Aalborg Universitet

SPATIAL KEYWORD QUERYING

RANKING EVALUATION AND
EFFICIENT QUERY PROCESSING

BY
ILKCAN KELES

DISSERTATION SUBMITTED 2018

AALBORG UNIVERSITY
DENMARK

Spatial Keyword Querying: Ranking Evaluation and Efficient Query Processing

Ph.D. Dissertation
Ilkcan Keles

Dissertation submitted April, 2018

Abstract

Due to the widespread adoption of mobile devices with positioning capabilities, notably smartphones, users increasingly search for geographically nearby information on search engines. Further, an analysis of user behavior finds that users not only search for local content, but also take action with respect to search results. In step with these developments, the research community has proposed various kinds of spatial keyword queries that return ranked lists of relevant points of interest. These proposals generally come with advanced query processing techniques, the goal being to make it possible for users to find relevant information quickly. Most of the proposals employ a simple ranking function that takes only textual relevance and spatial proximity into account. While these proposals study the query processing efficiency, they are generally weak when it comes to evaluation of the result rankings. We believe that ranking evaluation for spatial keyword queries is important since it is directly related to the user satisfaction.

The thesis addresses several challenges related to ranking evaluation for spatial keyword queries. The first challenge we address is forming ground-truth rankings for spatial keyword queries that reflect user preferences. The main idea is that the more similar an output ranking is to the ground-truth ranking, the better the output ranking is. The thesis proposes methods based on crowdsourcing and vehicle trajectories to address this challenge. These methods make it possible for researchers to propose novel ranking functions and to assess the performance of these functions. As such, the thesis makes a step towards more advanced and complex ranking functions that correspond better to user preferences. The contributions of the thesis can also be used to evaluate hypotheses regarding different keywords and geographical regions. Along these lines, it might be possible to employ different ranking functions for different queries in the same system. The thesis also addresses the problem of detecting the visited points of interest in a GPS dataset and proposes algorithms to tackle this problem. These visits offer insight into which points of interest are of interest to drivers and offer a means of ranking for points of interest.

More specifically, the thesis first proposes a technique based on crowd-

sourcing to obtain partial rankings corresponding to user preferences. The method employs pairwise relevance questions and uses a gain function to decrease the number of questions that need to be processed to form a ranking, thus reducing the cost of crowdsourcing. The resulting partial rankings can be used to assess the quality of ranking functions.

Next, the thesis proposes a system, CrowdRankEval, to evaluate ranking functions for a given set of queries and corresponding query results obtained using various ranking functions. The system utilizes the aforementioned crowdsourcing-based method to form ground-truth rankings for queries. The system provides an easy-to-use interface that allows users to visualize the rankings obtained from crowdsourcing and to view the evaluation results.

Further, the thesis proposes a crowdsourcing-based approach to evaluate two ranking functions on a set of spatial keyword queries. The approach takes budget constraints into account and utilizes a learn-to-rank method and an entropy definition to determine the most important questions to compare two ranking functions for a given query.

The thesis also proposes a method that uses GPS data to extract ground-truth rankings for spatial keyword queries. The idea is to use historical trips to points of interest to determine the relative popularity of points of interest. The experimental findings suggest that the proposed method is capable of capturing user preferences.

Finally, the thesis formalizes a so-called k-TMSTC query that targets users looking for groups of points of interest instead of single points of interest. Two algorithms based on density-based clustering are proposed to process this query. Experiments show that the proposed methods support interactive search.

Resumé

Udbredelsen af mobile enheder, i særdeleshed smartphones med indbygget GPS, har medført at brugere i stigende grad søger efter information om det område de befinder sig i. Derudover viser analyse af brugeradfærd at brugere ikke alene søger efter information om deres omgivelser, men også agerer på den. Som følge deraf har forskere foreslået en række forskellige spatiale søgeordsforespørgsler der tager både brugerens søgeord og lokation i betragtning og returnerer en rangeret list af interessepunkter. Disse forslag kommer med typisk med avancerede teknikker til processering af forespørgsler med henblik på hurtigt at give brugere svar på deres forespørgsler. De fleste af forslagene bruger en simpel rangeringsfunktion, der kun tager tekst og spatial relevans i betragtning og processerer forespørgsler hurtigt, men overvejer ikke kvaliteten af svaret på forespørgslen. Vi mener at evalueringen af spatiale søgeordsforespørgslers resultater er vigtig da kvaliteten af resultaterne er direkte relateret til brugertilfredshed.

Denne afhandling adresserer adskillige udfordringer i forbindelse med evalueringen af spatiale søgeordsforespørgsler. Den første udfordring vi adresserer er at etablere sande rangeringer af forespørgselsresultater baseret på brugerpræferencer. Jo tættere et forespørgselsresultat er på den sande rangering jo bedre er den. Afhandlingen foreslår metoder til at disse at adressere denne udfordring baseret på crowdsourcing og bilers færden i vejnet. Disse metoder tillader forskere at foreslå og evaluere nye rangeringsfunktioner. Derfor bringer denne afhandling os tættere på mere avancerede og komplekse rangeringsfunktioner der stemmer bedre overens med brugerpræferencer. Afhandlingens bidrag kan endvidere bruges til at evaluere hypoteser vedrørende forskellige søgeord og geografiske lokationer. Det kan måske også lade sig gøre at bruge forskellige rangeringsfunktioner til forskellige typer af forespørgsler. Afhandlingen foreslår også algoritmer til at adressere problemet med detektering af besøgte punkter fra GPS-data. Besøgene giver indsigt i hvilke punkter er interessante for bilister og en måde at rangere interessepunkterne.

Mere specifikt foreslår afhandlingen først en teknik baseret på crowdsourcing til at opnå delvise rangeringer svarende til brugerpræferencer. Tek-

nikken stiller en række spørgsmål til crowdsourcingarbejdere og bruger en funktion til at afgøre hvilket spørgsmål der skal stilles næste gang for at få mest mulig indsigt i brugerpræferencer og reducerer derfor omkostningen ved crowdsourcing. De resulterende delvis rangeringer kan bruges til at evaluere kvaliteten af rangeringsfunktioner.

Dernæst foreslår afhandlingen et system, CrowdRankEval, til at evaluere rangeringsfunktioner for et givet sæt af forespørgsler med tilhørende rangerede resultater. Systemet benytter den tidligere nævnte crowdsourcemetode til at danne sande rangeringer til at evaluere rangeringsfunktionerne.

Endvidere præsenterer afhandlingen en crowdsourcingbaseret metode til at evaluere to rangeringsfunktioner på et sæt af spatiale søgeordsforespørgsler. Metoden tager omkostningsbegrænsninger i betragtning og bruger en maskinlært rangeringsfunktion samt en definition af entropi til at afgøre hvilke spørgsmål er mest væsentlige når to rangeringsfunktioner sammenlignes.

Afhandlingen foreslår også en metode der benytter sig af GPS-data til at finde sande rangeringer af spatiale søgeordsforespørgsler. Metoder bruger historiske ture til interessepunkter til at afgøre interessepunkternes relative popularitet. Eksperimenter viser at den foreslåede metode er i stand til at finde brugerpræferencer.

Endelig formaliserer afhandlingen den såkaldte k-TMSTC forespørgsel som er målrettet brugere der leder efter grupper af interessepunkter fremfor enkelte interessepunkter. Afhandlingen foreslår to algoritmer baseret på densitetsbaseret klyngedannelse til processering af denne type forespørgsel. Eksperimenter viser at de foreslåede metoder understøtter interaktiv søgning.

Acknowledgments

I would like to thank a number of people who have helped and supported me during my Ph.D. studies.

First of all, I would like to thank my supervisor Christian S. Jensen for giving me the opportunity to work with him. I am grateful for his constructive feedback and invaluable support during my Ph.D. studies. He has always been very patient with me, and I am extremely thankful for his support. I would also like to thank my co-supervisor Simonas Šaltenis for having the time for discussions and providing constructive comments regarding my work. His door was always open for discussing the next steps.

Further, I would like to thank all my colleagues and friends at the Database, Programming and Web Technologies (DPW) Group for providing a cozy, friendly, stimulating and highly intellectual work environment. They were always ready to help me whenever I am in need. Special thanks goes to Nurefşan Gür for her friendship and encouragement throughout these years, to Rudra Nath and Muhammad Aamir for their support on getting used to Aalborg when I first came here, to Robert Waury and Søren Kejser Jensen for the interesting discussions we had regarding our research, and for going through the thesis summary and providing me some comments, and to Tobias Skovgaard Jepsen for helping me on writing the Danish abstract. I would also like to appreciate the administrative staff Helle Schroll and Helle Westmark who were always there to support and guide.

I would also like to thank Peer Kröger and Matthias Schubert for hosting me at Ludwig Maximilian University of Munich as a visiting Ph.D. student and for their inspiring suggestions and generous support during my stay abroad.

This thesis would not have been a reality without the help, support, and patience of my family, especially my wife Emel. She was always there to support and help me and she was always patient with me when I have deadlines piling up. I cannot thank enough to my wife for being that patient, caring and understanding throughout my PhD. My family and my wife's family also helped me with their constant support and prayers even though they were far away.

Contents

II Papers 41

Contents

Contents

Thesis Details

The main body of this thesis consists of the following papers.

[A] Ilkcan Keles, Simonas Šaltenis, Christian Søndergaard Jensen, "Synthesis of Partial Rankings of Points of Interest Using Crowdsourcing". *In Proceedings of the 9th Workshop on Geographic Information Retrieval (GIR 2015), Paris, France, pages 15:1-15:10, 2015.*

[B] Ilkcan Keles, Christian Søndergaard Jensen, Simonas Šaltenis, "CrowdRankEval: A Ranking Function Evaluation Framework for Spatial Keyword Queries". *In Proceedings of the 17th IEEE International Conference on Mobile Data Management (MDM 2016), Porto, Portugal, pages 353-356, 2016.*

[C] Jinpeng Chen, Hua Lu, Ilkcan Keles, Christian Søndergaard Jensen, "Crowdsourcing Based Evaluation of Ranking Approaches for Spatial Keyword Querying". *In Proceedings of the 18th IEEE International Conference on Mobile Data Management (MDM 2017), Daejeon, South Korea, pages 62-71, 2017.*

[D] Ilkcan Keles, Christian Søndergaard Jensen, Simonas Šaltenis, "Extracting Rankings for Spatial Keyword Queries from GPS Data". *In Proceedings of the 14th International Conference on Location Based Services (LBS 2018), Zurich, Switzerland, pages 173-194, 2018.*

[E] Ilkcan Keles, Matthias Schubert, Peer Kröger, Simonas Šaltenis, Christian Søndergaard Jensen, "Extracting Visited Points of Interest from Vehicle Trajectories" *In Proceedings of the Fourth International ACM Workshop on Managing and Mining Enriched Geo-Spatial Data (GeoRich 2017), Chicago, Illinois, pages 2:1-2:6, 2017.*

[F] Ilkcan Keles, Dingming Wu, Simonas Šaltenis, Christian Søndergaard Jensen, "Transportation-mode Aware Spatial Keyword Querying for Point of Interest Clusters". *To be submitted for journal publication*.

This thesis has been submitted for assessment in partial fulfillment of the PhD degree. The thesis is based on the submitted or published scientific papers which are listed above. Parts of the papers are used directly or indirectly in the extended summary of the thesis. As part of the assessment, co-author statements have been made available to the assessment committee and are also available at the Faculty. The permission for using the published and accepted articles in the thesis has been obtained from the corresponding publishers with the conditions that they are cited and DOI pointers and/or copyright/credits are placed prominently in the references.

Part I

Thesis Summary

Thesis Summary

1 Introduction

1.1 Background and Motivation

Commercial search engines process billions of queries on a daily basis. A recent document published about usage statistics of Google[1] [1] reports that the search engine processed more than 9 billion queries per day in 2016. Further, the number of people who access the web from geo-positioned devices such as smartphones and tablets increases day by day. In addition, the amount of geo-tagged web content, i.e., content on the web that is mapped to a spatial location also increases. As a result of this, the percentage of queries that have local intent on commercial search engines is substantial. An analysis on the users' local search behavior [19] states that 80% of the users look for geographically relevant information on the search engines. Another finding of the same analysis is that half of the users with local intent who access the search engines using tablets visit one of the stores on the same day. This usage statistics demonstrates the significance of the location-based web services.

The research community has proposed many different spatial keyword query types in order to address the needs of the users with local intent. The aim of spatial keyword queries is to find nearby points of interest that are relevant to user-provided keywords. A point of interest (PoI) in this context is generally defined with an identifier, a location and a document containing the textual information regarding the PoI. It might also have other attributes such as opening hours information, rating and expensiveness level. In this thesis, we focus on top-k spatial keyword queries. A top-k spatial keyword query $q = \langle \lambda, \psi, k \rangle$ takes three arguments: a query location λ, a set of query keywords ψ, and the requested number of PoIs k [9]. It outputs a ranked list of k PoIs. The PoIs are ranked according to a score produced by a ranking function that takes the relevance of the PoI to the query into

---

[1]https://www.google.com

account. A ranking function may consider various attributes regarding the PoIs, the query region, and the neighborhood of the PoI. However, most of the proposals in the literature utilize a simple ranking function, namely a weighted linear combination of the textual relevance of the PoI document to the query keywords and the spatial proximity of the PoI location to the query location. An example top-k spatial keyword query corresponding to a tourist's search for restaurants to have a dinner in Aalborg is shown in Figure 1. The black marker denotes the location of the tourist, i.e., the query location, the keyword is "restaurant", and k is set to 20 in this example. The red markers denote the output PoIs, and the number on the markers denote the rank of the PoI in the output ranking.

Fig. 1: An Example Top-k Spatial Keyword Query (Map data ©2018 Google)

Existing studies [11, 15, 27, 35] regarding top-k spatial keyword queries focus on efficient processing of the queries and cover query performance evaluation of the proposed methods. However, evaluation of the quality of the ranking functions is not included in these studies. We believe that such evaluation is quite important since it is directly related to the user satisfaction with the location-based web services. If users can find what they look for among the highest ranked PoIs, they will be more satisfied with the output. Thus, a good ranking function should produce rankings that correspond to the user preferences which are often implicit.

One approach to evaluate a ranking function is to compare its output to a

ground-truth ranking that corresponds to the users' choices. The main idea is that the closer the output ranking produced by a ranking function is to the ground-truth ranking, the better the ranking function is. The main challenge is computing such a ground-truth ranking.

