to a place P_i by calculating the weighted sum of the nearest neighbors' ratings and their

[6] considering the collection process obtrusive and the explicit ratings unreliable (see also section 4.1.1)

activity similarity to Alice:

$$p_{Alice,P_i} = \frac{\sum_{H \in NN} (sim^{\text{Alice}}_{\text{Cafes}}(H) * r_{H,P_i})}{\sum_{H \in NN} sim^{\text{Alice}}_{\text{Cafes}}(H)}$$

This calculus leads to the predicted ratings p_{Alice,P_i} summarized in Table 6.11.

	Eins A	**BarRosso**	**21 Lounge**	**St. Martin**
p_{Alice,P_i}	0.46	0.14	0.10	0.54

Table 6.11: Alice's predicted ratings for places in *NonVisited*

Comparing these results with the ratings from the "general similarity" model, the place Eins A has a rather high rating in both models. Therefore, Alice's intention to visit Eins A is likely to get enforced. The BarRosso cafe, on the other hand, has received a low predicted rating (0.14) with respect to "activity similarity". Being aware of the way both models predict ratings[7], Alice could reconsider the relatively high value obtained from the "general similarity" model attributed to BarRosso (0.51).

Absolute Popularity

Alice might be merely wondering which the generally most popular cafes in the city are, because she is new in the area and simply likes crowds, for example. Moreover, she might be looking for a rather less turbulent place where she can talk to her old friend. For cases like this, Crumblr offers the "absolute popularity" component of its recommendation model. The term "absolute" refers to the neutrality to the user profiles attached to the place visits.

Let *count* be the vector whose components express the total visit counts for every place in *NotVisited*. The absolute place popularity rating $popularity_P$ for a place P is directly proportional to its visit count, normalized by the maximum visit count:

$$popularity_P = \frac{count_P}{\max_{k \in NotVisited} \{count_k\}}$$

The visit count is normalized by the maximum visit count, since Crumblr makes no claim of having complete knowledge about the real-world visits.

[7]More on explanations will be given in section 6.6.3.

Therefore, the normalized ratings enable Alice to quickly identify the places that are usually more often visited than others among Crumblr's users. According to the user-place matrix given in Table 6.5, the absolute place popularity ratings are summarized in Table 6.12.

	Eins A	**BarRosso**	**21 Lounge**	**St. Martin**
$popularity_P$	1	0.31	0.66	0.56

Table 6.12: Absolute popularity ratings for places in *NonVisited*

Among the four cafes in *NonVisited*, Eins A seems to be the most popular cafe. BarRosso, on the other hand, is less visited by Crumblr's users. Note that "absolute place popularity" is not a personalized rating, as it does not consider user profiles. In a nutshell, by considering only visits to *NonVisited*, this method orders the places in *NonVisited* on a $[0, 1]$ scale by comparing them with the most popular place.

Overall Analysis

Considering the ratings obtained from the three discussed recommendation models, Alice gets the overall impression that BarRosso is not a good candidate for her. Eins A and St. Martin, on the other hand, are two cafes she considers worth visiting.

6.6.2 Route Recommendations in Crumblr

This section provides a detailed coverage of the recommendation model for routes, conceptually introduced in section 4.1.3.

The following scenario will serve as an example for the discussion of the recommendation approaches: Alice has just moved to a new neighborhood and wants to have some new "Jogging" routes recommended.

Activity Similarity

Consider another user in Crumblr, Bobby. Bobby and Alice do not know each other, but they usually took the same jogging routes in Alice's old neighborhood. Figure 6.26 illustrates the jogging route graph from Alice's old neighborhood. The dashed blue lines represent routes Bobby has taken, whereas Alice's routes are displayed as green lines. The red rectangles represent vertices in the graph, while the dashed lines represent the edges, i.e.

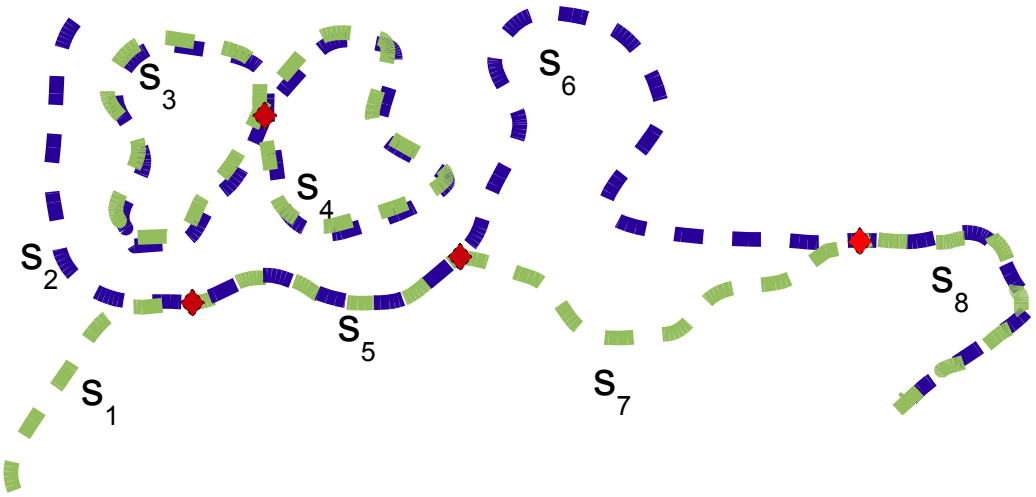


Figure 6.26: Alice's (green) and Bob's (blue) jogging routes in Alice's old neighborhood

route segments. The route segments s_3, s_4, s_5, and s_8 are shared by both users[8].

Table 6.13 summarizes the visit counts for the given route graph. The rows represent visit count vectors $v^k_{Jogging}$ for user k and activity "Jogging".

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
Alice	7	0	8	8	7	0	7	7
Bobby	0	15	17	17	15	15	0	15

Table 6.13: Route visits for the given graph example

Crumblr tries to benefit from this intuitive similarity between Alice and Bobby. It recommends meaningful route segments to Alice by using a variant of collaborative filtering. In the first step, Crumblr calculates the similarity between users. The basic idea is identical to the "activity similarity" for places, introduced in section 6.6.1. Equally, the "activity similarity" between two users is calculated as the cosine vector similarity (equation 6.9) between their visit count vectors v^k_a with respect to a given activity a. Continuing the

[8]Crumblr recognizes shared segments due to the route aggregation methods already described in section 6.5.

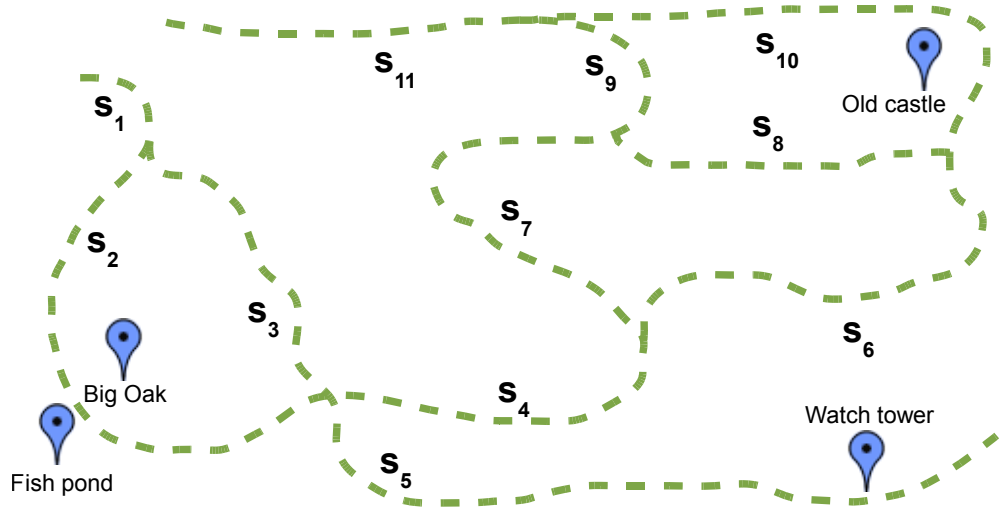


Figure 6.27: Route network and places in the vicinity of Alice's new home

example, the "Jogging" similarity between Alice and Bobby is

$$sim_{\mathrm{J}}(\text{Alice,Bob}) = \frac{v^{Alice}_{\text{Jogging}} \cdot v^{Bobby}_{\text{Jogging}}}{\|v^{Alice}_{\text{Jogging}}\| \|v^{Bobby}_{\text{Jogging}}\|} \approx 0.70$$

Figure 6.27 depicts the route network aggregated from uploaded tracks in Alice's *new* neighborhood, including some places around (blue markers) as well. Bobby is also very active in Alice's new neighborhood and jogs there quite often. Alice is not aware of Bobby, but getting routes recommended from him would very likely suit Alice well, as already stated. Since Crumblr has estimated a high similarity between Alice and Bobby with respect to "Jogging" (0.70), Bobby's favorite jogging routes are considered highly interesting to Alice. Table 6.14 describes the visits to route segments in the graph depicted in Figure 6.27, for the activity "Jogging". Let us assume that

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}
Bobby	12	0	12	8	4	7	1	6	4	2	8
Charlie	15	8	7	11	1	1	2	9	7	5	11
Steve	18	14	4	17	1	4	14	8	10	4	18

Table 6.14: User-segment matrix for the route network in Alice's new vicinity, for the activity "Jogging"

Charlie and Alice have no shared route segments at all – Charlie's activity

similarity to Alice is effectively 0.0. Steve has been added to the example as someone who shares several routes with Alice. A graph of Steve's and Alice's shared routes similar to the one depicted in Figure 6.26 is left out – Steve's resulting "Jogging" similarity to Alice has the value 0.23, however.

In the next step of classic collaborative filtering, several users with the highest similarity values (nearest neighbors) are picked out. Charlie's data is removed from consideration to account for the zero similarity to Alice. Crumblr picks Bobby and Steve as the nearest neighbors *NN*.

Finally, classic collaborative filtering tries to predict the rating that Alice is going to provide for an item. Similar to places, Crumblr does not collect explicit ratings for route segments. The ratings must therefore be implicitly derived from the observed user behavior captured in Table 6.14. Considering a user U, the more often U has visited a segment compared to other segments, the higher its rating should be. The approach taken for places is reused for route segments: pick the U's most often visited segment s_{max} and normalize the other segments' visit count by s_{max}'s visit count. This leads to a rating interval $[0, 1]$, with 1 being the highest rating. Table 6.15 summarizes the derived ratings for Bobby and Steve, r_{Bobby,s_i} and r_{Steve,s_i} respectively.

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}
r_{Bobby,s_i}	1	0	1	0.67	0.33	0.58	0.08	0.50	0.33	0.17	0.75
r_{Steve,s_i}	1	0.78	0.22	0.94	0.06	0.22	0.78	0.44	0.56	0.22	1

Table 6.15: Derived implicit ratings

Crumblr predicts the rating p_{Alice,s_i} Alice is going to give to a segment s_i by calculating the weighted sum of the other users' ratings and their similarity to Alice:

$$p_{Alice,s_i} = \frac{\sum_{H \in NN} (sim_{\mathrm{J}}(\mathrm{Alice}, H) * r_{H,s_i})}{\sum_{H \in NN} sim_{\mathrm{J}}(\mathrm{Alice}, H)}$$

This calculation leads to the predicted ratings p_{Alice,s_i} summarized in Table 6.16.

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}
p_{Alice,s_i}	1	0.19	0.81	0.74	0.26	0.49	0.26	0.49	0.39	0.18	0.81

Table 6.16: Predicted segment ratings for Alice

For example, segment s_3 received a much higher rating than s_2. Obviously, s_3 is a segment regularly taken by Bobby, whereas s_2 is Steve's preferred segment. Consequently, Alice's high similarity with Bobby leads to $p_{Alice,s_3} \gg p_{Alice,s_2}$. On the other hand, segment s_7 is avoided by Bobby and favored by Steve, implying a low predicted rating p_{Alice,s_7}. In summary, "activity similarity" effectively rates Bobby's favorite segments higher than Steve's.

Distance to Relevant Places

In addition to the targeted "Jogging" activity, Alice would like to have some "Sightseeing" spots along the routes. Crumblr therefore provides the "distance to relevant places" component, which takes into account how far the relevant places are when rating route segments.

Let act_P be the specified activity describing the desired places, e.g. "Sightseeing". Further, let $Places_{act}$ be the set of places satisfying the following conditions:

- The places are suitable for the activity act_P, with the suitability already defined in section 6.6.
- The places are within the region of interest as specified by Alice (cf. section 4.2.3).

For each segment s_i and for each place $P \in Places_{act}$, Crumblr first calculates the distance from s_i to P. Formally, the distance from a place P to a route segment s_i is the shortest distance between P and s_i:

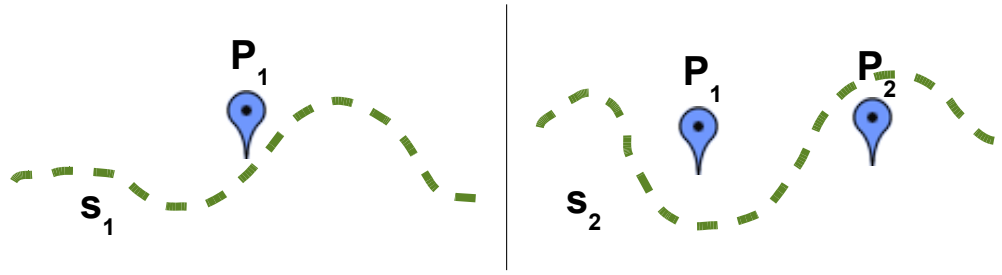
$$dist(s_i, P) = \min_{a \in s_i} \min_{b \in P} d(a, b)$$

Further, let *NearbyPlaces* denote the set obtained by removing all places from $Places_{act}$ being too far away from all s_i. Formally, these are places having $dist(s_i, P) > dThresh$ for all segments s_i, where $dThresh = 100$ meters.

Having calculated the distances to all places in *NearbyPlaces*, Crumblr derives a final rating $rating_{dist}(s_i)$ for segment s_i. Consider Figure 6.28(a), where segment s_1 has one "Sightseeing" place nearby, P_1. Let the distance from s_1 to P_1 be $dist(s_i, P) = 2$ meters. Intuitively, one would expect the rating to correlate negatively with $dist(s_i, P)$, for example:

$$\text{rating}_{dist}(s_i, P) = \frac{c}{c + dist(s_i, P)} \tag{6.10}$$

where c is a constant which aims at reducing the effect of very small distances. For demonstrative purposes, Crumblr sets $c = 20$. According to equation 6.10, the final rating for s_1 would be $rating_{dist}(s_1) = 0.9$.



(a) One nearby place (b) Two nearby places

Figure 6.28: Examples of route segments and *NearbyPlaces*

However, this approach is appropriate only if $|NearbyPlaces| = 1$, i.e. if there is only one place to consider. For example, Figure 6.28(b) illustrates the case where this approach might deliver unsatisfactory results. Let $dist(s_2, P_1) = 30\text{m}$ and $dist(s_2, P_2) = 35\text{m}$. Even though both places are more than 2 meters away from s_2, the fact that there are two relevant places nearby instead of one should imply a rating that is higher than the one obtained for the case in Figure 6.28(a). A simple average of the individual distance ratings for P_1 and P_2 would lead to a lower rating, whereas the sum of the individual distance ratings would still result in a too low rating:

$$rating'_{dist}(s_2) = \sum_{k=1}^{\eta} \text{rating}_{dist}(s_i, P_k) = \frac{20}{20+30} + \frac{20}{20+35} = 0.76$$

where η is the number of places in *NearbyPlaces*.

Intuitively, the rating should account for both the number of nearby places *NearbyPlaces* and the distance to these places. To account for this, the following equation can be used:

$$\text{rating}''_{dist}(s_i) = \ln(e + \eta - 1) * \text{rating}'_{dist}(s_i) \tag{6.11}$$

The term $\ln(e + \eta - 1)$ is a factor reflecting the significance of the size of *NearbyPlaces*. In the example from Figure 6.28(b), the resulting rating would be:

$$\text{rating}''_{dist}(s_2) = \ln(e + 2 - 1) * 0.76 \approx 1.3 * 0.7 = 0.99$$

and the rating for s_1 from Figure 6.28(a):

$$\text{rating}''_{dist}(s_1) = \ln(e + 1 - 1) * 0.9 = 0.9$$

yielding the desirable result when comparing both ratings:

$$\text{rating}''_{dist}(s_2) > \text{rating}''_{dist}(s_1)$$

However, since the results of both equation 6.10 and 6.11 can be greater than 1, Crumblr simply cuts off this excess: a rating greater than 1 can be simply interpreted as "good enough", or "there is plenty of relevant places nearby". Therefore, the final equation for calculating the rating of a segment s_i is:

$$\text{rating}^*_{dist}(s_i) = \max\{\text{rating}''_{dist}(s_i), 1\} \tag{6.12}$$

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}
$\text{rating}^*_{dist}(s_i)$	0	1	0	0	0.8	0.1	0	0.5	0	0.7	0

Table 6.17: Distance ratings for the route network in Alice's neighborhood

Table 6.17 summarizes the calculated distance ratings for the route network in Alice's neighborhood from Figure 6.27, with the blue markers representing the set of places $Places_{act}$. Segments with a zero rating have no "Sightseeing" places within distance *dThresh*. Segment s_2 has the maximum rating since it has two relevant places in its immediate vicinity. Due to the aforementioned $\ln(e + \eta - 1)$ factor, s_2's distance rating is greater than s_5's, even though "Watch tower" is very close to s_5.