This thesis considers two methods to build ground-truth rankings for spatial keyword queries: crowdsourcing and extracting from vehicle trajectory data. Note that, the ground-truth ranking building proposed in this thesis cannot be used as a substitute for query processing since it takes too much time in both cases. In addition to that, ground-truth rankings can be built only for queries in limited geographical areas by the proposed methods. For the case of crowdsourcing, the area is limited to the geographical regions that have enough workers to provide meaningful conclusions. For the vehicle trajectory data, the area is limited to the geographical coverage of the GPS records. We expect that user-data driven evaluations of ranking functions in well-chosen test areas can lead to conclusions and insights that apply more generally.
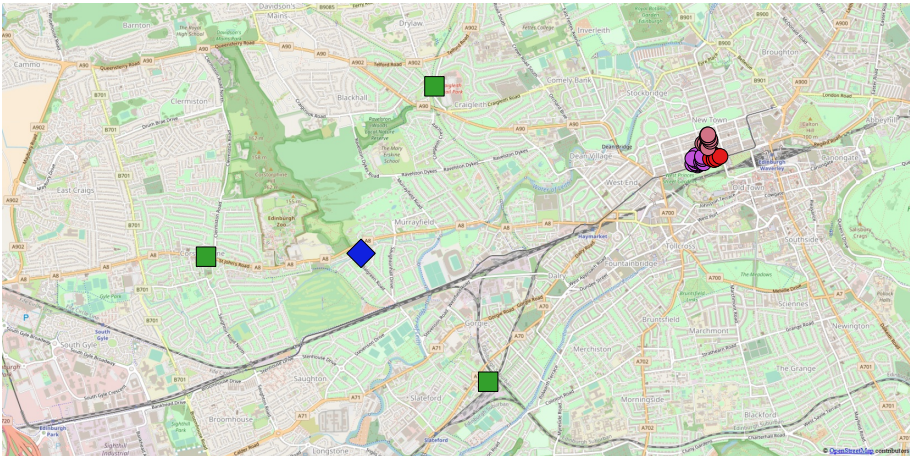
Fig. 2: An Example Groups Query (Map data ©OpenStreetMap contributors)

Another shortcoming of the spatial keyword query proposals in the literature is that most of these proposals return a list of individual objects as output. However, a user might be interested in finding a group of objects in some cases. For instance, a user who wants to buy a pair of shoes might want to check different options in a couple of shops before deciding what to buy. Another use case would be a tourist who wants to do some sightseeing around the city. It would be much easier for this tourist to visit a group of attractions close to each other instead of visiting attractions that are located far away from each other. An example query for a user who is looking for fashion stores in Edinburgh that corresponds to the first use case is given

in Figure 2. The blue diamond denotes the location of the user. The green squares denote the 3 closest PoIs and the circles denote the top-3 groups of relevant PoIs. This example illustrates a case where it would be much more convenient for the user to visit the group of PoIs instead of the closest PoIs. The user might also find a larger number of options if he visits the group of PoIs. In order to address these use cases, this thesis considers a new spatial keyword query type to retrieve groups of relevant objects.

1.2 Crowdsourcing-based Ranking Evaluation

Crowdsourcing [22] has been used extensively in the recent years to accomplish tasks that are difficult for computers but easy for human beings. Examples of such tasks include collecting relevance assessments for information retrieval systems [2, 7], and answering database queries using human knowledge [18, 31, 34]. There are many crowdsourcing platforms such as Amazon Mechanical Turk[2] and Crowdflower[3] that provide interfaces for publishing tasks, assigning tasks to selected workers out of the platform's worker pool, and paying fees to the workers for completed tasks. A task can be very simple such as answering a question and tagging an image or complex such as summarizing a text and translating a text. In this thesis, we employ crowdsourcing to form ground-truth rankings and to evaluate ranking functions for top-k spatial keyword queries.

Paper A considers using crowdsourcing to form ground-truth rankings for top-k spatial keyword queries. It proposes an algorithm to obtain partial rankings of the given PoIs for a given query. It also provides a simulation-based evaluation of the algorithm and shows that the algorithm performs well in various settings. Paper B demonstrates a complete framework to evaluate the performance of ranking functions for spatial keyword queries on top of the crowdsourcing platform Crowdflower. Paper C addresses a slightly different problem. Instead of forming the ground-truth ranking of all relevant PoIs, it considers the case of having two ranking functions to compare and having a budget constraint on the number of tasks performed by crowdsourcing workers. So, the problem is to find out which ranking function performs better for the given queries without exceeding the given budget.

1.3 GPS-based Ranking Evaluation

GPS trajectories provide information regarding location history and movement patterns of users. GPS trajectory data is utilized to identify the visits

---

[2]https://www.mturk.com/
[3]https://www.crowdflower.com/

to PoIs [20, 32], to recommend locations to the users based on their trajectories [29, 42, 43], and to understand the significant places within the region [4, 6, 10, 44]. The source of GPS data can be either the mobile devices or the vehicles. The difference is that the former includes the GPS recordings corresponding to walking as well as driving.

In this thesis, we utilize vehicle-based GPS trajectories obtained from 354 drivers for a period of 9 months with 1 hz frequency. The complete dataset contains around 0.4 billion records mostly located in or around Aalborg, Denmark. Paper D addresses the problem of constructing rankings for spatial keyword queries using GPS data. Paper E tackles the problem of discovering visits to PoIs from vehicle trajectory data. This information is needed to rank the PoIs.

1.4 Spatial Keyword Querying for PoI Clusters

Most of the spatial keyword query types has a single-PoI granularity which means that they return an output consisting of individual PoIs. The users might request groups instead of single PoIs in some use cases. Collective spatial keyword queries [12, 30] are proposed for the users whose needs cannot be fulfilled by a single PoI but a group of PoIs. An example would be a person who wants to visit a book store and have a drink afterwards. So, this person needs a set consisting of at least a book store and a cafe/bar located close to each other. Another use case is when the person needs a group of similar PoIs so as to check different options and prices.

Top-k groups spatial keyword query [37] and *top-k spatial textual cluster query* (*k*-STC) [39] types are proposed for the latter kind of user behavior. Both queries return top-*k* groups of relevant PoIs according to a cost function. The first query type considers any grouping of the objects, and the second query type only considers the density-based clusters [16]. However, both of these proposals have some drawbacks. The cost function utilized by top-*k* groups query does not consider in-group distances in a suitable way since it only considers the diameter of the group. A group might have PoIs far away from each other even though it has the same diameter with another group that has PoIs closer to each other. The first drawback of top-*k* clusters query is that the user needs to provide density-based clustering parameters as the query parameters. This is not realistic since the user might not know the region that well. The second drawback is that the cost function of *k*-STC query does not consider the transportation-mode of the user. In other words, the cost of a cluster is not affected by the users' transportation modes to the cluster and in the cluster.

Paper F formalizes top-*k* transportation-mode aware spatial textual cluster (*k*-TMSTC) queries and proposes methods to process them in order to address the drawbacks of *k*-STC queries.

1.5 Organization

The rest of this summary is organized as follows. Section 2 summarizes Paper A and describes a new algorithm named PointRank to obtain partial rankings for spatial keyword queries using crowdsourcing. Section 3 summarizes Paper B and describes a framework to evaluate ranking functions for spatial keyword queries. The framework utilizes the PointRank algorithm to form ground-truth rankings of queries. Section 4 summarizes Paper C and describes a method to assess which ranking function performs better given two ranking functions and a set of queries. The method aims to complete the assessment efficiently and accurately within a given budget. Section 5 summarizes Paper D and explores the idea of using GPS records instead of crowdsourcing to form rankings for spatial keyword queries. The proposed method contains a new stop assignment algorithm to find the visited PoI for a given stop. It also contains a smoothing method based on PageRank algorithm to extend the spatial coverage of the method. Section 6 summarizes Paper E and focuses on the problem of stop assignment to PoIs and describes a new algorithm based on a Bayesian network. Section 7 summarizes Paper F and introduces k-TMSTC queries and outlines two algorithms to process these queries. Finally, Section 8 gives a summary of the contributions in the thesis, and Section 9 concludes the thesis summary.

2 Crowdsourcing-based Ranking Synthesis

This section gives an overview of Paper A [25].

2.1 Problem Motivation and Statement

Ranking synthesis for top-k spatial keyword queries is the first step towards the ranking function evaluation since the synthesized ranking can be used as ground-truth ranking for evaluation purposes. In order to synthesize rankings, we need a way to obtain user feedback about the relevant PoIs for a specific query. We choose to use crowdsourcing since it provides easy access to human knowledge and it has been used to obtain rankings of objects in the literature [14, 38, 41]. Stoyanovich et al. [38] utilizes listwise relevance questions to obtain rankings from each crowdsourcing worker. In other words, each worker is supposed to provide a complete ranking for a given list of objects. In the context of spatial keyword queries, it would be pretty tough for a worker to provide a complete ranking for a list of PoIs that are located around the city. For this reason, we utilize pairwise relevance questions. A pairwise relevance question asks which of the two PoIs are more relevant to the provided query. The other studies [14, 41] assume that there exists a

total ranking amongst the objects. This assumption forces workers to provide a ranking even when they think the objects might be equally relevant or very difficult to compare. For instance, if a Chinese restaurant and an Italian restaurant are in the same neighborhood and have the same price range, then the choice would depend directly on personal preferences. In this situation, ranking one or the other higher should not make a difference. Therefore, a method to synthesize PoI rankings should not have the assumption of total ranking.

We consider a setting where q is a top-k spatial keyword query and D is the set of PoIs relevant to q. We assume that there exists a pairwise relevance relation $\prec$ on D. $x \prec y$ means that y is more relevant to the given query than x. This relation is irreflexive, asymmetric and transitive. Paper A addresses the problem of forming a pairwise relevance relation given a query q and a set of PoIs D by means of crowdsourcing.

2.2 PointRank Algorithm

We propose PointRank algorithm to build rankings for top-k spatial keyword queries. The algorithm uses answers to pairwise relevance questions from crowdsourcing workers. A pairwise relevance question is defined with a pair of PoIs and a query, and asks the workers to provide the PoI that is more relevant to the query. The algorithm iterates until whole partial ranking for the PoIs included in D is uncovered. In each iteration, it first determines the question to be processed and then processes the question.

Determining the Next Question

Normally, for n PoIs, we have $C(n,2)$ pairwise relevance questions. In order to decrease the number of questions to be asked to crowdsourcing workers, we employ a gain definition that takes transitivity of the pairwise relation into account. Given a pairwise question, the gain is defined as the number of questions that we can infer the answers for by asking the question. Let us assume that (p_i, p_j) is the pairwise relevance question under consideration. If the answer is that they are incomparable, we cannot use it to infer new pairwise relevances using transitivity. If the answer is $p_i \prec p_j$, we can infer three new pairwise relevances depending on what we already know about the pairwise relevances:

- $p_k \prec p_j$ can be inferred if we already know $p_k \prec p_i$

- $p_i \prec p_k$ can be inferred if we already know $p_j \prec p_k$

- $p_k \prec p_l$ can be inferred if we already know both $p_k \prec p_i$ and $p_j \prec p_l$

The other possible answer $p_j \prec p_i$ can also lead to some inferences of additional pairwise relations. The gain is then defined as the average of the number of possible inferences for these two possible answers. This gain definition is used to determine the next question to be processed. We compute the gain for all possible pairwise relevance questions and choose the question with maximum gain as the next question.

> **Example 2.1 (Computing Gain)**
> Let $D = \{p_1, p_2, p_3, p_4, p_5\}$ be the set of PoIs and $R = \{p_1 \prec p_2, p_1 \prec p_5, p_4 \prec p_2, p_4 \prec p_5\}$ be the set of pairwise relevances the algorithm has formed up to now. Further, the pairwise relevance question (p_2, p_3) is also processed and this pair is decided to be incomparable. Let (p_3, p_5) be the pair of PoIs that we want to compute the gain for. There are two possible answers to the pairwise relevance question regarding this pair that might lead to new inferences:
>
> 1. $p_3 \prec p_5$: This answer does not lead to any new pairwise relevance inferences since we do not have any pairwise relevances in the form of $p_i \prec p_3$ or $p_5 \prec p_j$.
>
> 2. $p_5 \prec p_3$: This answer leads to 2 new inferences. The algorithm can infer $p_1 \prec p_3$ and $p_4 \prec p_3$ since we have $p_1 \prec p_5$ and $p_4 \prec p_5$.
>
> Since gain is defined as the average, the gain of (p_3, p_5) is 1.

Processing a Question

To process a question, the PointRank algorithm follows an iterative approach. In each iteration, the pairwise relevance question is assigned to a number of crowdsourcing workers. After all the answers are collected, these answers are compared with the previous iterations' answers using the Chi-square (χ^2) test [36]. This test is a way of assessing whether the change in the answers to the question might be just due to chance. The PointRank algorithm has a p-value threshold parameter (*pvalue*) and a probability threshold parameter (*pt*). If the p-value corresponding to the χ^2 value exceeds *pvalue*, the algorithm concludes that the change is due to chance. In other words, the algorithm concludes that the workers have a consensus about the question's answer. In this case, the probability for each answer is computed. If the answer with the maximum probability has the required probability threshold then it is finalized as the answer to the question. Otherwise, the algorithm is not capable of determining the answer for this question.

PointRank proceeds with another iteration when there is no consensus between the workers. PointRank has a parameter that determines the initial

number of assignments (*ina*). Since we need at least two iterations to check for consensus, we assign the question to *ina* workers in both iterations. In the following iterations, the number of assignments is set to the total number of assignments in the previous iterations. This makes it possible to have a meaningful result from the χ^2 test. PointRank also has two parameters to control the number of iterations for each question: a minimum number of iterations parameter (*minni*) and a maximum number of iterations parameter (*maxni*). If the answer to the question is not determined after *maxni* iterations, the algorithm concludes that the workers do not agree about the pairwise relevance.

Table 1: Processing of the question (p_3, p_5)

	$p_3 \prec p_5$	$p_5 \prec p_3$	Incomparable	p-value
1st Iteration	5	1	4	
2nd Iteration	2	7	1	0.022531
3rd Iteration	4	14	2	0.154104
Total	11	22	7	
Probability	0.275	0.55	0.175	

Example 2.2 (Processing of the Question (p_3, p_5))
Continuing from Example 2.1, we process the question (p_3, p_5). Let us assume that PointRank has the following parameters: *ina* $= 10$, *minni* $= 2$, *maxni* $= 5$, *pvalue* $= 0.10$ and *pt* $= 0.5$. The answers from workers for each iteration, p-values corresponding to the χ^2 test, total number of workers for each possible answer, and final probability values for each possible answers are shown in Table 1. The question is assigned to 10 workers in the first two iterations since *ina* is set to 10. The distribution in the first iteration is $(5, 1, 4)$ and the distribution in the second iteration is $(2, 7, 1)$. Since the p-value corresponding to χ^2 test (0.022531) is less than the *pvalue* parameter, the algorithm concludes that there is no consensus between the workers and continues with the third iteration. In the third iteration, the question is assigned to 20 workers and the distribution of the answers is $(4, 14, 2)$. The p-value between third iteration and the previous iterations is 0.154104. Since it is greater than the *pvalue* parameter, the algorithm concludes that there is an agreement between the workers regarding the question. Then, the probability values for each answer is computed. The answer with maximum probability is $p_5 \prec p_3$ and its probability (0.55) is greater than the *pt* parameter. For this reason, $p_5 \prec p_3$ is chosen as the answer to the question. The algorithm infers two pairwise relevances $p_1 \prec p_3$ and $p_4 \prec p_3$ as a result of this answer.