A possible improvement to the calculated distance rating might be the consideration of the calculated relevance ratings for the nearby places. The place ratings might be used as multiplicative factors when aggregating multiple place distances in equation 6.12.

Dedication

A jogging route might be limited in its dedication to "Jogging", because there are other users who frequently use that route segment for other activities such as "Biking" and "Hiking". Alice might want to avoid bikers and hikers while jogging and therefore prefer other routes.

To account for this, Crumblr rates the route segments depicted in Figure 6.27 by calculating the percentage of "Jogging" visits for each segment:

$$\text{r}_{Jogging}(s_i) = \frac{count_{Jogging}(s_i)}{count(s_i)}$$

where $count_{Jogging}(s_i)$ is the number of visits to segment s_i for the targeted activity "Jogging", and $count(s_i)$ is the total visit count for s_i.

The calculated "dedication" ratings might look like in Table 6.18. Compared with the result of the previously introduced ratings ("activity similarity" and "distance to places"), there are some significant differences in

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}
$\mathrm{r}_{Jogg.}(s_i)$	1	0.78	0.21	0.60	0.18	1	0.10	0.1	0.45	0.1	0.91

Table 6.18: Dedication ratings for the route network in Alice's neighborhood

ratings. For example, s_2 has a very high "dedication" rating (0.78) since there are mostly joggers active on this segment. On the other hand, the same segment is not preferred by her most similar user, Bobby (0.19). Segment s_5, being very close to a sightseeing place (a watch tower), has a good "distance" rating – however, it is also expected to be highly frequented by people exercising mostly activities other than "Jogging".

Absolute Popularity

Similar to the case outlined in section 6.6.1, Alice might be interested in identifying generally popular route segments (segments visited more often than others), without caring about user profiles.

Let *count* be the vector of total visit counts for every segment in the given graph. The absolute popularity rating pop_i for a segment s_i is directly proportional to its visit count, normalized by the maximum visit count:

$$pop_i = \frac{count_i}{\max\limits_{p \in E} \{count_p\}} \tag{6.13}$$

where E is the set of edges in the graph $G = (V, E)$. The visit count is normalized by the maximum visit count, since Crumblr makes no claim of having complete knowledge about the real-world visits. Therefore, the normalized ratings enable Alice to quickly spot popular route segments in her vicinity.

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}
pop_i	1	0.49	0.51	0.8	0.13	0.27	0.38	0.51	0.47	0.24	0.84

Table 6.19: Popularity ratings for the route network in Alice's neighborhood

Table 6.19 summarizes the popularity ratings for the given graph, calculated by applying equation 6.13. Route segment s_5, for example, is less frequented than segment s_4.

Overall Analysis

The introduced four components of Crumblr's recommendation model should allow Alice to gain valuable insight about the jogging routes in her vicinity.

She can choose the most appropriate criterion in the given situation, before deciding what route(s) to take. In just a few clicks, Alice can access various kinds of information about the nearby routes. Furthermore, Crumblr employs visual and textual explanation techniques for the provided recommendation models, as discussed in the following section.

6.6.3 Explanations

Data presentation and exploration on mobile devices are heavily affected by the small size and resolution of displays. Several approaches have been explored to express items' relevance on mobile devices, such as a vertical bar attached to the symbol whose height represents how much that object satisfies the user's query, and symbols with different opacity levels [Reichenbacher, 2004].

Crumblr provides several mechanisms for explaining the calculated ratings on the mobile device. As already described in section 4.1.4, the opacity of each item (route segment or place) is adjusted to the item's rating from the currently active component of the recommendation model. The components of the model are mapped to visualization layers, which can be individually selected from a menu. The approach of varying opacity is feasible since all components of the recommendation model yield values in the interval $[0, 1]$, leading to an opacity calculation that is easily understandable by users:

$$\text{opacity}(item) = \text{rating}(item) * \text{opacity}_{max} \tag{6.14}$$

This way, the ratings in $[0, 1]$ are presented to the user in an intuitive way: items that are more opaque than others are eye-catching and appear more relevant. A minimum threshold is set to assure that all filtered items remain visible.

In addition to the opacity-based approach, specific on-demand visual and textual explanations have been developed for each component of the recommendation model, which are discussed next.

Explaining Place Recommendations

For each component, its most important aspects and the derived explanation approach are outlined. The discussion assumes a user Alice requesting explanations for a recommended place P.

General similarity: The most important aspects here are the set of users with a high share of visits to P, and the users' general similarity to Alice.

Therefore, for a place P, the names of the users with the highest share of visits to P are displayed as an explanation for the recommended place. To account for the differences between shares of visits, the font size of the names is scaled accordingly. To express the estimated general similarity to Alice, the opacity of the user names is calculated similar to the formula given in equation 6.14.

Activity similarity: The most important aspects are the top N users most similar to Alice with respect to the given activity A and their favorite places for the given activity A. For the place P, the names of the selected users which have contributed to the place's rating are shown. The size and opacity of their names is adjusted to their derived implicit rating for P and the calculated activity similarity to Alice. Alice is also given the option to review the favorite places of the selected N users.

Absolute popularity: P's visit count, the visit count of the most visited place P_{max} for the given activity A, and P_{max}'s name are displayed.

Explaining Route Recommendations

For each component of the route recommendation model, its most important aspects and the derived explanation approach are outlined.

Activity similarity: The most important aspects are the top N users most similar to Alice with respect to the given activity A and their favorite route segments for the given activity A. Therefore, the names of the selected N most similar users are displayed, whereby the font opacity is scaled according to the activity similarity to Alice. Furthermore, Alice is given the option to select a user U and review U's preferred route segments in the region.

This approach differs slightly from the explanation approach taken for explaining "activity similarity" for places. It is not reasonable to let users select individual route segments from a dense route network on the small screen when requesting explanations; instead, the user is given the option to view other users' favorite segments on the map.

Distance to places: Since a segment's distance to relevant places and the number of nearby relevant places are the most important aspects here, the relevant places are displayed as place markers as well.

Dedication: Each segment's percentage of visits with the targeted activity A can be explained with the segment's varying opacity.

Absolute popularity: A segment's visit count compared to the visit count of the most visited segment is directly transformed to the opacity level. Therefore, the varying opacity is considered sufficient in order to spot popularity trends in the displayed network of route segments.

The employed explanation approaches are derived from the most important aspects of the corresponding recommendation model. In particular, by showing the names of the users that contributed the most to the calculated ratings and further providing detailed explanations, the system's trustworthiness and serendipitous interactions between users are further promoted. A visual demonstration of the employed explanation approaches can be found in section 4.2.

Chapter 7

Discussion

Having described the employed algorithms in detail, this chapter provides some critical remarks and discusses general ideas about possible future work.

7.1 Cold Start and Personal Use

In section 2.6, the two dimensions of motivations for sharing place information have been discussed. Although Crumblr aims at covering all of the identified four categories of use, the "personal reminiscence" category remains not fully explored. Currently, users are only able to review past visits and activities before uploading them to the server; after that, this data cannot be reviewed any more. By offering users a more complete personal memory function, i.e. a way to recall to mind *all* past visited places and experiences, the system's usefulness in its 'blank state' and the users' motivation to capture place visits via Crumblr should be significantly higher.

Another way of reducing the problem of the initial lack of observed behavior might be a varying automation of acquiring preferences. For example, the user could be prompted to enter an initial profile of interests, which might be further updated by the system.

7.2 Temporal Aspects

The developed recommendation models for places and routes do not consider the time dimension when building user profiles and utilizing them for item recommendations. For example, the absolute popularity rating of a place (cf. section 6.6.1) does not account for the distribution of visits over the week, month, or year: the visit frequency of sidewalk cafes can be significantly different on work days compared to weekends. Moreover, a mecha-

nism is needed for keeping the data up-to-date: place visits older than several months or years should be considered less relevant than place visits within the last few days. It is also imaginable that the geographic shapes of routes and places change over time. By introducing a stop threshold for the *Shape-Update* method described in section 6.2.2, the place shapes would eventually stop updating and future changes to place shapes could not be considered.

Furthermore, Crumblr currently uses historic context only for updating the user profile. By extrapolating future user context, the system can provide additional functionality to the user based on the anticipated user location. For example, CityVoyager models users' movements using a first-order Markov model (i.e. a Markov model in which the probability of the next state is dependent only upon the current state), using geographic areas as states [Yuichiro and Masanori, 2006]. Transition probabilities are calculated from periodically plotted user locations, and a higher probability indicates more chances of a user advancing to the area. The places picked out by the filtering algorithm are weighted according to the areas in which they are located, with large weight values being added to places in the same area as the user, or in areas with large transition probabilities. Ashbrook and Starner also use first-order Markov models [Ashbrook, 2002]. They imagine early-reminder applications and solutions to the common problem of scheduling a meeting for several people as potential use cases for anticipating where users will go to and when they will do so.

7.3 Location Sensing Technologies

Section 2.4 outlined the common location sensing technologies available for today's mobile devices. Crumblr's visit recognition algorithm is highly tailored to the GPS technology; therein lies it main weakness as well. By incorporating other technologies for estimating a device's location, such as Bluetooth or WiFi localization approaches based on trilateration, the accuracy of the approach could be improved. Especially in cases where the GPS signal reception is very weak or not available at all, such complementary technologies might provide a valuable additional source of location information.

7.4 Data Management and Reliability

Crumblr's place visit recognition approach does not allow the user to manually specify visits that were not recognized by the system (false negatives).

Furthermore, the system does not allow users to remove places or route segments. For example, when a restaurant goes out of business, it makes no sense to further consider it for recommendation. Single contributions (place visits and GPS tracks) currently also cannot be removed from the aggregated database state. The system should be protected against single malicious users trying to disrupt the whole database. Wikipedia, for example, stores the whole editing history for each wiki page, allowing roll-backs to previous states carried out by other users. Further research is required in order to effectively translate such approaches to systems dealing with spatial data, such as Crumblr. For example, route segments with a high number of (unique user) submissions can most likely be relied upon. A user ranking system could also be employed, so that submitters of erroneous data can be penalized.

7.5 External Services and Data Interoperability

Crumblr currently only uses its own internal repository for storing and retrieving content. The possibilities of incorporating external content could provide additional benefit to the user. For example, by considering data about opening times of restaurants, meaningless recommendations could be avoided. Moreover, by including semantic descriptions of the landscape such as water bodies and non-passable areas, the process of aggregating GPS tracks could be supported in a meaningful way (see also section 6.5.2).

The option to import data from related systems such as Qype (for places) or TrailGuru (for routes) could also prove beneficial regarding cold start problems. Conceptual compatibilities must be evaluated first, as well as potential costs and license issues. Crumblr's spatial database of aggregated place shapes and routes might also prove beneficial for other information systems relying on such content. An example for an open-source project that might be a good candidate for data exchange with Crumblr is OpenStreetMap[1], which aims at providing completely free geographic data such as street maps and places of interest.

Crumblr's server offers an open interface based on standardized technologies. However, to truly enable service and data interoperability, standards regarding protocols, interfaces, and messages are needed. The Open Geospatial Consortium (OGC)[2] is an increasingly growing international industry

[1] `http://wiki.openstreetmap.org`
[2] `http://opengeospatial.org`

consortium of leading companies, government agencies, and universities trying to make complex spatial information accessible to all kinds of applications by defining necessary standards.

New users currently must create a new account in order to use Crumblr. However, this might require too much effort for people who just want to have a look at the application. This issue could also be solved by enabling service interoperability. A technology for cross-site authentication recently gaining momentum is *OpenID*[3]. By employing OpenID, users can immediately start using Crumblr by providing their OpenID credentials obtained from a trusted OpenID provider. On the server side, Crumblr can be integrated in the OpenID infrastructure via a dedicated OpenID plugin for the Grails framework.

7.6 Closer Integration with Captchr

Several integration scenarios are imaginable when discussing the relation between Crumblr and Captchr. Currently, Käppler's Captchr system utilizes *explicitly captured* visits and user ratings (on a 1–5 scale) about places when building user profiles. In order to further integrate the two systems, a hybrid approach could include Captchr's explicit ratings to augment the inferred preferences. Moreover, Captchr's user profiles might be extended to include route visits captured by Crumblr, adding another component to the user profile model.

In Flickr, annotation (as textual tags) serves both personal and social purposes, which in turn increase the incentives for tagging and result in a relatively high number of annotations [Ames and Naaman, 2007]. Similarly, Captchr allows users to tag their blog entries. Tagging uploaded GPS routes might be a promising idea for Crumblr, improving the system's overall effectiveness and offering additional functionality.

Another point of integration could be Captchr's process of capturing microblog entries. Before submitting a new blog entry in Captchr, in most cases the user must first select a place as the current location from a map-based view. To further ease this process, Captchr could benefit from Crumblr's approach to associate visits to places. Specifically, Crumblr could first perform the place recognition algorithm on recently captured GPS data and recommend the most likely place for the current location.

Finally, when explaining the "general similarity" rating for places, Crumblr could forward users to Captchr's website responsible for more comprehensively explaining the "general similarity" between two users.

[3] `http://openid.net`

7.7 User Interface Issues

Section 6.6.3 demonstrated Crumblr's layered visualization approach to make the recommendation models more transparent to the user. As a possible improvement, users could be given the choice to aggregate the individual components into one "global" rating. In order to retain a certain level of transparency, the aggregation should be made transparent as well. This could be achieved by providing graphical interface elements to control the calculation of the global rating, for example. Each component of the recommendation model could be associated with a "slider", which expresses the weight given to the component. This way, users might easily combine the individual criteria when asking for personalized recommendations without giving up the transparency of the recommendation process. However, the aggregation of individual components requires different explanation techniques, which would have to be explored first.

In spite of Crumblr's aim at reducing the information overload, the small screen size makes it extremely challenging to optimize the visualization of recommended items. In its current version, Crumblr will display all items that fall within the specified radius of interest and are considered suitable for the given activity. If a threshold was added in the recommendation process, only the items with a rating value greater than the threshold would be displayed. However, if the items were not uniformly distributed over the screen, the image might still appear cluttered. Finding ways to visualize icons in an easy-to-understand uncluttered fashion while satisfying various users' requirements is a challenging task. Two fundamental tasks arise from this issue: the design of icons or the encoding of values with icons, and the problem of how to properly place icons on maps. A feasible solution might be the design of "aggregator icons" to indicate clusters of individual overlapping icons. It is also possible to indicate the number of aggregated icons by slightly increasing the size of aggregators. A survey of visualization strategies addressing the problem of overlapping icons can be found in [Buriget and Chittaro, 2008].

7.8 Privacy

Additional levels of sharing may prove useful. In some applications users are able to share private locations while excluding them from public view. Each level of added sharing, however, adds complexity to the interface, and thus more research is needed to identify ways to maintain ease of use.

7.9 Machine Learning Approaches

When machine learning algorithms have been applied to the analysis of people's spatial behavior, they have focused on people's transitions between places. For example, Liao et al. utilized relational Markov networks (RMN) to learn both high level human activities and significant places [Liao et al., 2005]. CitySense analyzes people's movement patterns to infer groups of similar places. In order to create a network of movement between locations, CitySense's algorithm first needs to identify and isolate the top N nightlife places in the city. Thus, machine learning algorithms address a rather higher-level problem than clustering algorithms do. The two approaches might be considered as complementary: the results of a visit recognition algorithm, which captures the characteristics of places and estimates activities, can serve as suitable input (activity and place candidates) for machine learning algorithms.

Chapter 8

Conclusion

This thesis presented an overview of the recent trends and developments in the Web and the mobile computing sphere. Since the turn of the millennium, the rising popularity of applications and services such as blogs, video sharing, and social networking platforms has changed the way we interact and network. Web 2.0 applications act as platforms creating collaborative, community-based sites, where users provide and organize the content.

A trend in the mobile computing world gains momentum and further promotes the vibrancy of platforms and services relying on user contributions – the wide-spread proliferation of open mobile platforms and devices. As the modern life style becomes faster and faster, people are in need of tools and technologies that assist them in the organization of their daily lives. People's everyday activities are related to the physical environment determined by space and time in two ways. On the one hand, by engaging in everyday activities, people develop personal preferences about places and routes they visit and a sense about the distinctions between them. On the other hand, observed patterns in people's spatial behavior can be used to characterize the visited places and routes. Personal preferences and spatial characteristics of places and routes can be captured via modern mobile technology and location-based services in a non-obtrusive way. Mobile recommendation systems utilizing observed spatial behavior can help users to find places and routes matching their interests and current situation.

Positioned in the converging field between the Web 2.0 paradigms and current mobile computing trends, this thesis proposed Crumblr, a novel approach to personalized recommendations of shared spatial content. In order to achieve this goal, three steps were identified: user observation, aggregation of spatial and contextual data, and personalized filtering and recommendation of spatial data. The following paragraphs provide a brief summary of the main contributions developed in this thesis.

Visit Recognition

This thesis developed a novel semi-automatic approach to acquire user preferences about places and routes, trying to achieve a good balance between unobtrusive automation and reliable user interaction methods. Crumblr's visit recognition algorithm was constructed by building on existing work on extracting place visits from GPS traces. Several suggestions were developed in order to effectively deal with noisy and sparse GPS data. Furthermore, custom extensions are proposed to improve the detection of false positives, i.e. urban canyons. Besides recognizing the visited places, Crumblr also aims at estimating the performed activities, based on aggregated history data.