2.3 Discussion

In order to evaluate the proposed algorithm, an experimental analysis on a simulated environment is conducted. We choose a simulation-based evaluation since it might require a large budget to do the experimental evaluation on a real crowdsourcing platform. Simulated workers are modeled with a reliability value and answer the questions according to pre-generated ground truth rankings. Reliability value is defined as the probability that the worker provides the correct answer. We compare PointRank algorithm with a baseline algorithm. The baseline algorithm assigns each question to a fixed number of workers (n) and the answer is decided using majority voting. We include baseline algorithms with $n = 40$, $n = 70$ and $n = 100$ in the experimental evaluation. We present a comparison between the pre-generated ground truth rankings and the output ranking from the algorithms. We use Kendall tau distance [17] to compute how close two rankings are. The Kendall tau distance is basically defined as the ratio of the number of pairs of PoIs that the rankings do not agree on to the total number of pairs of PoIs. So, it becomes 0 if the rankings agree on each pair and 1 if the rankings disagree on each pair. We also report the number of assignments needed by the algorithm since it is directly related to the cost of a crowdsourcing algorithm as it is required to pay a fee for each assignment completed by a crowdsourcing worker.
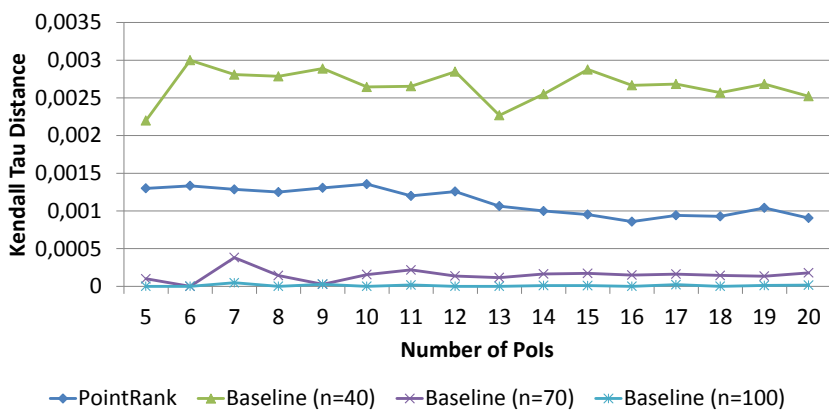
Fig. 3: Kendall Tau Distance vs Number of PoIs [25]

Figures 3 and 4 show the effect of the number of PoIs on the Kendall-tau distance and the number of assignments. The number of assignments increases as the number of PoIs increases as expected for all algorithms. The figures suggest that PointRank algorithm constructs better rankings than the
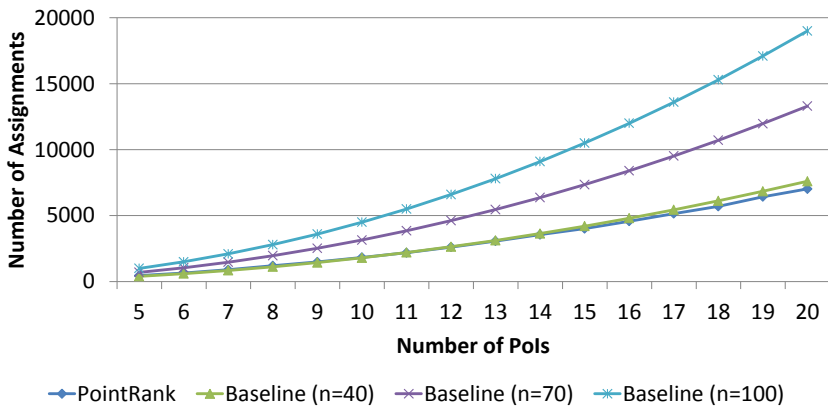
Fig. 4: Number of Assignments vs Number of PoIs [25]

baseline algorithm with $n = 40$ even though both algorithms require almost identical number of assignments. Our experimental evaluation also shows that PointRank algorithm produces better rankings than the baseline algorithm when the worker reliability is quite low. This is very important for a crowdsourcing algorithm since it is not possible to assume that all the workers are reliable in crowdsourcing platforms.

3   CrowdRankEval Framework

This section gives an overview of Paper B [24].

3.1   Problem Motivation and Statement

We propose the PointRank algorithm which addresses ranking synthesis via crowdsourcing in Paper A. The main motivation behind this algorithm is to use the rankings generated by it in ranking function evaluation for top-*k* spatial keyword queries. The underlying idea is that a ranking function that produces rankings close to the ground-truth rankings can be considered a good ranking function.

Paper B addresses building a system that makes it possible for researchers to perform this evaluation. There are three important requirements for such a system:

- Evaluation of ranking functions requires a set of queries that the researchers want to evaluate the ranking functions on and a set of rankings obtained by applying the ranking functions on the input queries.

For this reason, such a system should have interfaces to make it possible for the users to provide these inputs.

- In order to construct the ground-truth rankings, such a system should provide an implementation of the PointRank algorithm on top of a crowdsourcing platform. So, it is required to communicate with the crowdsourcing platform to be able to publish tasks and get answers from the workers.

- The users should be able to visualize the output rankings and the evaluation results.

3.2 Use Case and Workflow

We demonstrate the CrowdRankEval framework as a complete ranking function evaluation framework for top-*k* spatial keyword queries. It is intended to address two use cases. The first use case consists of determining the best parameter configuration for an existing ranking function and finding out whether a new ranking function performs better than the existing ones for a set of queries. The second use case is checking various hypotheses to understand the relationship between ranking functions and keywords or regions. For instance, one hypothesis may be "The expensiveness level of the PoI is more important than the spatial proximity of the PoI to the query location if the keyword is related to expensive items such as furniture and mobile phones. On the other hand, the expensiveness level is less important than the spatial proximity if the keyword refers to a daily food product such as burger and pizza.".
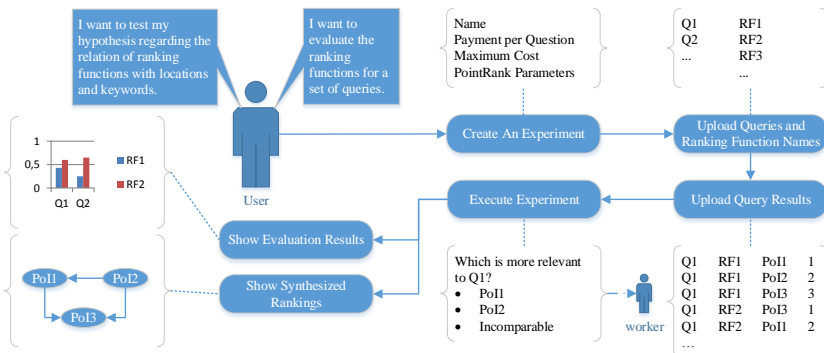
Fig. 5: Workflow of CrowdRankEval [24]

14

The core unit of CrowdRankEval is termed an experiment. An experiment consists of a set of queries, output rankings for these queries by the ranking functions to be evaluated, an experiment name, crowdsourcing parameters such as payment per question and maximum cost, and PointRank parameters. The workflow of the framework is displayed in Figure 5. The user is expected to create an experiment with the required parameters. Then, queries and ranking functions should be uploaded to the framework. Then, query results obtained by applying the input ranking functions for the input queries should be uploaded. The next step is executing the experiment. This basically means building rankings for the input queries via crowdsourcing using the PointRank algorithm. After the rankings are built, the user can display the output rankings and display the evaluation results of the input ranking functions.

3.3 Framework

The CrowdRankEval framework consists of four modules to address the aforementioned requirements. User interface module handles the graphical user interface of the framework. This includes providing interfaces for the users to upload data, to display the status of their experiments and to illustrate the output rankings and evaluation results for their experiments. Data preparation module takes care of necessary checks to make sure that the uploaded data is correct and does not have any missing information. It also forms the set of PoIs for each query from the query results uploaded. PointRank module contains an implementation of the PointRank algorithm on top of CrowdFlower. This module takes care of the communication between CrowdRankEval and CrowdFlower. Finally, evaluation module performs the evaluation of the ranking functions by comparing the input rankings for each query to the ground-truth ranking obtained from the PointRank module for the query. We utilize Kendall tau distance for ranking comparison. It presents the evaluation results on different levels to make it possible for the users to understand which ranking function performs better for each query, for each location, for each keyword, and in general.

Example 3.1 (Evaluation on CrowdRankEval)
Figures 6 and 7 show an example evaluation of an experiment conducted on CrowdRankEval. This experiment consists of 2 queries with keywords "bar" and "bibliotek" (means library in Danish). Both queries have a location in the central part of Aalborg. Figure 6 illustrates the ranking obtained for the query with "bar" keyword. The ranking is shown as a graph consisting of the PoIs and a directed edge (p_i, p_j) denotes that p_j is more relevant to the input query than p_i. Incomparable PoIs are shown with a dashed undirected edge between them. If the PointRank algorithm is not

Fig. 6: Visualization of Synthesized Ranking on CrowdRankEval [24]
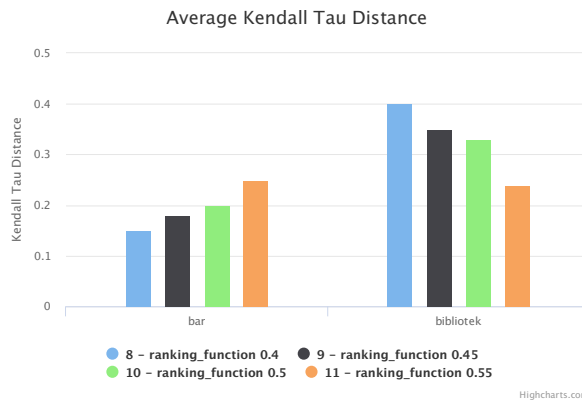
Fig. 7: Evaluation Results on CrowdRankEval

able to find the answer for a pair of PoIs, then there is no edge between them. In this example, we have no such pairs. Figure 7 illustrates the evaluation results. This figure illustrates the Kendall tau distance between the input rankings and the synthesized ranking. It can be seen that the ranking function that performs best for the keyword "bar" performs worst for the keyword "bibliotek".

16

3.4 Discussion

Paper B provides a demonstration of a new ranking function evaluation framework. We consider top-*k* spatial keyword queries in the framework, but it is also applicable for other top-*k* query types. The drawback of the framework is that it does not test the knowledge of the crowdsourcing workers prior to task assignment. This can be addressed as a future work since most of the platforms allow task requester to place some tests for workers before they are eligible for the tasks. Another future work direction would be to integrate ways of handling unreliable and/or malicious workers to the PointRank algorithm in order to get better results.

4 Crowdsourcing-based Evaluation of Ranking Approaches

This section gives an overview of Paper C [13].

4.1 Problem Motivation and Statement

Paper A proposes the PointRank algorithm to synthesize ground-truth rankings for a given set of PoIs and Paper B demonstrates a complete framework for ranking function evaluation. However, if there are only a limited number of ranking functions to evaluate, constructing a ground-truth ranking might not be necessary. For instance, if we want to evaluate two ranking functions for a query and a PoI is not contained in both of the rankings produced by these functions, it is not needed to know what the exact position of this object is in the ground-truth ranking. Moreover, there is no need to ask about two PoIs, if both rankings agree about the ranking of these PoIs. Another shortcoming of PointRank is that it does not take the budget constraint into account. To synthesize rankings for M queries, PointRank creates $M \cdot C(n, 2) \cdot 2^{maxni-1} \cdot ina$ assignments in the worst case assuming a set of n PoIs is provided for each query. It might require a large budget to complete these assignments for large values of M and n. Therefore, a framework that considers budget constraints for ranking evaluation of top-*k* spatial keyword queries is needed.

Paper C tackles the problem of evaluation of ranking approaches for spatial keyword querying on a given set of PoIs under a budget constraint. More specifically, Paper C focuses on having two input ranking functions. However, the proposed methodology can be applied when there are more ranking functions under consideration.

17

4.2 Solution Overview

The proposed framework takes two ranking functions f_1 and f_2, a set of PoIs, and a set of M queries and finds out which of the ranking functions produces better rankings for the given set of queries. Our framework employs pairwise relevance questions as in PointRank algorithm. The complete framework is shown in Figure 8. Our framework is composed of M local evaluations each of which corresponds to an input query and a global evaluation that decides on the better ranking function with respect to the results of the local evaluations.

A local evaluation consists of three elements. The first element is the ranking process. It simply computes top-k rankings l_1 and l_2 for the input query from the input set of PoIs by applying the functions f_1 and f_2, respectively.

The second element is the matrix based question model. The main task of this element is to decrease the number of pairwise relevance questions and to determine the most important questions in order to comply with the budget constraint. We employ a learn-to-rank method to form a top-k ranking l_{12} to cover the important features of both rankings. This element requires a set of training queries. We first form two rankings corresponding to f_1 and f_2 for each training query. Then, a score for each object included in these lists is computed with respect to its rank in both lists. Objects together with their scores form the training data. After training a ranking function, we use the ranking function to generate scores for the PoIs included in $l_1 \cup l_2$ and the first k elements of this set constitute l_{12}. Before generating the questions, the matrix based question model eliminates the questions that both l_1 and l_2 agree on the answer. We utilize an entropy definition to decide whether we should ask the question to the crowdsourcing workers. Entropy of an object is defined with respect to the representativeness of the keywords contained in its documents. If the entropy difference between a pair of objects is less than a threshold, a question regarding this pair is added to the set of generated questions. The algorithm to generate questions stops when the size of this set reaches to the given budget constraint. The algorithm first checks the pairs of objects included in l_{12}. If we still have remaining budget, $l_c = l_1 \cap l_2$ is considered. If there is still space for more questions, then we check the remaining objects in the set $l_r = l_1 \cup l_2 \setminus l_c$.

Crowdsourcing based evaluation is the third element of the local evaluation. This element first collects answers for the generated questions. We utilize three different methods to determine the answer for a pairwise relevance question. The first method is majority voting. The answer given by the majority of the workers is selected as the answer to the question. The second method is voting based on constant confidence. The confidence value for a worker is the same for all questions in this method. The third method is voting based on dynamic confidence. A worker can specify a confidence value

Local Evaluation (one query)

Ranking Process

Query q_i → Data → f_1 → l_1

f_2 → l_2

Matrix Based Question Model

Cost Budget

Training Queries

Binary questions → Question Generator ← l_1, l_2, l_{12} ← Learn to Rank

Crowdsourcing Based Evaluation

Crowdsourcing Platform

Answers in partial matrix → Matrix Factorization → *Complete matrix* → Counting → l_1, l_2, l_f

Global Evaluation (*M* queries)

Score Aggregator ← *Score f_1 or f_2* ← Ranking List Comparator ← *For q_1*

...

← *Score f_1 or f_2* ← *For q_i*

Final Conclusion ← Ranking Function Scorer

← *Score f_1 or f_2* ← *For q_M*

...

Fig. 8: System Framework [13]

for a specific question in this method. We store the answers in a $n \times n$ matrix where n is the number of objects. However, we have a partial matrix after collecting the answers from crowdsourcing platform due to the budget constraints. Then, we apply non-negative matrix factorization to fill the missing values of the matrix. The last step of this element is forming the final ranking for the input query. We utilize Borda-count [5] to form the ranking.

> **Example 4.1 (Ranking with Borda-Counts)**
> Let $O = \{p_1, p_2, p_3, p_4, p_5\}$ and we have pairwise relevances $\{p_1 \prec p_2,$
> $p_1 \prec p_3, p_1 \prec p_4, p_1 \prec p_5, p_2 \prec p_4, p_3 \prec p_2, p_3 \prec p_4, p_3 \prec p_5, p_5 \prec p_2,$
> $p_5 \prec p_4\}$. The Borda counts for the PoIs are as follows: $g(p_1) = 0$, $g(p_2) = 3$, $g(p_3) = 1$, $g(p_4) = 4$, $g(p_5) = 2$. So the final ranking is $\langle p_4, p_2, p_5, p_3, p_1 \rangle$.

Global evaluation takes the rankings produced by ranking functions f_1 and f_2 (l_1 and l_2) and the final ranking (l_f) for each query q_i. Then, global evaluation component compares l_1 and l_2 with l_f using Kendall tau distance [17] for q_i. If l_1 has a lower Kendall tau distance than l_2, it is considered as a vote for f_1. Otherwise, it is a vote for f_2. The global evaluation component concludes that the ranking function with the majority of the votes is a better ranking function.

4.3   Discussion

In order to evaluate the proposed framework, we focus on two ranking functions. These functions (f_1 and f_2) are obtained by changing the weighting parameter in a ranking function widely used in the literature for spatial keyword querying. Then, we obtain a vector containing the number of votes for f_1 for a set of queries using the maximum budget. Then, we compare the number of votes for f_1 obtained by our framework for different parameter settings with the vector obtained with the maximum budget using Cosine similarity.