Addressing both personal and social motivations for sharing location information, the visit recognition approach also accounts for the privacy issues inherent in systems relying on users' location data. By giving users control over their data, the system's acceptance is further promoted.

Route Aggregation

Existing work on route aggregation methods from [Morris et al., 2004] has been analyzed and several extensions have been suggested. Related systems aiming at constructing a route library do not utilize personalization techniques in order to further optimize the information flow when providing route information to users. By incorporating a layer of contextual data, i.e. the users' identities and the performed activities, Crumblr's route network is made suitable to personalized recommendations.

Modeling Places

Existing systems dealing with places employ a rather simplistic place model, describing places as single points in space. By offering a more accurate place model in form of geographic regions, Crumblr is able to improve both its visit recognition approach and the spatial assistance to users on the move. An approach to collaboratively update the place shapes has been designed and discussed as well.

Personalized Recommendations

By analyzing the collected place and route visits, user preferences can be implicitly constructed. The user's preferences and current situation can be used in order to personalize the retrieval of the aggregated spatial content. Several personalized recommendation models have been developed for this

purpose. Among the employed recommendation methods are user-based collaborative filtering techniques, several statistics-based methods, and a novel combination of places and routes. By capturing the subsequent place and route visits, the system closes its implicit feedback loop.

Explanations

Specific visualization and explanation techniques have been proposed in order to further drive the system's usability and acceptance. The suggested explanation approaches were derived from the employed recommendation models – by presenting the most important aspects of the utilized recommendation approach, the system's believability and trustworthiness is promoted. In particular, the demonstrated user-centered explanations also foster interactions between users, going beyond pure spatial assistance. Finally, Crumblr's user interface has been developed with the previously identified usability considerations for mobile devices in mind.

Working Prototype

In order to demonstrate the developed ideas, a proof-of-concept prototype has been developed for the Google Android mobile platform using open-source software solutions. A comprehensive "Admin Interface" has been developed and deployed on an application server for test purposes as well.

Evaluation

Even though the discussion of Crumblr's algorithms included simulated tests and user stories, the proposed ideas still lack practical evaluation to prove their effectiveness. This project's resource constraints prohibited a necessary large-scale quantitative or qualitative evaluation of the Crumblr system. Nevertheless, once Android devices are available, an actual user study could be performed. Some first thoughts on how this evaluation could be performed is given next.

Considering user observation, the effectiveness of the proposed visit recognition algorithm should be measured. Two metrics from information retrieval can be used for this purpose – *precision* and *recall*. Precision refers to the number of correctly recognized place visits divided by the total number of place visits recognized by Crumblr. Recall indicates what percentage of users' place visits were successfully recognized by Crumblr. The semi-automatic approach to sharing place information (i.e. associating visits to places and

estimating activities) can only be evaluated by performing a large-scale user study with a subsequent series of structured interviews.

The effectiveness of data aggregation methods (updating place shapes and aggregating GPS tracks) needs to be qualitatively investigated by collecting large amounts of GPS data.

In [Herlocker et al., 2004], Herlocker et al. concluded that evaluating recommendation systems and making the results comparable to related work is inherently difficult for several reasons. First, different algorithms may be better or worse on different data sets. Many collaborative filtering algorithms have been designed specifically for data sets in which there are many more users than items, and vice versa. The second reason that evaluation is difficult is that the goals for which an evaluation is performed may differ. Most of the early evaluation work focused specifically on the "accuracy" of collaborative filtering algorithms in "predicting" withheld ratings. However, even early researchers recognized that when recommendations are used to support decisions, it can be more valuable to measure how often the system leads its users to wrong choices. A few researchers have argued that these issues are all details, and that the bottom-line measure of a recommendation system's success should be user satisfaction [Herlocker et al., 2004]. Again, this phenomenon can only be investigated in a user study when Android handhelds become available.

Bibliography

[Abowd et al., 1999] Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., and Steggles, P. (1999). Towards a better understanding of context and context-awareness. In *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307, London, UK. Springer-Verlag.

[Alexander, 2006] Alexander, B. (2006). Web 2.0: A new wave of innovation for teaching and learning. *EDUCAUSE*. `http://net.educause.edu/ir/library/pdf/ERM0621.pdf` [Last access: 2008-05-10].

[Ames and Naaman, 2007] Ames, M. and Naaman, M. (2007). Why we tag: motivations for annotation in mobile and online media. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 971–980, New York, NY, USA. ACM.

[Anderson, 2007] Anderson, P. (2007). What is web 2.0 - ideas, technologies and implications for education. *JISC Technology and Standards Watch*. `http://www.jisc.ac.uk/media/documents/techwatch/tsw0701b.pdf` [Last access: 2008-05-14].

[AOL Developer Network, 2008] AOL Developer Network (2008). AOL announces new Open Mobile software platform. `http://dev.aol.com/openmobile/pr02112008` [Last access: 2008-07-24].

[Ashbrook, 2002] Ashbrook, D. (2002). Learning significant locations and predicting user movement with gps. In *ISWC '02: Proceedings of the 6th IEEE International Symposium on Wearable Computers*, page 101, Washington, DC, USA. IEEE Computer Society.

[Ashbrook and Starner, 2003] Ashbrook, D. and Starner, T. (2003). Using GPS to learn significant locations and predict movement across multiple users. *Personal Ubiquitous Comput.*, 7(5):275–286.

[Bajaj et al., 2002] Bajaj, R., Ranaweera, S. L., and Agrawal, D. P. (2002). Gps: Location-tracking technology. *Computer*, 35(4):92–94.

[Beenen et al., 2004] Beenen, G., Ling, K., Wang, X., Chang, K., Frankowski, D., Resnick, P., and Kraut, R. E. (2004). Using social psychology to motivate contributions to online communities. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 212–221, New York, NY, USA. ACM.

[Bellotti et al., 2008] Bellotti, V., Begole, B., Chi, E. H., Ducheneaut, N., Fang, J., Isaacs, E., King, T., Newman, M. W., Partridge, K., Price, B., Rasmussen, P., Roberts, M., Schiano, D. J., and Walendowski, A. (2008). Activity-based serendipitous recommendations with the Magitti mobile leisure guide. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1157–1166, New York, NY, USA. ACM.

[Berners-Lee, 1990] Berners-Lee, T. (1990). The worldwideweb browser. `http://www.w3.org/People/Berners-Lee/WorldWideWeb.html` [Last access: 2008-05-22].

[Berners-Lee, 1999] Berners-Lee, T. (1999). *Weaving the Web*. Texere Publishing Ltd.

[Bondi, 2000] Bondi, A. B. (2000). Characteristics of scalability and their impact on performance. In *WOSP '00: Proceedings of the 2nd international workshop on Software and performance*, pages 195–203, New York, NY, USA. ACM.

[Brady Forrest , 2008] Brady Forrest (2008). CitySense: Lets You Know What Everbody's Doing. O'Reilly Radar – `http://radar.oreilly.com/2008/06/citysense-reality-mining-iphone.html` [Last access: 2008-08-19].

[Brynjolfsson et al., 2003] Brynjolfsson, E., Hu, Y., and Smith, M. D. (2003). Consumer surplus in the digital economy: Estimating the value of increased product variety at online booksellers. *Management Science*, 49(11):1580–1596.

[Buriget and Chittaro, 2008] Buriget, S. and Chittaro, L. (2008). Decluttering of Icons based on Aggregation in Mobile Maps. In *Map-based Mobile Services – Design, Interaction and Usability*, pages 13–32. Springer, Berlin.

[Chen and Kotz, 2000] Chen, G. and Kotz, D. (2000). A survey of context-aware mobile computing research. Technical Report TR2000-381, Department of Computer Science, Dartmouth College.

[Cohen, 2003] Cohen, B. (2003). Incentives build robustness in bittorrent. `http://citeseer.ist.psu.edu/cohen03incentives.html` [Last access: 2008-06-10].

[Cormen et al., 2001] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms, Second Edition*. The MIT Press.

[Deitel et al., 2007] Deitel, H. M., Deitel, P. J., and Nieto, T. R. (2007). *Internet and World Wide Web How to Program*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 4th edition.

[Djuknic and Richton, 2001] Djuknic, G. M. and Richton, R. E. (2001). Geolocation and assisted gps. *Computer*, 34(2):123–125.

[Doreen Carvajal, 2006] Doreen Carvajal (2006). Dial right up for the photo sideshow. International Herald Tribuine – the global edition of the New York Times `http://www.iht.com/articles/2006/10/22/business/content.php` [Last access: 2008-07-24].