Figures 9a and 9b show how our framework is affected by the budget constraint and the matrix factorization. Figure 9a illustrates that our algorithm is sensitive to the budget constraint. In other words, it is clear that the framework provides better results when the available budget increases. Figure 9b shows the effect of matrix factorization on our framework. The results provide clear evidence that we obtain better results by utilizing matrix factorization. The experimental evaluation also suggests that voting based on dynamic confidence performs better than majority voting and voting based on constant confidence. This is expected since workers might not provide the same confidence for every question they answer. For instance, a worker
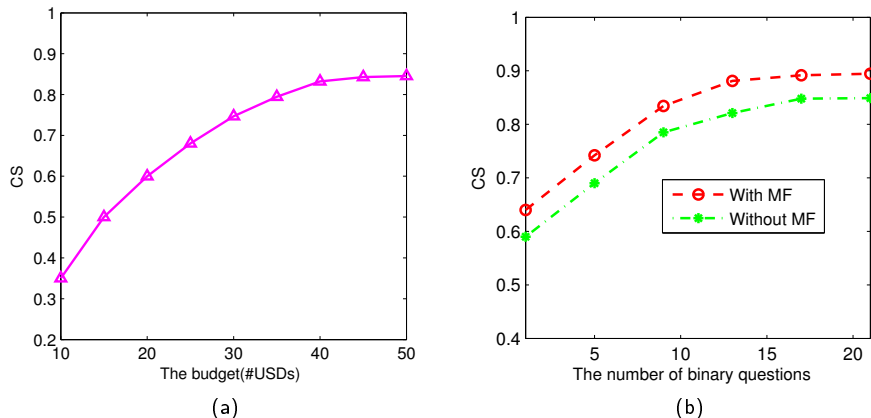
Fig. 9: Effects of Budget and Matrix Factorization [13]

might be more confident about answering about a pair of PoIs that he regularly visits than a pair of PoIs he has only heard of.

5 GPS-based Ranking Synthesis

This section gives an overview of Paper D [23].

5.1 Problem Motivation and Statement

Paper A and Paper C focus on constructing rankings for spatial keyword queries by means of crowdsourcing. However, crowdsourcing-based methods have some drawbacks. First, they are quite costly since one has to pay for each task completed by a crowdsourcing worker. Second, it requires extremely long time since one has to wait until all of the tasks are completed by crowdsourcing workers. Third, to answer the pairwise questions generated by the algorithms we propose, workers need to be knowledgeable about the geographical region that the questions cover since we focus on spatial relevance. Moreover, even though they are familiar with the region, they might not have information about the PoIs included in a question. For these reasons, Paper D focuses on utilizing GPS data instead of crowdsourcing.

Paper D utilizes GPS records collected from vehicles. The following definition is reproduced from [23].

Definition 0.1. *A GPS record G is a four-tuple $\langle u, t, loc, im \rangle$, where u is the ID of a user, t is a timestamp, loc is a pair of Euclidean coordinates representing the location, and im is the vehicle ignition mode.*

Paper D addresses the problem of synthesizing a ranking for a given top-*k* spatial keyword query using a set of GPS records S_G and a set of PoIs S_P.

5.2 Method Overview

The main assumption behind the proposed method is that users first search for PoIs and then decide on which PoI to visit. The underlying spatial region is modeled as a regular grid. Each trip to a PoI is considered as a vote for the PoI for the grid cell containing the source location of the trip.

Our method has two phases: model-building and ranking-building. The output of model-building phase is a regular grid that contains two values for each cell and for each PoI regarding the trips starting from the cell to the PoI. The first value is the number of the trips and the second value is the number of users making these trips. In order to build the model, it first extracts the set of trips to PoIs from S_G. Then, it sets these two values with respect to the set of trips. A cell might have missing values for some PoIs due to the lack of trips from the cell to these PoIs. In order to address this problem, model-building phase includes a smoothing method based on PageRank algorithm [33]. Model-building phase is explained in detail in Section 5.3.

Ranking-building phase constructs a top-*k* ranking for a given top-*k* spatial keyword query from the output model. First, the grid cell is identified by the query location. Then, the PoIs relevant to the query keywords are ranked according to a score computed by a weighted sum of the number of trips and the number of users values with respect to a weighting parameter (β).

5.3 Model-Building Phase

The model-building phase utilizes historical trips. In order to extract trips to PoIs, we first need to find users' stops. This is quite straightforward since we have the ignition mode attribute. If the engine of the vehicle is off for more than a threshold parameter, we consider that the driver has made a stop to visit a PoI. Then, we need to eliminate the visits to home and work locations, since they do not provide any value to capture users' preferences for PoIs. People generally spend a considerably long time at their home and work locations. They also visit these locations more frequently than any other PoIs. Based on these observations, we utilize a density-based clustering approach in order to detect the stops corresponding to these locations. We basically cluster a user's stops and check the frequency of stops and the average time the user spends in each cluster. If both of these values in a cluster are above the threshold parameters, we conclude that the cluster corresponds to the user's home or work location.

The model-building phase proceeds with the assignment of stops to PoIs

after excluding stops for home and work locations. An assignment of a stop s to a PoI p means that the user made the stop s to visit p. Two methods are proposed for the stop assignment problem: Distance based assignment (DBA) and temporal pattern enhanced assignment (TPEA). DBA assigns the stop to the closest PoI if there are less than a number of PoIs within a distance threshold from the stop. However, DBA cannot assign stops that have too many or no PoIs around them. For these stops, we propose TPEA method that is built on density based clustering. It uses the information provided by the user's assigned stops in the output of DBA. The TPEA method first clusters the user's stops. For each cluster, the user's temporal visit patterns are extracted from the assigned stops within the cluster. These patterns are employed to identify the PoI a user visited as a result of an unassigned stop. The core idea is that if a user has only visited p_i on a specific time period and if the user has an unassigned stop s_j that is near p_i during the same time of day, it is highly probable that s_j also corresponds to p_i. An example temporal pattern would be visiting a kindergarten to pick up kids around 16:00 on weekdays. The set of trips to PoIs is formed with respect to the assigned stops.
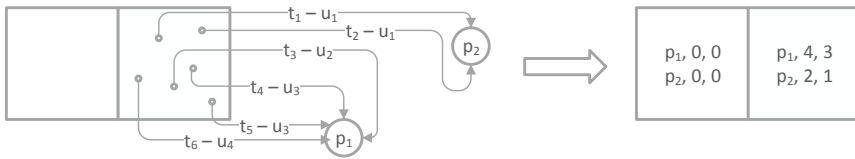
Fig. 10: Example Initialization of Grid Cells

Using the set of trips, the number of trips and the number of unique users are computed for each grid cell. An example is illustrated in Figure 10. There are 2 grid cells and 2 PoIs in this example. A trip is denoted by a pair (t_i, u_j) where t_i is the identifier of the trip and u_j is the identifier of the user. The set of trips contains 6 trips completed by 4 users. The initialized grid is shown on the right.

As can be seen in Figure 10, some cells might have missing values since they do not have any trips starting within the cell. This might be due to the lack of users from this cell in the GPS dataset. It might also be due to not being able to identify the visited PoIs for some stops due to high PoI density around them. This is an issue for the model since it limits the geographical coverage. If the input query corresponds to a cell with missing values, our model is not able to provide a ranking. To address this issue, we propose a personalized PageRank-based algorithm to smooth the grid values. The main intuition is that PoIs visited by the users starting from nearby cells

may be possible targets for the users located in the cell with missing values. PageRank algorithm [8] is proposed for ranking web pages for search engine queries. It takes incoming links to a web page and outgoing links from a web page into account to determine a score for a web page. If a web page is linked from important web pages, it is considered important, so it should be ranked higher. The algorithm is basically a random walk over the graph of web pages. It starts with a random vertex and it follows either an outgoing edge or continues with another randomly selected vertex. The probability for a vertex to be randomly chosen is the same for all vertices in the PageRank algorithm. Personalized PageRank algorithm [33] determines this probability with respect to the personal preferences.

In order to apply personalized PageRank algorithm for smoothing, we first build a directed graph from the underlying grid. We introduce an edge between each pair of neighbor cells with a weight inversely proportional to the distance between them. The personalization parameter is set according to the initial cell values. PageRank algorithm outputs a page rank value for each vertex. The page rank value of a vertex is the probability that the random walker will visit the vertex. After obtaining the page rank values, the total number of users and the total number of trips are distributed with respect to these values. For instance, let us assume that we are smoothing the number of users and the total number of users is 50. The number of users value for a grid cell with an output page rank value 0.3 is set to 15 after smoothing.

5.4   Discussion

To evaluate our proposed method, we utilize a GPS dataset consisting of around 0.4 billion GPS records collected from 354 vehicles traveling around Nordjylland, Denmark for a period of 9 months in 2014. The PoI dataset is collected from Google Places API and it has $10,000$ PoIs in or around Aalborg. There are 88 PoI categories in the dataset.

We first build a set of ground-truth assignments using home/work stops identified in the GPS dataset so as to evaluate our stop assignment methods. For each home/work cluster, the center of the cluster is added to the PoI dataset as a home/work PoI. The ground-truth data then consists of the home/work stops assigned to the corresponding home/work PoIs. We use closest assignment (CA) method as a baseline. It simply assigns each stop to the closest PoI. We present precision and recall values for CA, DBA and TPEA methods. We consider home/works stops assigned to the correct home/work PoIs as true positives. We consider stops that are not identified as home/work stops and are assigned to home/work PoIs as false positives. Unassigned home/work stops and home/work stops that are assigned to incorrect PoIs are considered as false negatives.
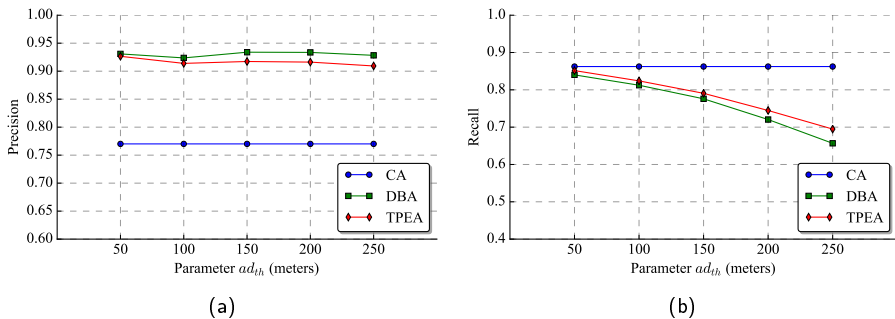
Fig. 11: Precision and Recall [23]

Figure 11 illustrates precision and recall values obtained by CA, DBA and TPEA methods. Figure 11a shows that both DBA and TPEA methods have higher precision than the CA method. TPEA has a slightly lower precision than DBA due to the false positive assignments obtained by using temporal visit patterns. Figure 11b shows that CA has higher recall than the DBA and TPEA methods. This is due to the fact that we consider unassigned home/work stops as false negatives.

We also conduct experiments in order to assess the effect of smoothing on the proposed method. As aforementioned, the proposed method utilizes a personalized Page-Rank based smoothing method and this might introduce some distortion in the initial values of the grid cells. To check the extent of the distortion, we build top-10 rankings for the grid cells with enough data before and after smoothing. We utilize Kendall tau distance [17] to determine the similarity of these rankings and we present the distribution of the Kendall tau distance values between them.

Figure 12 shows the box plot of Kendall-tau distance values between top-10 rankings extracted before and after smoothing for different weighting parameter (β) values. Green lines denote the means and red lines denote the medians of the distance values. The average Kendall tau distance between the rankings is around 0.15. This means that 85% of the existing pairwise relations between PoIs are preserved after smoothing. It is also possible to see that Kendall tau distance does not exceed 0.1 for 50% of the grid cells. The experimental evaluation results suggest that the proposed smoothing method preserves the characteristics of original data while enlarging the geographical area covered.

In addition to the experiments mentioned, we conduct a set of experiments to evaluate the effect of weighting parameter on the output ranking. The results show that the proposed method is largely insensitive to the weighting parameter.

25
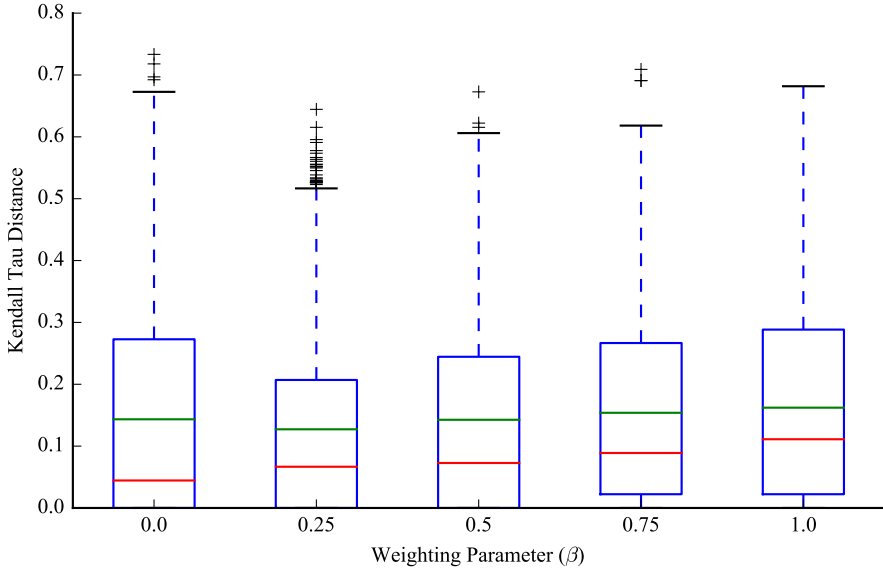
Fig. 12: Kendall Tau Distance Distribution [23]

6 Extracting Visits to PoIs from GPS data

This section gives an overview of Paper E [26].

6.1 Problem Motivation and Statement

Paper D proposes a method to construct rankings for spatial keyword queries using GPS data obtained from vehicles. An important step in this method is the stop assignment to PoIs due to the fact that these assignments form a foundation for the grid-based model to rank the PoIs. Both DBA and TPEA methods are not able to assign any stops to a PoI if it is surrounded by other PoIs. TPEA utilizes temporal visit patterns of the users to improve on DBA. However, since it first utilizes DBA to find these patterns, a PoI that does not have any stops assigned to will not have any patterns associated with it. It is important to be able to detect the visits to PoIs in highly dense areas since it will make it possible to form rankings that reflect the users' preferences better.

Given a set of GPS records, Paper E addresses the problem of finding users' visits to PoIs. This problem statement covers two problems: Detecting the stops in the set, and stop assignment to PoIs. Paper E utilizes the stop extraction method used in Paper D and focuses on the stop assignment to PoIs.

6.2 Solution Overview

We propose a method to assign stops to PoIs. The proposed method utilizes a Bayesian network together with distance based filtering to assign a stop. The Bayesian Network models the relation between the temporal features of a stop and a PoI category. Distance based filtering is used to find the set of candidate PoIs just like DBA. To assign a stop, the method first finds the candidate PoI categories within a distance threshold (ad_{th}) of the stop. Then, the probability of visiting a PoI of this category is computed using the Bayesian network for each category. If we have a single candidate PoI of the category with maximum probability, the stop is assigned to this PoI.
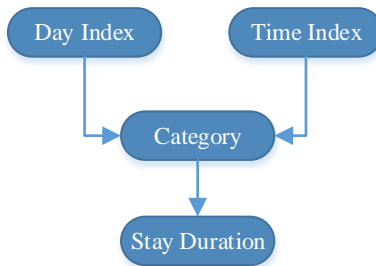
Fig. 13: Structure of the Bayesian Network [26]

Figure 13 shows the structure of the Bayesian network. Day index, time index and stay duration are the attributes of a stop and we are interested in finding the probability of visiting a category for given values of these attributes. This structure simply means that PoI category depends on the day index and the time index, and that stay duration depends on the category. We divide the time into equal intervals and time index is the identifier for the time period that the stop happened. To illustrate, if the time period is 2 hours and the stop time is 17:23, the time index becomes 8.

The structure of the Bayesian network is decided after preliminary analysis of a labeled dataset constructed with DBA with a conservative setting. We only assign a stop to a PoI if there is only one PoI within 100 meters of the stop. Our preliminary analysis shows that people have a tendency to visit some categories on specific days and on specific time intervals. For instance, our preliminary analysis on the labeled data suggests that people visit bakeries in the morning and pubs at night. For this reason, we have edges from day index and time index nodes to PoI category node in the Bayesian network. Figure 14 shows the stay duration distributions for some PoI categories. These distributions provide evidence on the effect of PoI category
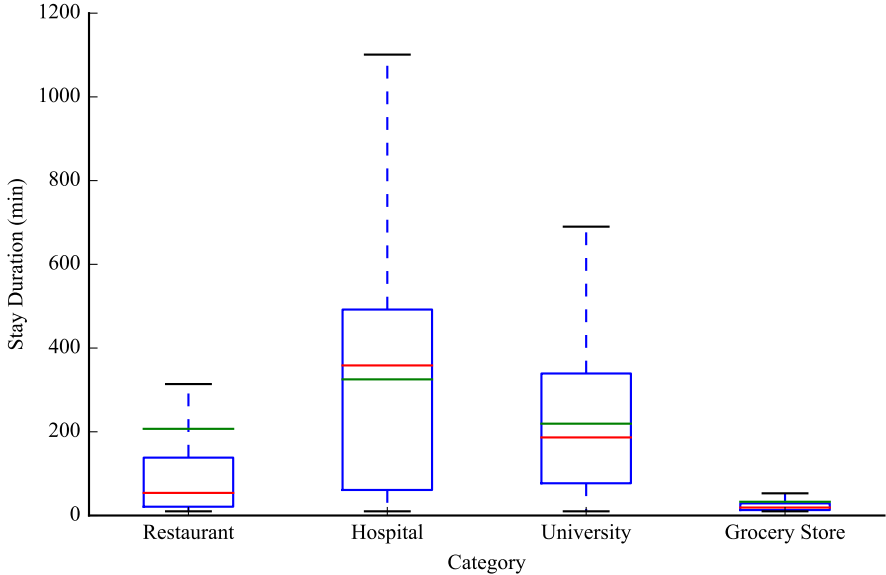
Fig. 14: Stay Duration Distribution for Different Categories [26]

on the stay duration. For instance, it is clear that people spend more time in restaurants than grocery stores. Hence, we have an edge from PoI category node to stay duration node.

In order to learn the Bayesian network, we use the same labeled dataset. Learning a Bayesian network corresponds to building the conditional probability tables for each node.

6.3 Discussion

We use the same GPS dataset and PoI dataset as used in Paper D to evaluate the proposed method. The cardinality of the labeled dataset obtained with the distance threshold of 100 meters is 36,691. We employ 10-fold cross validation to determine training and test sets. To make it certain that we have more than one PoI option for a test stop, we introduce two parameters: a distance factor parameter df, and a minimum PoI count parameter mpc. If a stop does not have at least mpc PoIs within $ad_{th} \cdot df$ meters, it is not added to the test set. The default values for df and mpc are 2 and 3, respectively. We also change our algorithm to return a list of possible categories ordered with respect to their joint probability computed by the Bayesian network. We report precision at position n ($p@n$), mean reciprocal rank (mrr) and number of possible categories (npc). $p@n$ corresponds to the ratio of the stops that have their correct PoI category in first n positions in the output list. mrr is

the average position of the correct category in the output list and *npc* is the average number of candidate PoI categories.

(a) on *p@n*

(b) on *mrr* and *npc*
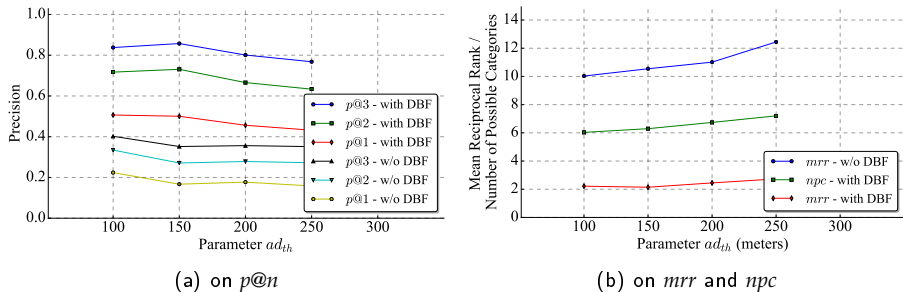
Fig. 15: Effect of ad_{th} [26]

Figures 15a and 15b show the effect of distance threshold on the proposed method. Figure 15a demonstrates that precision decreases as ad_{th} increases. This is expected since the set of candidate categories also gets larger as ad_{th} increases as illustrated in Figure 15b. The figures also show that the proposed method is capable of capturing temporal characteristics of the visits to PoIs with different categories since it achieves a *p@3* value around 0.8 and an *mrr* value of 2 out of 6 possible categories. Finally, it is also possible to see that distance based filtering improves the proposed method significantly. Figure 15b demonstrates that when we do not use distance based filtering, the method has an *mrr* value of 10–12 out of 88 possible categories. This result suggests that some PoI categories have similar temporal characteristics and distance based filtering helps us to filter out categories with similar characteristics.

7 Querying Point of Interest Clusters

This section gives an overview of Paper F.

7.1 Problem Motivation and Statement

Spatial keyword query proposals in the literature mostly target individual PoIs. However, users may be interested in a group of similar and nearby PoIs instead of a single PoI in some cases. To illustrate, a user who wants to purchase a laptop might visit many electronic stores to find the best option. There are two query proposals for such users: Top-*k* groups query [37] and top-*k* spatial textual cluster (*k*-STC) query [39]. Both of these queries identify the best *k* groups with respect to a cost definition. Top-*k* groups

query considers all possible groups of relevant PoIs while *k*-STC query only considers density-based clusters [16] of relevant PoIs. The cost definitions of these query types do not take the user's transportation mode into account. We believe that a user's transportation mode affects their preferences. For instance, a user with a car would be willing to travel more than a user who wants to walk to reach to a PoI. Another drawback of *k*-STC queries is that users are expected to provide density-based clustering parameters. It may be challenging for a user to provide meaningful parameters since this requires knowledge about the query region. Paper F extends the *k*-STC query by removing the density-based clustering parameters and adding parameters regarding transportation modes of a user, and formalizes the *k*-TMSTC query as follows:

Definition 0.2. *A **top-k transportation-mode aware spatial textual clusters (k-TMSTC)** query is defined as $q = \langle \lambda, \psi, k, tm_c, tm_i \rangle$, where λ is a point location, ψ is a set of keywords, k is the number of result clusters, and tm_c and tm_i are transportation modes for traveling to the cluster and traveling in the cluster, respectively. A cluster is defined as a subset of the set of relevant PoIs and a k-TMSTC query identifies the k best maximal density-based clusters with respect to a cost function. The transportation mode can be one of the following: driving, cycling, walking, and public transportation.*

A density-based cluster on a set of objects is defined with respect to two parameters: ϵ and *minpts*. Before defining a density-based cluster, we need to provide relevant definitions from density-based clustering. A core object is an object that has at least *minpts* objects within its ϵ-neighborhood. In other words, for a core object, there should be at least *minpts* objects whose distance to the object is less than ϵ. An object o_i is said to be directly reachable from o_j if o_j is a core object and o_i is in o_j's ϵ-neighborhood. If there is a chain of objects $o_1,...,o_k$, where $o_1 = o_i$, $o_k = o_j$ and o_m is directly reachable from o_{m+1} for $1 \leq m < k$, we say that o_i is reachable from o_j. A maximal density-based cluster is formed by a core object and all reachable objects from the core object. *k*-TMSTC query identifies top-*k* density-based clusters on the set of relevant PoIs with respect to a cost function.

There are two cost functions that can be utilized for *k*-TMSTC queries: a spatial cost function and a spatio-textual cost function. Both cost functions consider the travel duration which is the total duration that the user needs to travel to visit all PoIs in the cluster. The spatial cost function is defined as the ratio of this duration to the cardinality of the cluster. The spatio-textual cost function is defined as the ratio of this duration to the average textual relevance of the PoIs in the cluster.

Paper F addresses the problem of developing an efficient algorithm to process *k*-TMSTC queries with a response time that supports interactive search. Since, ϵ and *minpts* parameters are required to define a density-based cluster,

an algorithm to process k-TMSTC queries should also include a method to determine these parameters.

7.2 DBSCAN-based Algorithm

We propose an algorithm based on DBSCAN to process k-TMSTC queries. The algorithm assumes that upper bounds for the ϵ parameter (ϵ_{ub}) and the distance between the query location and a PoI (Δ_{ub}) are available. It also assumes that a lower bound and an upper bound for *minpts* parameter are available. The main idea behind the algorithm is to reuse the existing proposals for k-STC queries [39] while processing k-TMSTC queries. In order to achieve that, we need to determine the density based parameters and form k-STC queries corresponding to these parameters. The algorithm first obtains the set of relevant PoIs D_ψ by issuing a range query centered at the query location with a range of Δ_{ub}. This range query also considers the textual relevance of the PoIs to the query keywords. In other words, if a PoI is not relevant to the query keywords, it is not included in the output set. Then, the algorithm determines possible ϵ values for each *minpts* value in parallel. The algorithm employs an approach used in VDBSCAN algorithm [28]. This approach first finds the distances between objects and their k^{th} nearest neighbors where k is set to *minpts*. The distance values are then sorted to find out different density levels. The *minpts* value together with the ϵ values corresponding to these density levels are selected as the density-based clustering parameters.

After determining pairs of density-based clustering parameters, a k-STC query is formed for each pair. Then, the algorithm processes these k-STC queries in parallel. To process a k-STC query, the algorithm iterates over the set of relevant PoIs D_ψ. In each iteration, the algorithm issues a range query centered at the current object's location with a range of ϵ value and determines whether the current object is a core object. If so, the cluster corresponding to the current object is formed and is added to the output. The output clusters obtained by processing all k-STC queries are then ordered with respect to their cost and k clusters with the lowest cost values constitute the output of the given k-TMSTC query.

DBSCAN-based algorithm has two versions. The first version uses an IR-tree to find k^{th} nearest neighbor distances and to process range queries. IR-tree is a hybrid index structure based on R-tree [21] and inverted files [45]. An inverted file has two elements. The first element is a vocabulary consisting of distinct words in the textual descriptions of the indexed dataset. The second element is a posting list for each word. An item in the posting list contains an object identifier and the weight of the word in the object. IR-tree extends R-tree with inverted files to make it possible to check for textual relevance. The second version uses a grid-based index structure named

SGPL [39] to process range queries. It first creates an $n \times n$ grid on the dataset, and the grid cells are indexed by a space filling curve. Then, an SGPL is formed for each word w. The SGPL of w is a sorted list of entries of the form $\langle c, S_{w,c} \rangle$, where c is the index value of a cell and $S_{w,c}$ is a set of objects that contain w in their textual definition and are located in cell c.

We also propose an algorithm named FastKDist to find the list of kth nearest neighbor distances in a given set of objects using SGPL. The idea is borrowed from by Wu and Tan [40], who propose an algorithm to process k-nearest neighbor queries on a grid-based index. They build a visit order for the grid cells to efficiently process these queries. This ensures a minimal number of visited cells. We use the same method on SGPL to be able to find k^{th} nearest neighbor distances taking also the textual relevance into account.

7.3 OPTICS-based Algorithm

Given n different *minpts* values and an average of m ϵ values for each *minpts* value, DBSCAN-based algorithm has to process $n \times m$ k-STC queries in parallel. OPTICS-based algorithm reduces the number of queries that need to be processed to n by utilizing the OPTICS algorithm [3]. OPTICS algorithm takes a *minpts* value and a generating ϵ (ϵ_g) value as parameters and outputs a density-based cluster order. A cluster order is an ordered list of objects together with their core and reachability distance values. The core distance of an object is the minimum ϵ value that makes the object a core object. In other words, it is the distance of the kth nearest neighbor to the object when k is set to *minpts*. Reachability distance of an object is the maximum ϵ value that makes the object directly reachable from one of the core objects with respect to ϵ_g. The cluster order is then used to extract clusters for any ϵ value that is less than ϵ_g. The algorithm iterates over the cluster order and checks the reachability distance and core distance values of the current object. If its reachability distance does not exceed ϵ, this means that the current object is reachable from the current cluster. So, it is added to the cluster. Otherwise, the algorithm checks its core distance. If it does not exceed ϵ, a new cluster is initialized with the current object.

Another advantage of OPTICS-based algorithm is that it is possible to determine density-based clustering parameters while constructing the cluster order. So, we do not need a separate phase to determine these parameters unlike the DBSCAN-based algorithm.

OPTICS-based algorithm starts with obtaining the set of relevant PoIs. Then, the algorithm processes the k-TMSTC query for each possible *minpts* value in parallel. To do that it utilizes an updated version of OPTICS that returns ϵ values together with the cluster order. Normally, OPTICS algorithm issues a range query to find ϵ_g-neighborhood of each object. The updated version issues a range query together with k^{th} nearest neighbor query with

$k = minpts$ and returns the distance value together with the ϵ_g neighborhood. These distance values obtained after processing all objects are used just like in the DBSCAN-based algorithm to determine the ϵ values. Then, the clusters corresponding to these ϵ values are extracted from the cluster order. These output clusters are merged after parallel execution. The clusters are then ordered with respect to their cost and k clusters with the lowest cost values form the output of the k-TMSTC query. The OPTICS-based algorithm also has two versions: one based on IR-tree and one based on SGPL.

7.4 Discussion

To evaluate the proposed algorithms, we use a real dataset from Yelp that contains around $157,000$ PoIs. We use 10 randomly chosen keywords from the top-50 PoI categories in the dataset. Query locations are chosen from the center regions of the 10 cities with the highest PoI density. The set of queries consist of k-TMSTC queries created for each location and for each keyword with $k = 6$. So, the cardinality of this set is 100. We use a fixed k value since it does not affect the query processing performance of the algorithms. We evaluate the performance of four versions of the algorithms: the DBSCAN-based algorithm on an IR-tree (v1), the DBSCAN-based algorithm on SGPL (v2), the OPTICS-based algorithm on an IR-tree (v3), and the OPTICS-based algorithm on SGPL (v4).

(a)   (b)
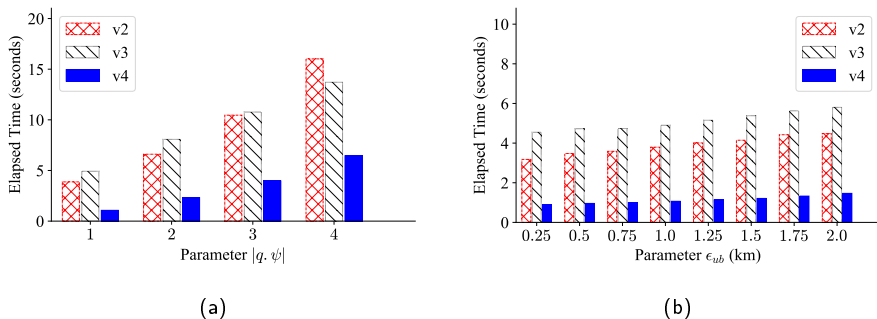
Fig. 16: Effects of Parameters $|q.\psi|$ and ϵ_{ub}

Figure 16a shows the effect of the number of query keywords on the performance of the algorithms. We do not report the elapsed time for v1 since it performs significantly worse than its competitors. The performance decreases as the number of query keywords increases, as expected. The number of relevant objects increases when the number of query keywords increases. This results in an increase in the number of range queries issued; thus, the query performance decreases. Figure 16b illustrates the effect of the upper

bound for ϵ (ϵ_{ub}) on the performance of the algorithms. The performance of the algorithms becomes worse as ϵ_{ub} increases. The reason for the decrease in the performance is that a larger ϵ_{ub} value means more k-STC queries for the DBSCAN-based algorithm, and a larger area for range queries for the OPTICS-based algorithm.

Overall, the experimental evaluation suggests that the OPTICS-based algorithm on the SGPL (v4) outperforms the other algorithms. The evaluation also shows that our proposed algorithms are able to provide a response time that supports interactive search.

8   Summary of Contributions

The thesis provides methods of ranking function evaluation for spatial keyword queries using crowdsourcing and historical trajectory data. The thesis also introduces a spatial keyword query type targeting PoI clusters. To sum up, the papers in the thesis make the following contributions.