[Ester et al., 1996] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 226–231.

[Froehlich et al., 2006] Froehlich, J., Chen, M. Y., Smith, I. E., and Potter, F. (2006). Voting with your feet: An investigative study of the relationship between place visit behavior and preference. In *Ubicomp*, pages 333–350.

[Glover, 2004] Glover, A. (2004). Feeling Groovy. Published on IBM developerWorks. `http://www-128.ibm.com/developerworks/java/library/j-alj08034.html` [Last access: 2008-08-11].

[Goldberg et al., 1992] Goldberg, D., Nichols, D., Oki, B. M., and Terry, D. (1992). Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70.

[Google, 2007] Google (2007). Android - An Open Handset Alliance Project. Published on Google's website. `http://code.google.com/android/what-is-android.html` [Last access: 2008-05-21].

[Graham, 1972] Graham, R. L. (1972). An efficient algorithm for determining the convex hull of a finite planar set. *Inf. Process. Lett.*, 1(4):132–133.

[Guttman, 1984] Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. In Yormark, B., editor, *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, pages 47–57. ACM Press.

[Herlocker et al., 2000] Herlocker, J. L., Konstan, J. A., and Riedl, J. (2000). Explaining collaborative filtering recommendations. In *CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 241–250, New York, NY, USA. ACM.

[Herlocker et al., 2004] Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53.

[Hightower et al., 2005] Hightower, J., Consolvo, S., LaMarca, A., Smith, I. E., and Hughes, J. (2005). Learning and recognizing the places we go. In Beigl, M., Intille, S. S., Rekimoto, J., and Tokuda, H., editors, *Ubicomp*, volume 3660 of *Lecture Notes in Computer Science*, pages 159–176. Springer.

[Hilmar Schmundt , 2007] Hilmar Schmundt (2007). Can TomTom's Navigation 2.0 End Traffic Jams? Spiegel Online – `http://www.spiegel.de/international/business/0,1518,524160,00.html` [Last access: 2008-07-24].

[Horrigan, 2006] Horrigan, J. B. (2006). Home broadband adoption 2006. Technical report, Pew Internet & American Life Project. Home broadband adoption is going mainstream and that means user-generated content is coming from all kinds of internet users.

[inCode Wireless, 2005] inCode Wireless (2005). The next step for Location Based Services - How vertical integration, easy to use applications and the right business model can help to turn LBS into a success. `http://www.incodewireless.com/media/whitepapers/2005/WP_TheNextStepForLBS-(Feb.pdf` [Last access: 2008-07-23].

[Informa Telecoms & Media, 2006] Informa Telecoms & Media (2006). Mobile Search and Content Discovery (market report). `http://shop.telecoms.com/marlin/30000000861/MARKT_EFFORT/marketingid/20001339867?proceed=true&MarEntityId=1216066609950&entHash=1027127dc00` [Last access: 2008-07-24].

[Iqbal and Lim, 2007] Iqbal, M. U. and Lim, S. (2007). Privacy implications of automated GPS tracking and profiling. Second Workshop on Social Implications of National Security: From Dataveillance to Uberveillance, Wollongong, Australia – `http://www.homelandsecurity.org.au/files/RNSA_Social_Implications07_TEXT.pdf#page=225` [Last access: 2008-07-26].

[Japanese Statistics Bureau, 2006] Japanese Statistics Bureau (2006). Survey on time use and leisure activities. `http://www.stat.go.jp/english/data/shakai/index.htm` [Last accessed: 2008-08-25].

[Kang et al., 2004] Kang, J. H., Welbourne, W., Stewart, B., and Borriello, G. (2004). Extracting places from traces of locations. In *WMASH '04: Proceedings of the 2nd ACM international workshop on Wireless mobile applications and services on WLAN hotspots*, pages 110–118, New York, NY, USA. ACM.

[Käppler, 2008] Käppler, M. (2008). Captchr: Towards Context-Aware Exchange of Situated Social Media in Mobile Environments. Master's thesis, Department of Computer Science, Technical University of Kaiserslautern, Germany.

[Kaufman and Rousseeuw, 1990] Kaufman, L. and Rousseeuw, P. (1990). *Finding Groups in Data: an introduction to cluster analysis*. Wiley.

[Kindberg et al., 2005] Kindberg, T., Spasojevic, M., Fleck, R., and Sellen, A. (2005). The ubiquitous camera: An in-depth study of camera phone use. *IEEE Pervasive Computing*, 4(2):42–50.

[Klepis et al., 2001] Klepis, N., Nelson, W., Ott, W., Robinson, J., Tsang, A., Switzer, P., Behar, J., Hern, S., and Engelmann, W. (2001). Developing GIS-Supported Location-Based Services. In *Journal of Exposure Analysis and Environmental Epidemiology*, volume 11, pages 231–252.

[Kumparak, 2008] Kumparak, G. (2008). Android vs. LiMo: What's the difference? Published on MobileCrunch.com. `http://mobilecrunch.com/2008/05/14/android-vs-limo-whats-the-difference/` [Last access: 2008-08-11].

[Laningham, 2006] Laningham (2006). Ibm developerworks interviews: Tim berners-lee. developerWorks Interviews. `http://www-128.ibm.com/developerworks/podcast/dwi/cm-int082206.txt` [Last access: 2008-07-10].

[Lassica, 2007] Lassica, J. D. (2007). The Mobile Generation - Global Transformations at the Cellular Level. A Report of the Fifteenth Annual Aspen Institute Roundtable on Information Technology. `http://www.aspeninstitute.org/atf/cf/%7BDEB6F227-659B-4EC8-8F84-8DF23CA704F5%7D/C&S_The_Mobile_Generation.pdf` [Last access: 2008-04-23].

[Liao et al., 2005] Liao, L., Fox, D., and Kautz, H. A. (2005). Location-Based Activity Recognition using Relational Markov Networks. In Kaelbling, L. P. and Saffiotti, A., editors, *IJCAI*, pages 773–778. Professional Book Center.

[LiMo Foundation, 2007] LiMo Foundation (2007). LiMo Foundation – Homepage. `http://www.limofoundation.org/` [Last access: 2008-07-24].

[Ludford et al., 2007] Ludford, P. J., Priedhorsky, R., Reily, K., and Terveen, L. (2007). Capturing, sharing, and using local place information. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1235–1244, New York, NY, USA. ACM.

[Marmasse and Schm, 2000] Marmasse, N. and Schm, C. (2000). Location-aware information delivery with commotion. pages 157–171. Springer.

[Matyas, 2007] Matyas, S. (2007). Collaborative Spatial Data Acquisition – A Spatial and Semantic Data Aggregation Approach. *10th AGILE International Conference on Geographic Information Science*.

[Michel Deriaz , 2008] Michel Deriaz (2008). FoxyTag – a free, legal and collaborative system to signal speed cameras on mobile phones. `http://www.foxytag.com/en/presentation.html` [Last access: 2008-07-24].

[Miller and Larson, 1992] Miller, C. A. and Larson, R. (1992). An explanatory and "argumentative" interface for a model-based diagnostic system. In *UIST '92: Proceedings of the 5th annual ACM symposium on User interface software and technology*, pages 43–52, New York, NY, USA. ACM.

[Moore, 1965] Moore, G. E. (1965). Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117.

[Morris et al., 2004] Morris, S., Morris, A., and Barnard, K. (2004). Digital Trail Libraries. In *JCDL '04: Proceedings of the 4th ACM/IEEE-CS joint conference on Digital libraries*, pages 63–71, New York, NY, USA. ACM.

[Nielsen Mobile, 2008] Nielsen Mobile (2008). Critical Mass - The Worldwide State of the Mobile Web. `http://www.nielsenmobile.com/documents/CriticalMass.pdf` [Last access: 2008-07-24].

[Nivala and Sarjakovski, 2003] Nivala, A.-M. and Sarjakovski, L. T. (2003). An Approach to Intelligent Maps: Context Awareness. In *Workshop "HCI in mobile Guides"*, Udine, Italy.

[NokiaSiemensNetworks, 2005] NokiaSiemensNetworks (2005). New horizons. Published quarterly on http://www.nokiasiemensnetworks.com. `https://www.nokia.com/NOKIA_COM_1/Operators/Systems_&_Solutions/Mobile_Entry/Newsletter/Newsletter_Archive/pdf/nokia_newhorizons_issue_q4_2005.pdf` [Last access: 2008-06-16].

[NokiaSiemensNetworks, 2007] NokiaSiemensNetworks (2007). Empowering communities with affordable communications. Published on http://www.nokiasiemensnetworks.com. `http://www.nokiasiemensnetworks.com/NR/rdonlyres/73F16444-BD37-42AB-BE0B-4AC8CAD29DEA/0/NGM_brochure_AW.pdf` [Last access: 2008-07-20].