- Paper A [25] proposes the PointRank algorithm to form partial rankings of PoIs for a given top-k spatial keyword query. The algorithm is based on the knowledge of crowdsourcing workers regarding the query and the PoIs. Moreover, it also provides a gain definition to reduce the number of questions needed by the algorithm. The paper also introduces a new evaluation methodology based on simulation of crowdsourcing workers. The experimental evaluation suggests that the algorithm is able to provide better results than a baseline algorithm with the same cost.

- Paper B [24] demonstrates a complete system based on the PointRank algorithm to evaluate the ranking functions employed by spatial keyword queries. The system is capable of building rankings on an actual crowdsourcing platform, presenting the output to the users, and performing evaluation of the input ranking functions on a set of queries.

- Paper C [13] proposes a framework to address the problem of ranking evaluation under budget constraints. The framework includes a multistep process to decrease the number of questions to be processed. It also provides an entropy definition to understand the significance of a pairwise relevance question. The experimental evaluation findings suggest that the proposed framework is efficient and effective in assessing the quality of the ranking functions for spatial keyword queries.

- Paper D [23] investigates utilizing historical trips to build ground-truth rankings for spatial keyword queries. A method is proposed to extract these rankings from a set of GPS records obtained from vehicles. The

paper also proposes a new algorithm to identify the PoI visited as a result of a stop. Finally, it proposes a smoothing method to address the problem of data sparsity due to lack of users from specific spatial regions in the input set. An experimental evaluation is presented using a real GPS dataset consisting of GPS records obtained from 354 drivers for 9 months. The experimental evaluation suggests that the proposed stop assignment algorithm performs better than a method based on distance. Moreover, the evaluation findings provide clear evidence that the proposed smoothing algorithm is able to solve the data sparsity problem without distorting the original data.

- Paper E [26] proposes a stop assignment method for GPS data based on the intuition that visits to PoIs of different categories have different characteristics. The method proposed in the paper uses stay duration, day and time of the visit to determine the PoI category visited. The experimental evaluation suggests that the proposed method offers a high precision for regions with high PoI density.

- Paper F introduces k-TMSTC query that is a new spatial keyword query type that targets PoI clusters by taking the transportation mode into account. A DBSCAN-based algorithm and an OPTICS-based algorithm are proposed to process k-TMSTC queries. The experimental evaluation suggests that the proposed algorithms are able to offer a processing time that can support interactive search of the PoI clusters.

9   Conclusion

The thesis addresses the lack of ranking function evaluation for spatial keyword queries. The first three papers (Paper A, Paper B, and Paper C) focus on utilizing crowdsourcing for ranking function evaluation. Paper A proposes the PointRank algorithm to obtain ground-truth rankings for spatial keyword queries. The algorithm does not assume a total ranking on the PoIs and utilizes pairwise relevance questions. A simulation-based experimental evaluation suggests that the algorithm performs better than a baseline algorithm even when the reliability of the crowdsourcing workers is low. Paper B proposes a system called CrowdRankEval that is built on top of the Point-Rank algorithm. It is a complete framework to evaluate ranking functions for a given set of PoIs, a given set of queries, and a given set of query results. This system is a step towards more advanced and complex ranking functions for location-based services since it enables ranking function evaluations that were not possible before. Paper C proposes a method for ranking function evaluation without obtaining ground-truth rankings for spatial keyword queries. The method uses pairwise relevance questions just like PointRank.

Further, it utilizes a learn-to-rank method in order to capture the characteristics of the input rankings. This is used to determine the most important pairwise relevance questions that should be asked to the crowdsourcing workers. The method posts these questions to the crowdsourcing platform until the budget is exhausted. In order to determine the pairwise relevances that are not decided due to budget constraints, it uses a matrix factorization method. If a ground-truth ranking is needed for a query, the PointRank algorithm can be employed. However, if the aim is to evaluate a number of ranking functions, the method proposed in Paper C can be employed. It might be needed to evaluate another ranking function for the same query set after some time, and this might require another execution of the method in Paper C. This happens when a new output ranking for a query contains different PoIs than the PoIs included in the previous rankings and when the output ranking has pairs that are not concordant with the previous rankings.

Although the experimental evaluation shows that the crowdsourcing-based methods are capable of capturing user preferences, they have drawbacks that make it difficult to use them for a large scale evaluation. First, they are expensive and time-consuming since one has to pay a fee for each crowdsourcing task and it might take some time until workers answer all questions. Second, it is difficult to recruit workers who know about the spatial region that the query is issued in or know about the PoIs included in the output rankings. For this reason, Paper D proposes a model based on GPS-data. The proposed model uses historical trips to PoIs as a foundation for building ground-truth rankings. The paper also proposes a stop assignment method that takes into account users' temporal behavior patterns and uses a smoothing method to increase the spatial extent of the model. An experimental evaluation using a real GPS data suggests that the proposed model is able to capture user preferences while building ground-truth rankings. The stop assignment method proposed in Paper D is unable to assign stops to PoIs when the PoI density is high around the stop. To be able to identify the visited PoI in a region with high PoI density, Paper E proposes a method based on a Bayesian network that takes temporal attributes of stops into account. Empirical studies demonstrate that the Bayesian network structure is able to model the relationship between temporal attributes of stops and PoI categories.

The thesis also proposes a new query type to address browsing user behavior. Paper F formalizes the top-k transportation-mode aware spatio-textual clusters query and proposes two methods to process this query type efficiently.

The contributions of the thesis can be employed to increase the user satisfaction from location-based services. Future generations of location-based services can utilize the methods proposed to assess different hypotheses regarding query keywords, query locations, and different types of users. For instance, a ranking function that gives best results in a rural area might not

be the best ranking function in a city center. Another example would be that a ranking function that performs best for users with a car will probably perform poorly for users who plan to walk. Our proposals to evaluate ranking functions will make it possible to understand such relationships between queries and ranking functions. So, it might be possible for a system to employ different ranking functions for different scenarios due to the contributions of the thesis. The k-TMSTC query type can be used to extend the services provided by location-based services for the users who want to explore different options.

It is important to note that the proposals of the thesis to build ground-truth rankings should not be used as a replacement for query processing methods for spatial keyword queries. One reason is that the proposals are only applicable to limited geographical regions, namely the regions that the crowdsourcing workers are knowledgeable about for the crowdsourcing-based proposals and the regions covered by GPS records for the GPS-based proposal. Another reason is that the crowdsourcing-based proposals are not capable of building ground-truth rankings in interactive time since it requires assigning the questions to crowdsourcing workers and collecting answers from them. A key limitation of the GPS-based proposal is that it requires identifying the visited PoIs correctly. The thesis proposes two methods for this purpose. However, the proposed TPEA method cannot assign stops to PoIs in regions with high PoI density. The Bayesian network method proposed in Paper E solves this problem to some extent. However, it still fails when there are too many PoIs of the same category around a stop. This limitation can be addressed by employing GPS data collected from mobile phones instead of GPS data collected from vehicles and by using a PoI database that contains polygon information for each PoI. Use of walking GPS data and polygon information for PoIs may allow us to detect the exact PoI visited. So, it would be possible to identify visited PoIs even in the regions with high PoI density.

References

[1] (2017, april) Google annual search statistics. Accessed: 2018-03-06. [Online]. Available: http://www.statisticbrain.com/google-searches/

[2] O. Alonso and R. Baeza-Yates, "Design and implementation of relevance assessments using crowdsourcing," in *Proceedings of the 33rd European Conference on Information Retrieval (ECIR 2011)*. Springer, 2011, pp. 153–164.

[3] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," in *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD '99)*. ACM, 1999, pp. 49–60.

References

[4] D. Ashbrook and T. Starner, "Using GPS to learn significant locations and predict movement across multiple users," *Personal Ubiquitous Comput.*, vol. 7, no. 5, pp. 275–286, Oct. 2003.

[5] J. A. Aslam and M. Montague, "Models for metasearch," in *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '01).* ACM, 2001, pp. 276–284.

[6] T. Bhattacharya, L. Kulik, and J. Bailey, "Extracting significant places from mobile user GPS trajectories: A bearing change based approach," in *Proceedings of the 20th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL '12).* ACM, 2012, pp. 398–401.

[7] R. Blanco, H. Halpin, D. M. Herzig, P. Mika, J. Pound, H. S. Thompson, and T. Tran Duc, "Repeatable and reliable search system evaluation using crowdsourcing," in *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '11).* ACM, 2011, pp. 923–932.

[8] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," in *Proceedings of the Seventh International Conference on World Wide Web (WWW7).* Elsevier, 1998, pp. 107–117.

[9] X. Cao, L. Chen, G. Cong, C. S. Jensen, Q. Qu, A. Skovsgaard, D. Wu, and M. L. Yiu, "Spatial keyword querying," in *Proceedings of the 31st International Conference on Conceptual Modeling (ER 2012).* Springer, 2012, pp. 16–29.

[10] X. Cao, G. Cong, and C. S. Jensen, "Mining significant semantic locations from GPS data," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 1009–1020, Sep. 2010.

[11] ——, "Retrieving top-k prestige-based relevant spatial web objects," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 373–384, Sep. 2010.

[12] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi, "Collective spatial keyword querying," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD '11).* ACM, 2011, pp. 373–384.

[13] J. Chen, H. Lu, I. Keles, and C. S. Jensen, "Crowdsourcing based evaluation of ranking approaches for spatial keyword querying," in *Proceedings of the 18th IEEE International Conference on Mobile Data Management (MDM 2017).* IEEE, 2017, pp. 62–71.

[14] X. Chen, P. N. Bennett, K. Collins-Thompson, and E. Horvitz, "Pairwise ranking aggregation in a crowdsourced setting," in *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining (WSDM '13).* ACM, 2013, pp. 193–202.

[15] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 337–348, Aug. 2009.

[16] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96).* AAAI Press, 1996, pp. 226–231.

References

[17] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee, "Comparing partial rankings," *SIAM Journal on Discrete Mathematics*, vol. 20, pp. 47–58, 2004.

[18] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin, "Crowddb: Answering queries with crowdsourcing," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD '11)*. ACM, 2011, pp. 61–72.

[19] Google. (2014, May) Understanding consumers' local search behavior. Accessed: 2018-03-06. [Online]. Available: https://goo.gl/7TrZNl

[20] Q. Gu, D. Sacharidis, M. Mathioudakis, and G. Wang, "Inferring venue visits from GPS trajectories," in *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'17)*. ACM, 2017, pp. 81:1–81:4.

[21] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data (SIGMOD '84)*. ACM, 1984, pp. 47–57.

[22] J. Howe, *Crowdsourcing: Why the Power of the Crowd Is Driving the Future of Business*, 1st ed. Crown Publishing Group, 2008.

[23] I. Keles, C. S. Jensen, and S. Saltenis, "Extracting rankings for spatial keyword queries from GPS data," in *Proceedings of the 14th International Conference on Location Based Services (LBS 2018)*. Springer, 2018, pp. 173–194.

[24] I. Keles, C. S. Jensen, and S. Šaltenis, "Crowdrankeval: A ranking function evaluation framework for spatial keyword queries," in *Proceedings of the 17th IEEE International Conference on Mobile Data Management (MDM 2016)*. IEEE, 2016, pp. 353–356.

[25] I. Keles, S. Saltenis, and C. S. Jensen, "Synthesis of partial rankings of points of interest using crowdsourcing," in *Proceedings of the 9th Workshop on Geographic Information Retrieval (GIR '15)*. ACM, 2015, pp. 15:1–15:10.

[26] I. Keles, M. Schubert, P. Kröger, S. Šaltenis, and C. S. Jensen, "Extracting visited points of interest from vehicle trajectories," in *Proceedings of the Fourth International ACM Workshop on Managing and Mining Enriched Geo-Spatial Data (GeoRich '17)*. ACM, 2017, pp. 2:1–2:6.

[27] Z. Li, K. C. K. Lee, B. Zheng, W. C. Lee, D. Lee, and X. Wang, "IR-tree: An efficient index for geographic document search," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 4, pp. 585–599, April 2011.

[28] P. Liu, D. Zhou, and N. Wu, "Vdbscan: Varied density based spatial clustering of applications with noise," in *Proceedings of the International Conference on Service Systems and Service Management (ICSSSM 2007)*, 2007, pp. 1–4.

[29] Y. Liu and H. S. Seah, "Points of interest recommendation from GPS trajectories," *Int. J. Geogr. Inf. Sci.*, vol. 29, no. 6, pp. 953–979, Jun. 2015.

[30] C. Long, R. C.-W. Wong, K. Wang, and A. W.-C. Fu, "Collective spatial keyword queries: A distance owner-driven approach," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13)*. ACM, 2013, pp. 689–700.

References

[31] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller, "Crowdsourced databases: Query processing with people," in *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research (CIDR '11)*, 2011, pp. 211–214.

[32] K. Nishida, H. Toda, T. Kurashima, and Y. Suhara, "Probabilistic identification of visited point-of-interest for personalized automatic check-in," in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '14).* ACM, 2014, pp. 631–642.

[33] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, TR 1999-66, 1999.

[34] A. Parameswaran and N. Polyzotis, "Answering queries using humans, algorithms and databases," in *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research (CIDR '11)*, 2011, pp. 160–166.

[35] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg, "Efficient processing of top-k spatial keyword queries," in *Proceedings of the International Symposium on Spatial and Temporal Databases (SSTD 2011).* Springer, 2011, pp. 205–222.

[36] R. Schumacker and S. Tomek, "Chi-square test," in *Understanding Statistics Using R*, 2013, pp. 169–175.

[37] A. Skovsgaard and C. S. Jensen, "Finding top-k relevant groups of spatial web objects," *The VLDB Journal*, vol. 24, no. 4, pp. 537–555, Aug. 2015.

[38] J. Stoyanovich, M. Jacob, and X. Gong, "Analyzing crowd rankings," in *Proceedings of the 18th International Workshop on Web and Databases (WebDB '15).* ACM, 2010, pp. 41–47.

[39] D. Wu and C. S. Jensen, "A density-based approach to the retrieval of top-k spatial textual clusters," in *Proceedings of the 25th ACM International Conference on Information and Knowledge Management (CIKM '16).* ACM, 2016, pp. 2095–2100.

[40] W. Wu and K. L. Tan, "isee: Efficient continuous k-nearest-neighbor monitoring over moving objects," in *Proceedings of the 19th International Conference on Scientific and Statistical Database Management (SSDBM 2007)*, 2007, pp. 36–36.

[41] J. Yi, R. Jin, S. Jain, and A. Jain, "Inferring users' preferences from crowdsourced pairwise comparisons: A matrix completion approach," in *Proceedings of the First AAAI Conference on Human Computation and Crowdsourcing (HCOMP 2013)*, 2013, pp. 207–215.

[42] V. W. Zheng, Y. Zheng, X. Xie, and Q. Yang, "Collaborative location and activity recommendations with GPS history data," in *Proceedings of the 19th International Conference on World Wide Web (WWW '10).* ACM, 2010, pp. 1029–1038.

[43] Y. Zheng and X. Xie, "Learning travel recommendations from user-generated GPS traces," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 1, pp. 2:1–2:29, Jan. 2011.

[44] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma, "Mining interesting locations and travel sequences from GPS trajectories," in *Proceedings of the 18th International Conference on World Wide Web (WWW '09).* ACM, 2009, pp. 791–800.

[45] J. Zobel and A. Moffat, "Inverted files for text search engines," *ACM Comput. Surv.*, vol. 38, no. 2, Jul. 2006.

Part II

Papers

Paper A

Synthesis of Partial Rankings of Points of Interest
Using Crowdsourcing

Ilkcan Keles, Simonas Šaltenis, Christian S. Jensen

Abstract

The web is increasingly being accessed from mobile devices, and studies suggest that a large fraction of keyword-based search engine queries have local intent, meaning that users are interested in local content and that the underlying ranking function should take into account both relevance to the query keywords and the query location. A key challenge in being able to make progress on the design of ranking functions is to be able to assess the quality of the results returned by ranking functions. We propose a model that synthesizes a ranking of points of interest from answers to crowdsourced pairwise relevance questions. To evaluate the model, we propose an innovative methodology that enables evaluation of the quality of synthesized rankings in a simulated setting. We report on an experimental evaluation based on the methodology that shows that the proposed model produces promising results in pertinent settings and that it is capable of outperforming an approach based on majority voting.

1   Introduction

Many web users view the web in terms of the results returned by search engines such as Google and Bing, as well as numerous vertical search engines. One source [1] reports that the Google search engine processes some 5–6 billion queries on a daily basis. At the same time, the web is rapidly becoming increasingly mobile. Specifically, while the web was accessed mostly from desktop computers in the past, it is now being accessed predominantly by mobile smartphone and tablet users [2]. Further, a recent study [3] reports that some 50% of all mobile Bing queries have what is termed local intent, meaning that the users are querying for web content that relates to geographically nearby points of interest (PoI).

This state of affairs calls for the integration of location into keyword based web querying, and the research community has proposed a range of spatial-keyword functionality that aims to find relevant nearby PoIs [4]. For example, some proposals identify top-k PoIs according to a scoring function that takes into account the relevance of the PoIs to user-provided query terms and an automatically supplied user location.

Existing proposals generally come with advanced indexing and query processing techniques that aim to provide computationally efficient support for the proposed functionality, and these techniques are generally subjected to detailed empirical studies of their computational efficiency. In contrast, studies of the quality of the ranked results that are computed by the proposed techniques are often entirely missing or are relatively small scale. While the results of k nearest neighbor queries are given by a mathematical definition, there is no mathematical definition of the best result to a keyword or spatial-keyword query. Rather, the best result is the result that users prefer. This is why search engines integrate user feedback into their algorithms. This also renders the evaluation of the quality of a ranking function very challenging. At the same time, the ability to be able to reliably assess the quality of ranking functions is essential to make progress on the design of more advanced and better ranking functions.

This paper proposes a model that uses crowdsourcing [5] for the assessment of the quality of rankings of sets of PoIs that are returned in response to spatial-keyword queries. Specifically, the model assigns pairwise relevance questions to workers, asking them to determine which of a pair of PoIs is the most relevant to a query given by query keywords and a location. Based on the answers received, the model synthesizes a partial ranking of the PoIs. It is then possible to compare the rankings produced by different ranking functions with the rankings achieved by the model, the idea being that the ranking function that returns results that are most similar to the synthesized ranking is preferable.

A model such as the one proposed here should deliver a synthesized ranking that is as good as possible while asking as few questions of the workers as possible. It is straightforward to count the number of questions asked, so the key challenge lies in determining how good a synthesized ranking is. The problem is that, in practice, no ground truth rankings are available that the synthesized rankings can be compared with. Thus, the question is how to determine the quality of a synthesized ranking.

To address this issue, we adopt a quality assessment methodology where we first construct a ground-truth ranking of PoIs. Then we simulate workers that answer pairwise relevance questions about the PoIs. More specifically, we vary the probability that a worker gives a correct answer (according to the constructed ground-truth ranking). The model then chooses questions to ask and synthesizes a ranking based on the, sometimes incorrect, answers received. The quality of the synthesized ranking can be quantified according to how close it is to the ground-truth ranking. This way, it is possible to study the trade-off between numbers of questions asked and ranking quality, and it is possible to compare different models and to choose the models that offers the best trade-off. The underlying assumption is that the model that performs the best in the simulated setting also performs the best in real-world settings where there is no ground truth to compare with.

A few studies exist [6–9] that address rank aggregation via crowdsourcing. Stoyanovich et al. [6] propose a method that uses listwise relevance questions for rank aggregation. In contrast, we have opted to use simple pairwise relevance questions that we believe are preferable in our setting where workers may not be sufficiently familiar with many PoIs to provide reliable answers. Two other studies [7, 8] assume that there is a total ranking between the objects being compared. We believe that this assumption is too strong in our setting, and we assume only a partial ranking of PoIs and allow workers to answer that a pair of PoIs are incomparable. These existing studies use majority voting when synthesizing a ranking from answers received from workers. Instead of asking each question to a fixed number of workers, we propose an iterative approach that determines the number of times to ask each question. Urbano et al. [9] propose a divide and conquer algorithm to obtain rankings using crowdsourcing, but they also do not consider the case of incomparable objects.

To summarize, we present a model for the synthesis of partial rankings of PoIs in response to spatial keyword queries based on answers to crowdsourced pairwise relevance questions. The main contributions are:

- We propose a model for synthesizing rankings of PoIs. The proposed model (i) exploits crowdsourced answers to simple pairwise relevance questions, (ii) supports incomparability of PoIs, (iii) supports partial rankings, and (iv) includes an iterative approach to determining the

> numbers of questions to be asked as opposed to using simple majority voting.

- We propose a new evaluation methodology that enables evaluation of the quality of rankings in a simulated setting.

- We present findings of an experimental study of the proposed model that includes a comparison with a majority voting approach; the findings suggest that the proposed model represents an improvement over the state-of-the-art.

The remainder of the paper is organized as follows. Section 2 covers related work and defines the problem addressed. The details of the proposed model are covered in Section 3. This is followed by an evaluation of the model in Section 4. Section 5 concludes and offers research directions.

2 Preliminaries

2.1 Problem Definition

We consider a setting where D denotes the set of PoIs returned in response to a spatial keyword query. A spatial keyword query takes keywords and a location as arguments and returns a ranked list of PoIs that are relevant to the query keywords and close to the query location. The relevance of a PoI to a query is defined in terms of its proximity to the query location and its textual relevance to the query keywords.

We assume a pairwise relevance relation $\prec$ on D that captures the notion of one PoI being more relevant to the result than another. The relation is a partial order and is *irreflexive* ($\forall p_i \in D$ ($p_i \not\prec p_i$)), *transitive* ($\forall p_i, p_j, p_k \in D$ ($p_i \prec p_j \wedge p_j \prec p_k \Rightarrow p_i \prec p_k$)), and *asymmetric* ($\forall p_i, p_j \in D$ ($p_i \prec p_j \Rightarrow p_j \not\prec p_i$)). If, for each pair ($p_i, p_j$), $p_i \prec p_j$ or $p_j \prec p_i$ the order is total. An ordered pair (p_i, p_j) that is an element of a pairwise relevance relation ($p_i \prec p_j$) is called a *pairwise relevance*.

In this setting, the problem is to design a model that is able to construct a pairwise relevance relation $\prec$ on D using crowdsourcing, by generating and crowdsourcing pairwise relevance questions and by aggregating the answers received, while achieving the following:

1. The synthesized relation $\prec$ should be of high quality, i.e., it should be similar to a ground-truth relation.

2. The relation should be synthesized in an efficient manner, i.e., the number of questions and answers needed should be low.

2.2   Related Work

A number of recent studies consider the use of crowdsourcing in order to apply human wisdom to computational tasks. These works include relational database systems (e.g., [10–12]) and information search and retrieval systems (e.g., [13, 14]). Some studies consider the use of crowdsourcing for specific database operations (sorts and ranking [15], filtering [16], maximum [17], and top-k and group by operations [18]).

Crowdsourcing is also used to perform relevance assesments in information retrieval. Normally, to evaluate relevance in information retrieval, relevance assesments are obtained from a group of experts. In order to reduce the cost and effort of relevance assesments, studies consider the use of crowdsourcing [19–21]. However, these studies only consider the use of graded relevance questions for assessing relevance. Specifically, workers are presented with a query and one result object, and they are then asked to choose one of several relevance levels for the relevance of the object to the query. This kind of question is difficult to apply in our setting where no predefined relevance levels for spatial keyword queries exist. Further, we believe that offering means of comparing the relevances of objects improves the quality of the workers' answers.

Some studies propose different methods to obtain rankings using crowdsourcing. Stoyanovich et al. [6] propose a method where workers are asked to rank a set of objects. Then, the results of these questions are aggregated into a global ranking. Yi et al. [7] propose a matrix completion approach to obtain a complete ranking with the use of pairwise comparisons in crowdsourcing tasks. They assume that the complete ranking function is a combination of a number of intrinsic ranking functions, which is called the low rank assumption. They also do not consider incomparable pairs, meaning that they assume every set of objects has a total ranking. Chen et al. [8] propose a model and an active learning scheme to infer the rankings of objects. They extend the Bradley Terry model [22] to incorporate worker reliability. They do not consider the case of objects being incomparable. Urbano et al. [9] propose an algorithm similar to Quicksort to compute the rankings for music documents. This algorithm cannot be used in our settings since its divide and conquer methodology fails when there are incomparable objects.

3   Proposed Method

In the following, we propose an algorithm called PointRank. Section 3.2 gives an overview of the algorithm, and Sections 3.3 and 3.4 detail how questions to the crowd are chosen and processed.

3.1   Preliminaries

A *pairwise relevance question* concerns a pair of PoIs and asks the crowd which of the two PoIs is more relevant to a query. An *assignment* is the assignment of a pairwise relevance question to a worker. The worker can give three different answers: the first PoI is more relevant, the second PoI is more relevant, and the PoIs are incomparable. A worker is expected to use the last option when the PoIs are very different and difficult to compare or if the worker feels that the comparison is up to personal preferences.

For each pairwise relevance question, the algorithm runs a number of iterations. In each iteration, the pairwise relevance question is assigned to a number of workers. To see whether there is a significant change between the answers of different iterations, the *Chi-square(χ^2) test* [23] is applied. We say that there is *consensus* regarding a pairwise relevance question when the change in the answers of the assignments between consecutive iterations is not significant.

The formula for the Chi-square test is $\chi^2 = \sum_{k=1}^{n} \frac{(o_k - e_k)^2}{e_k}$, where o_k refers to the observed value and e_k refers to the expected value. Knowing the value of the chi-square, the chi-square distribution table gives the corresponding *p-value*. The *p-value* is the probability that the difference between the expected and the observed values is due to chance.

3.2   PointRank Overview

We represent the input to the algorithm—the answers of workers to pairwise relevance questions—using an *edge-weighted directed graph* $G = (V, E, NW, NA)$, where V is the set of vertices that represent PoIs, E is a set of directed edges, each given by a pair of vertices, and NW and NA are edge weights, specifically, functions from $V \times V$ to the natural numbers. Here, $NA(p_i, p_j)$ is the number of assignments regarding the pairwise relevance question about PoIs p_i and p_j, and $NW(p_i, p_j)$ is the number of workers who prefer p_i over p_j. Ratio $NW(p_i, p_j)/NA(p_i, p_j)$ then defines the *probability of edge* (p_i, p_j).

To understand how the edges of the graph are created and updated, consider a pair of PoIs p_i and p_j. If some assignments regarding this pair have been processed, the graph will contain two edges: (p_i, p_j) and (p_j, p_i). Further, $NA(p_i, p_j)$ and $NA(p_j, p_i)$ will have the same value, the total number of assignments for this pair. Note that $NW(p_i, p_j) + NW(p_j, p_i) \leq NA(p_i, p_j)$. The sum is not always equal to the total number of assignments, since it is possible for a worker to answer that p_i and p_j are incomparable. This is recorded by incrementing the NA values on both of the edges without changing the NW values.

To record the output of the algorithm—the pairwise relevance relation— we maintain a two-dimensional matrix called the Pairwise Relevance Matrix

(PRM). PRM encodes all the pairwise relevances discovered by the algorithm from the answers of the crowd, including the pairwise relevances obtained directly from the answers as well as the pairwise relevances inferred using the transitivity property of the pairwise relevance relation. It is an $n \times n$ matrix, where n is the number of PoIs in the query answer. Each row and column of the matrix represents a PoI. The value of a cell records the pairwise relevance between two PoIs. Let us assume that we have a PRM M that is defined on the set of PoIs $D = \{p_1, .., p_n\}$. A cell $M[i, j]$ can have one of the five possible values:

- $M[i, j] = 1$ encodes that p_j is more relevant than p_i.

- $M[i, j] = 0$ encodes that p_i and p_j are incomparable. Since an object is not comparable with itself, the diagonal cells have 0's.

- $M[i, j] = -1$ encodes that p_i is more relevant than p_j. Since the pairwise relevance relation is asymmetric, if $M[i, j] = -1$ then $M[j, i] = 1$.

- $M[i, j] = 2$ encodes that the pair (p_i, p_j) has not yet been processed. In the beginning of the algorithm, all of the cells except the diagonal cells have this value. If $M[i, j] = 2$ then $M[j, i] = 2$.

- $M[i, j] = 3$ encodes that the pair (p_i, p_j) has been processed but that the algorithm cannot conclusively decide about the relation between p_i and p_j. If $M[i, j] = 3$ then $M[j, i] = 3$.

A PRM M has the following properties:

- *Transitivity*: This property is used to infer pairwise relevances. If $M[i, k] = 1$ and $M[k, j] = 1$ then $M[i, j] = 1$. This is not the case if, when the inference is made, $M[i, j]$ is already -1 or 0 due to previous answers of the crowd (as described next).

- *Possibility of Inconsistencies*: M can contain inconsistencies as workers may give contradicting answers. We say that there is an *inconsistency* regarding a pair of PoIs (p_i, p_j) if it is possible to infer $M[i, j] = 1$ using the transitivity property from the cells $M[i, k] = 1$ and $M[k, j] = 1$, but $M[i, j] \neq 1$.

The general flow of the algorithm is shown in Figure A.1. After initialization of the PRM, in each iteration, the algorithm goes through two different phases: determining the next question and processing the question. If the relevance of every pair is computed (there are no 2s in the pairwise relevance matrix), it returns the PRM.

The flow chart of determining the next question is presented in Figure A.2. The algorithm takes the PRM as input and determines the possible pairwise

3. Proposed Method
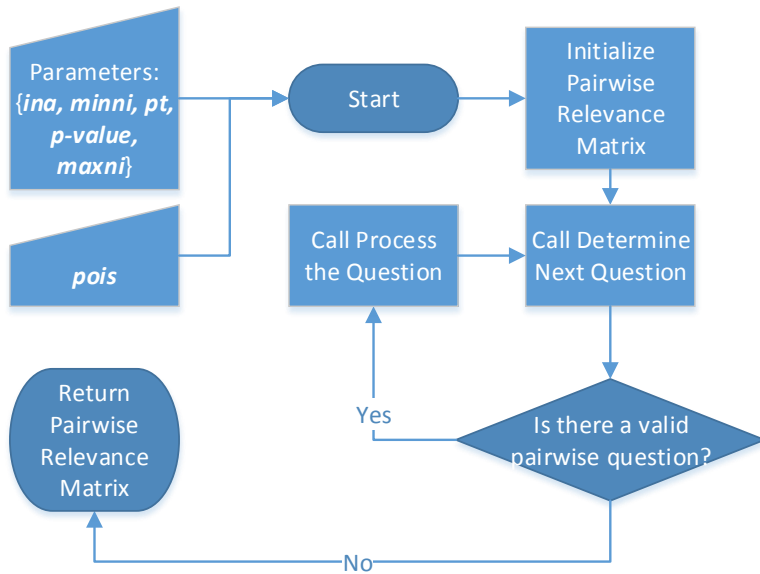
Fig. A.1: General Flow of PointRank
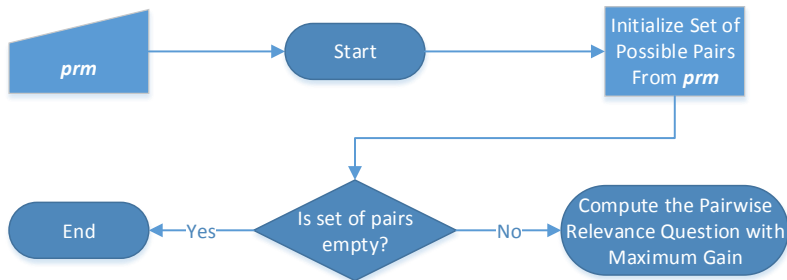
Fig. A.2: Flow of the Determine-Next-Question Phase

relevance questions to be asked next. If the set of questions is empty, it stops. Otherwise, it computes the gain for each of the questions and returns the question with the maximum gain.