[Open Handset Alliance, 2008] Open Handset Alliance (2008). OHA - Open Handset Alliance Homepage. `http://www.openhandsetalliance.org` [Last access: 2008-07-24].

[O'Reilly, 2005] O'Reilly, T. (2005). What is web 2.0 - design patterns and business models for the next generation of software. `http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html` [Last access: 2008-07-09].

[Pandey and Agrawal, 2006] Pandey, S. and Agrawal, P. (2006). A survey on localization techniques for wireless networks. *Zho-ngguó go-ngchéng xuéka-n ISSN 0253-3839*, 29(7):1125–1148.

[Pendleton, 2002] Pendleton, G. (2002). The fundamentals of gps. `http://www.directionsmag.com/article.php?article_id=228` [Last access: 2008-07-12].

[Reichenbacher, 2003] Reichenbacher, T. (2003). Adaptive Methods for Mobile Cartography. In *Proceedings of the 21st International Cartographic Conference ICC*, Cartographic Renaissance, pages 1311–1321, Durban, South Africa.

[Reichenbacher, 2004] Reichenbacher, T. (2004). Mobile Cartography - Adaptive Visualization of Geographic Information on Mobile Devices. PhD Thesis, TU München.

[Ricci and Nguyen, 2007] Ricci, F. and Nguyen, Q. N. (2007). Acquiring and revising preferences in a critique-based mobile recommender system. *IEEE Intelligent Systems*, 22(3):22–29.

[Rote, 1991] Rote, G. (1991). Computing the minimum hausdorff distance between two point sets on a line under translation. *Inf. Process. Lett.*, 38(3):123–127.

[Rudolph, 2007] Rudolph, J. (2007). *Getting Started With Grails*. Lulu.com.

[Sarwar et al., 2001] Sarwar, B., Karypis, G., Konstan, J., and Reidl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 285–295, New York, NY, USA. ACM.

[Schafer et al., 1999] Schafer, J. B., Konstan, J., and Riedi, J. (1999). Recommender systems in e-commerce. In *EC '99: Proceedings of the 1st ACM conference on Electronic commerce*, pages 158–166, New York, NY, USA. ACM.

[Schauer, 2005] Schauer, B. (2005). What put the '2' in web 2.0? Technical report, Adaptive Path. `http://www.adaptivepath.com/images/publications/essays/What_puts_the_2_in_Web_20.pdf` [Last access: 2008-05-23].

[Schilit et al., 1994] Schilit, B., Adams, N., and Want, R. (1994). Context-Aware Computing Applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, US.

[Schmidt-belz et al., 2002] Schmidt-belz, B., Nick, A., Poslad, S., and Zipf, A. (2002). Personalized and location-based mobile tourism services. In *Proc. of Mobile-HCI*.

[Schwab and Kobsa, 2002] Schwab, I. and Kobsa, A. (2002). Adaptivity through unobstrusive learning. *KI*, 4:3.

[Sense Networks , 2008] Sense Networks (2008). CitySense. CitySense Homepage – `http://www.sensenetworks.com/machine_learning.php` [Last access: 2008-08-19].

[Siemens, 2001] Siemens (2001). Siemens location services (lcs) — mobile phones with a sense of place. `www.siemens.ie/mobile/Press/LocationBasedServices.doc` [Last access: 2008-07-23].

[Steiniger et al., 2006] Steiniger, S., Neun, M., and Edwardes, A. (2006). Foundations of Location Based Services. `http://www.geo.unizh.ch/publications/cartouche/lbs_lecturenotes_steinigeretal2006.pdf` [Last access: 2008-07-23].

[Strahan, 2002] Strahan, R. (2002). Location Sensing Technologies. Report No. 2.2.1.1.2002. `emc2.ucd.ie/deliverables/e=mc2.2.1.1.2002.doc` [Last access: 2008-06-10].

[Surowiecki, 2005] Surowiecki, J. (2005). *The Wisdom of Crowds*. Anchor.

[Toivonen, 2007] Toivonen, S. (2007). Web on the Move - Landscapes of Mobile Social Media. Espoo 2007, VTT Tiedotteita, Research Notes 2403 – `http://www.vtt.fi/inf/pdf/tiedotteet/2007/T2403.pdf` [Last access: 2008-05-24].

[Virrantaus et al., 2001] Virrantaus, K., Markkula, J., Garmash, A., Terziyan, V. Y., Veijalainen, J., Katasonov, A., and Tirri, H. (2001). Developing GIS-Supported Location-Based Services. In *WISE (2)*, pages 66–75.

[Yuichiro and Masanori, 2006] Yuichiro, T. and Masanori, S. (2006). Cityvoyager: An outdoor recommendation system based on user location history. In *International Conference on Ubiquitous Intelligence and Computing*, pages 625–636. Springer, Berlin.

[Zhou et al., 2004] Zhou, C., Frankowski, D., Ludford, P., Shekhar, S., and Terveen, L. (2004). Discovering personal gazetteers: An interactive clustering approach. In *In Proc. ACMGIS*, pages 266–273. ACM Press.

[Zhou et al., 2005a] Zhou, C., Ludford, P. J., Frankowski, D., and Terveen, L. G. (2005a). How do people's concepts of place relate to physical locations? In *INTERACT*, pages 886–898.

[Zhou et al., 2005b] Zhou, C., Terveen, L., and Shekhar, S. (2005b). Discovering personal paths from sparse gps traces. In *In Proc. of the JCIS 2005 Workshop on Data Mining*.

List of Figures

List of Figures

List of Tables

Appendix A

Place Visit Recognition

Appendix A describes a selection of fictive test cases used to get preliminary results of Crumblr's place visit recognition algorithm. The algorithm has been deployed on the Crumblr server for testing purposes and made accessible via the Admin Interface. The following scenarios are depicted with satellite photos obtained from Google Maps. A fictive user Alice is assumed to have carried around a Google Android based device with the Crumblr client installed and running. Alice's GPS position data is shown in form of solid white and yellow rectangles. The white rectangles represent data that was ignored by Crumblr's visit recognition algorithm, whereas the yellow rectangles belong to the recognized place visits. To visually distinguish clusters, each clustered point has its cluster ID displayed in its yellow rectangle.

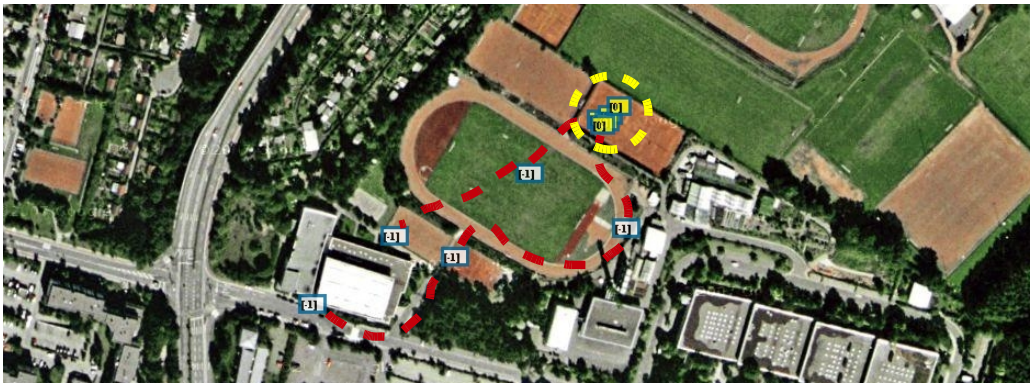


Figure A.1: The image shows Alice's path to the tennis courts, where she spent some time. Finally, she moved away from the courts. Crumblr correctly recognizes the visit to the tennis court.

Figure A.2: This image shows Alice's path from the city to the beach. Before going to the beach, she stops by her friend's house ("cluster" 0). Since the constraints from Definition 6.1.4 are fulfilled, Crumblr correctly detects this as a place visit. Finally, the beach visit is detected as well (cluster 1).

Figure A.3: This image shows a scenario where the algorithm fails to detect a visit to an indoor place (red circle). The constraints from Definition 6.1.4 are not fulfilled, because Alice produced two very close GPS readings before entering the house (e.g. she took a call before going in). This led to the extrapolation of a too small perimeter, causing Crumblr to interpret the subsequent loss of signal as an urban canyon. Instead of considering only the last two points before losing signal, a possible improvement would consider the last $k > 2$ points, and extrapolate the speed by employing a weighting method over the last $k - 1$ steps.

Figure A.4: This image shows Alice driving in New York/New Jersey. By driving through the Lincoln Tunnel (under the Hudson River), her phone's GPS signal gets lost. On the other side of the river, the position data is collected normally again. The constraints from Definition 6.1.4 are not fulfilled, because Alice was constantly moving. This caused Crumblr to correctly ignore the loss of signal.

Figure A.5: This image shows Alice's movement history in a part of New York. The two yellow circles represent two visits to the same place. Crumblr recognizes these clusters based on Def. 6.1.1 and merges them according to Kang et al.'s merging condition.

Figure A.6: This image shows another example of Alice's movement history. The two yellow polygons represent two visits to two different places. Crumblr recognizes these clusters based on Def. 6.1.1 and does not merge them, since they do not fulfill Kang et al.'s merging condition.

Appendix B

Updating Place Shapes

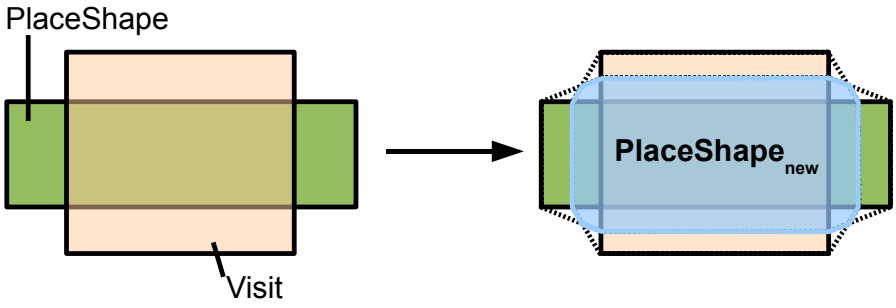


Figure B.1: Two overlapping rectangles are merged to a polygon. The resulting shape PlaceShape$_{new}$ expresses both positive growth in the vertical direction and negative growth in the horizontal direction.

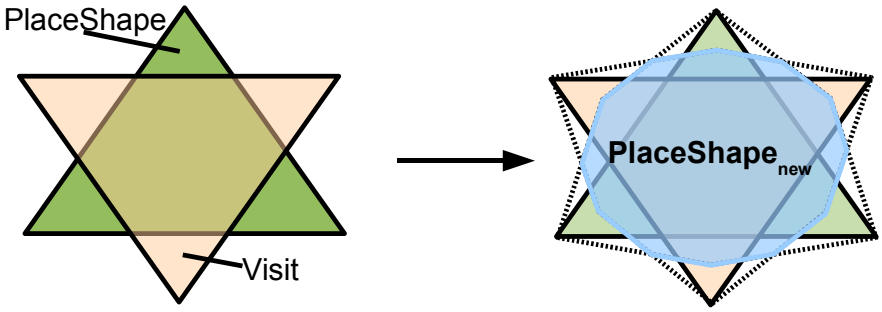


Figure B.2: Two overlapping triangles are merged to a polygon. The resulting shape PlaceShape$_{new}$ accounts for both positive and negative growth.

Appendix C

Admin Interface

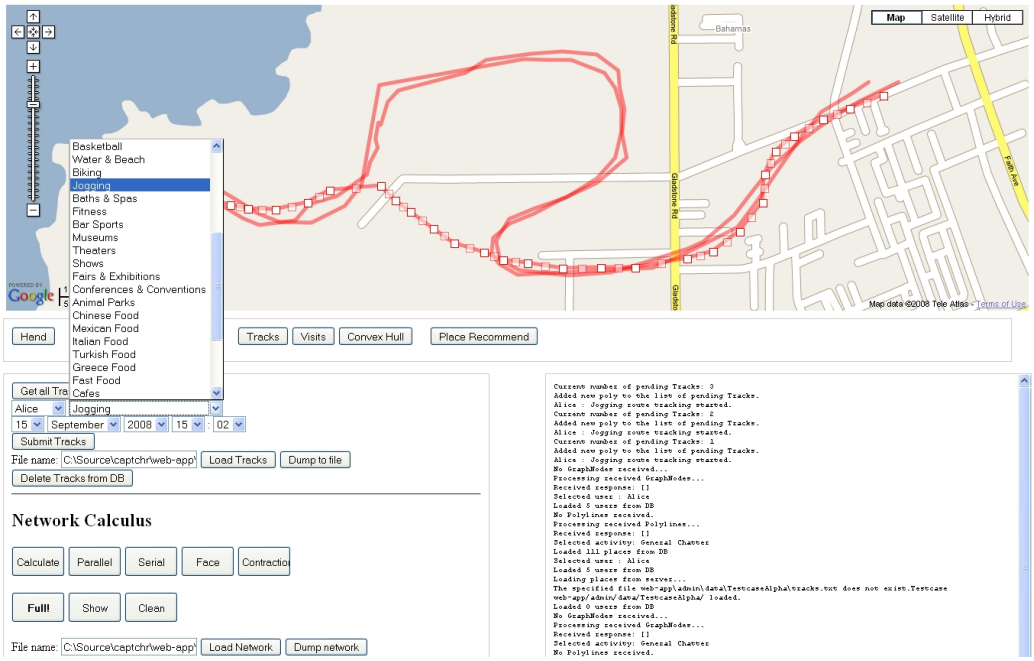


Figure C.1: The Admin Interface: creating GPS route tracks for user Alice and activity "Jogging"

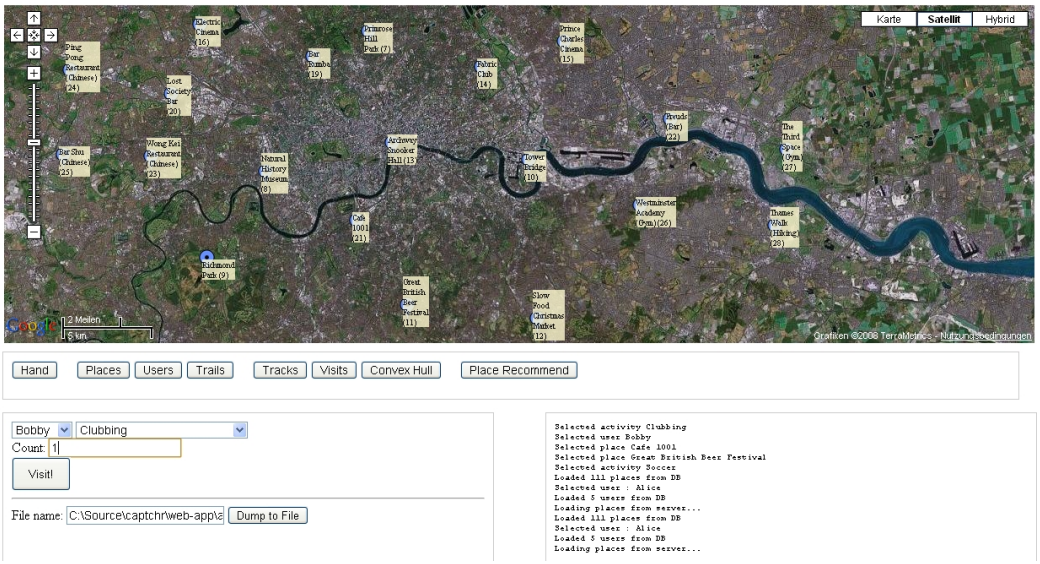


Figure C.2: The Admin Interface: adding a place visit for user "Bobby", activity "Clubbing", and place "Cafe 101" (London, UK)

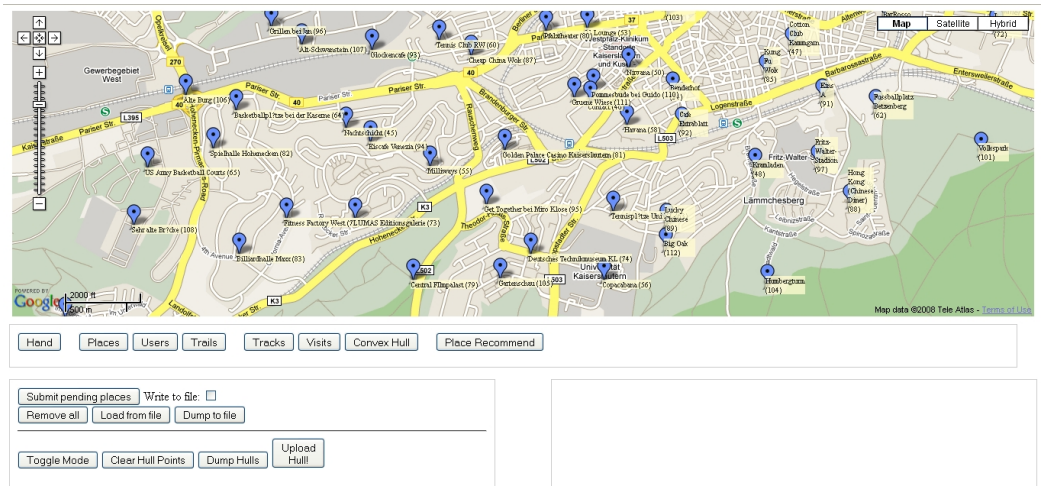


Figure C.3: The Admin Interface: places created in Kaiserslautern