The flow chart of processing the question phase is shown in Figure A.3.
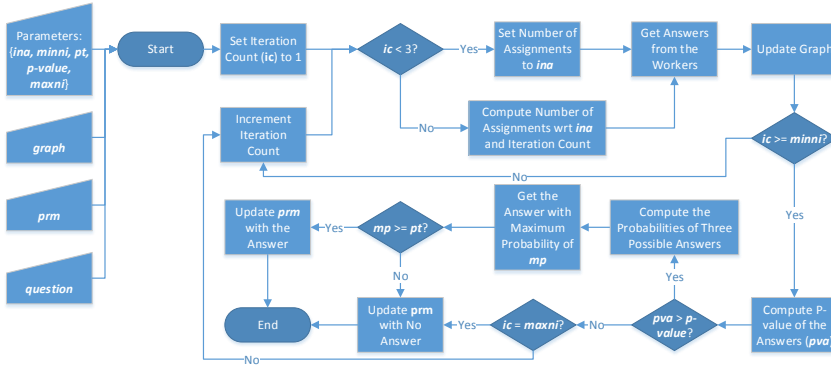
Fig. A.3: Flow of the Process-Question Phase

It takes the graph, the PRM, and the next question as input. It uses several parameters that are explained shortly. In this phase, the algorithm employs an iterative approach. In each iteration, it first determines the number of assignments for the question. Then it assigns the question to workers, gets the answers back, and updates the graph. Finally, it checks whether consensus on the question is reached and updates the pairwise relevance matrix accordingly.

PointRank uses six parameters: *pois*, *ina*, *minni*,*pvalue*, *maxni*, *pt*. Parameter *pois* is the list of PoIs to be ranked. Parameter *ina* is the initial number of assignments for each pairwise relevance question. As shown in Figure A.3, this parameter is used to determine the number of assignments for the question in the current iteration. If the iteration count is 1 or 2, the number of assignments is set to *ina*. Otherwise, it is computed with respect to *ina* and the iteration count. The algorithm makes at least two iterations in order to be able to apply significance testing to the answers of the workers. Parameter *minni* is the minimum number of iterations to stop creating new assignments for a pairwise relevance question. As shown in Figure A.3, the algorithm completes *minni* iterations before checking for consensus regarding a question. Parameter *pvalue* is the maximum *p-value* in the chi-square test needed to consider the changes in the answers to assignments in consecutive iterations as significant. As shown in Figure A.3, in each iteration, the algorithm applies the significance test to the accumulated answers for the question and the answers from this iteration. If the *p-value* of the test does not exceed *pvalue*, it continues to the next iteration. Parameter *maxni* is the maximum number of iterations for a pairwise relevance question. As shown in Figure A.3, if the question does not have a consensus after *maxni* iterations, the al-

gorithm cannot make a decision regarding this pair. In other words, if there is no consensus after *maxni* iterations, the algorithm stops and sets the value of the corresponding cells to 3 in the PRM. Parameter *pt* is the probability threshold needed to determine the answer for the pairwise relevance question. In other words, in order to conclude that PoI p_i is preferred over PoI p_j, the probability of the (p_i, p_j) edge should exceed *pt* as shown in Figure A.3.

Algorithm A.1 PointRank Algorithm

Input: *pois, ina, pt, minni, maxni, pvalue*
Output: *prm*
 1: $n \leftarrow pois.length$
 2: Initialize *prm* and *graph*
 3: $nq \leftarrow$ **DetermineNextQuestion**(*prm, n*)
 4: **while** $nq \neq null$ **do**
 5:     $graph, prm \leftarrow$ **ProcessTheQuestion**(*graph, prm, nq, ina, pt, minni, maxni, pvalue*)
 6:     $nq \leftarrow$ **DetermineNextQuestion**(*prm, n*)
 7: **end while**
 8: **return** *prm*

The complete algorithm is presented in Algorithm A.1. To build the graph and to incorporate the answers of the workers into the model, we employ an iterative approach. First, the algorithm initializes the pairwise relevance matrix and the graph of answers as shown in line 2. In the initial step, the algorithm checks whether there is a valid next question as shown in lines 3–4. If so, the algorithm processes the pairwise relevance question and gets the next question as shown in lines 5–6. If the algorithm does not need any further questions to complete the procedure, it returns the constructed pairwise relevance matrix as shown in line 8.

3.3 Determining the Next Question

To reduce the number of questions to ask to the crowd, we define a procedure to determine the next question. The next question is determined with respect to the current status of the pairwise relevance matrix and the gain criteria.

The algorithm checks the possible pairwise relevance questions and computes their gains. Then it returns the question with the maximum gain. If more than one question have the same maximum gain, it selects one of them randomly.

The complete algorithm to determine the next question is presented in Algorithm A.2. The algorithm first finds the set of unordered pairs that have not yet been processed as shown in line 1. If the set is empty, all pairs have been processed, and the algorithm returns *null* as shown in line 13. Then,

---

Algorithm A.2 DetermineNextQuestion Algorithm

Input: *prm*, *n*
Output: *nextQuestion*

  1: *pairs* ← $\{(i, j) \mid prm[i, j] = 2 \wedge i < j\}$
  2: **if** *pairs.length* $\neq 0$ **then**
  3:      Initialize gain array *gains*
  4:      **for** $k \leftarrow 0$ **to** *pairs.length* **do**
  5:          $(i, j) \leftarrow pairs[k]$
  6:          $gain_1 \leftarrow$ **ComputeGain**(*prm*, *i*, *j*)
  7:          $gain_2 \leftarrow$ **ComputeGain**(*prm*, *j*, *i*)
  8:          $gains[k] \leftarrow$ **Avg**($gain_1$, $gain_2$)
  9:      **end for**
10:      *nextQuestion* ← **GetPairWithMaximumGain**(*gains*)
11:      **return** *nextQuestion*
12: **else**
13:      **return** *null*
14: **end if**

---

for all the chosen pairs, the algorithm computes the gain. Since there are two possible outcomes for a question that may be used to infer new pairwise relevances (the two outcomes excluding "incomparable"), the algorithm computes the gain for both of them as shown in lines 6–7. Then, the average of these two values is taken as the gain of asking the question regarding this pair as shown in line 8. Finally, the algorithm returns the pair with the maximum gain value.

To define the gain, transitivity of the pairwise relevance relation is used. The gain of asking a question about a pair of PoIs is defined as the number of questions that may be eliminated by the answer to this question.

The complete algorithm to compute the gain with respect to an input of a matrix *prm*, row index *ri*, and column index *ci* is given in Algorithm A.3. The algorithm checks all of the pairs to determine which new pairwise relevances can be inferred.

More specifically, the algorithm checks the pairs that have not been processed or not determined yet and are different from the input pair (ri, ci) as shown in lines 5–6. The algorithm excludes inconsistent inferences as shown in line 7. The pairwise relevance $p_{ri} \prec p_{ci}$ can be used in three ways to infer new pairwise relevances:

- Transitivity can be used to infer $p_{ri} \prec p_j$ if it is known that $p_{ci} \prec p_j$. This type of gain is computed in lines 9–10.

- Transitivity can be used to infer $p_i \prec p_{ci}$ if it is known that $p_i \prec p_{ri}$. This type of gain is computed in lines 11–12.

Algorithm A.3 ComputeGain Algorithm

Input: *prm*, *ri*, *ci*
Output: *gain*
1: $gain \leftarrow 0$
2: $n \leftarrow prm.length$
3: **for** $i \leftarrow 0$ **to** n **do**
4:     **for** $j \leftarrow 0$ **to** n **do**
5:         **if** $i = ri \wedge j = ci$ **then**
6:             continue
7:         **else if** $prm[i,j] \neq 2 \wedge prm[i,j] \neq 3$ **then**
8:             continue
9:         **else if** $i = ri \wedge prm[ci,j] = 1$ **then**
10:             $gain \leftarrow gain + 1$
11:         **else if** $j = ci \wedge prm[i,ri] = 1$ **then**
12:             $gain \leftarrow gain + 1$
13:         **else if** $prm[i,ri] = 1 \wedge prm[ci,j] = 1$ **then**
14:             $gain \leftarrow gain + 1$
15:         **end if**
16:     **end for**
17: **end for**
18: **return** *gain*

- Transitivity can also be used to infer $p_i \prec p_j$ if it is known that $p_i \prec p_{ri}$ and $p_{ci} \prec p_j$. This type of gain is computed in lines 13–14.

    Example. Let $D = \{p_1, p_2, p_3, p_4, p_5\}$ be the PoIs to be ranked. Let *ina*, *minni*, *maxni*, *pvalue*, and *pt* be 5, 3, 5, 0.25, and 0.6, respectively. The current state of the graph and pairwise relevance table are given in Figure A.4 and Table A.1, respectively.

	p_1	p_2	p_3	p_4	p_5
p_1	0	1	2	0	1
p_2	-1	0	3	-1	2
p_3	2	3	0	2	2
p_4	0	1	2	0	1
p_5	-1	2	2	-1	0

Table A.1: Current State of the PRM

    To determine the next question, the algorithm initializes the set of possible pairwise questions as $P = \{(p_1, p_3), (p_2, p_5), (p_3, p_4), (p_3, p_5)\}$. Then, it computes the gain for each question.
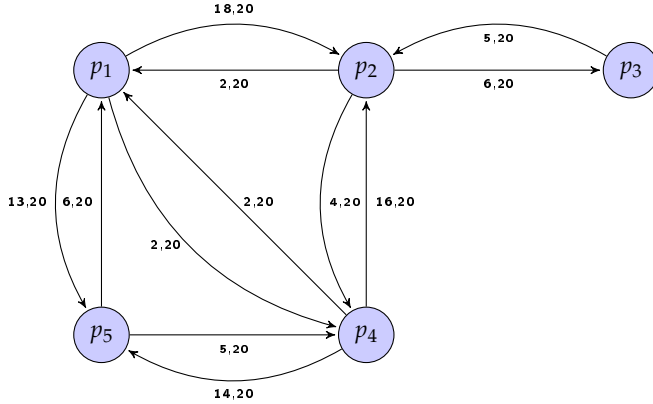
Fig. A.4: Current State of the Graph

For example, to compute the gain of (p_3, p_4), the algorithm checks two possible answers:

- $p_3 \prec p_4$. From this answer, the algorithm may infer $p_3 \prec p_2$ and $p_3 \prec p_5$ since $p_4 \prec p_2$ and $p_4 \prec p_5$ are in the PRM as shown in Table A.1. The gain of this answer is 2.

- $p_4 \prec p_3$. From this answer, the algorithm can not infer any pairwise relevances. It could infer a new pairwise relevance if $\exists p_i \in D(p_i \prec p_4 \lor p_3 \prec p_i)$. Since no such p_i exists, the gain of this answer is 0.

Since the gain is defined as the average of the gains of possible answers, the gain of (p_3, p_4) is 1. This question is one of the questions with the maximum gain, so it is selected as the next question.

3.4 Processing the Question

After the question is determined, the algorithm processes the question in a number of iterations. As shown in Algorithm A.4, in each iteration, the algorithm determines the number of assignments for the question, gets the answers from the workers, and updates the graph accordingly. The algorithm updates the PRM after it completes all the iterations for the question.

In the first iteration, the algorithm creates *ina* assignments for the question. In the upcoming iterations, the number of assignments is determined based on the total number of assignments in the previous iterations for the question. Specifically, if the algorithm is in the $(n + 1)$st iteration and the previous iterations had k assignments in total, the algorithm creates k additional assignments in this iteration. This ensures that the new iteration can have a significant effect for this pair of PoIs, which is tested using a significance test.

Algorithm A.4 ProcessTheQuestion Algorithm

Input: *graph, prm, nextQ, ina, pt, minni, maxni, pvalue*
Output: *graph, prm*

1: $ni \leftarrow 0$
2: $(i, j) \leftarrow$ the indexes of PoIs in *nextQ*
3: **while** true **do**
4:      Get edges from *graph* corresponding to *nextQ* as *currentEdges*
5:      Determine the number of assignments in this iteration
6:      Get answers for the assignments regarding *nextQ*
7:      Update *graph* with the answers
8:      **if** $ni < minni$ **then**
9:          $ni \leftarrow ni + 1$
10:          continue
11:      **end if**
12:      Get edges from *graph* corresponding to *nextQ* as *updatedEdges*
13:      $edgePvalue \leftarrow$ *p-value* corresponding to χ^2 value between *currentEdges* and *updatedEdges*
14:      **if** $edgePvalue \geq pvalue$ **then**   ▷ stopping criteria is reached
15:          $prb_1 \leftarrow NW/NA$ of the edge for the answer 1
16:          $prb_{-1} \leftarrow NW/NA$ of the edge for the answer -1
17:          $prb_0 \leftarrow 1 - prb_1 - prb_{-1}$
18:          $edgeProbability \leftarrow \max(prb_1, prb_{-1}, prb_0)$
19:          $answer \leftarrow$ the answer with the maximum probability
20:          **if** $edgeProbability > pt \wedge answer = 0$ **then**
21:              Set $prm[i, j]$ and $prm[j, i]$ to 0
22:          **else if** $edgeProbability > pt$ **then**
23:              Set $prm[i, j]$ to 1 and $prm[j, i]$ to -1 or vice versa with respect to the answer.
24:              **UpdateMatrix**(*prm*)
25:          **else**
26:              Set $prm[i, j]$ and $prm[j, i]$ to 3
27:          **end if**
28:          break
29:      **else if** $ni = maxni$ **then** ▷ maximum number of iterations are finished but the question still has no consensus.
30:          Set $prm[i, j]$ and $prm[j, i]$ to 3
31:          break
32:      **end if**
33: **end while**
34: **return** *graph, prm*

When the answers from the workers are received, the graph is updated accordingly. If the graph does not have any edge between the PoIs, two edges are added to the graph. Otherwise, the *NA* and *NW* weights of the corresponding edges are updated. It should be noted that if a worker's answer is that the PoIs are incomparable, only the *NA* weight is incremented on the two edges; the *NW* weights are not changed.

When the graph is updated, the algorithm computes a *p-value* with respect to the previous iteration as shown in line 13. Here, the sets that we compare in the χ^2 test are the $NW(p_i, p_j)$ and $NW(p_j, p_i)$ values in the previous iteration and in the current iteration. So in this test, the degree of freedom is 1. Then, the algorithm checks whether it has reached the stopping conditions as shown in lines 14–32. There are two types of stopping conditions:

- The edge has consensus. The test for this case is shown in line 14. When this is the case, the algorithm computes the probability values of the possible answers and gets the answer with the maximum probability as shown in lines 15–19. It then updates the cells in the pairwise relevance matrix as shown in lines 20–24. If no answer has a probability value that exceeds the probability threshold *pt*, the algorithm sets the cells of the pairwise relevance matrix to 3 as shown in line 26.

- The maximum number of iterations is reached, but there is no consensus. The test for this case is shown in line 29. When this occurs, the corresponding cells of the pairwise relevance matrix are set to 3. .

The algorithm for updating the PRM in line 24 is the same as the ComputeGain algorithm (Algorithm A.3). However, instead of incrementing the gain in lines 10,12, and 14 of Algorithm A.3, the matrix is updated with the inferred value. Note that we do not change the value of a cell to an inferred value if it is already 1, -1, or 0 (line 8 in Algorithm A.3). In other words, we choose to value the direct opinion of the crowd over inferred information, even if this involves inconsistencies. To illustrate, if question (p_i, p_j) is asked to the workers, their consensus answer is $p_i \prec p_j$, and the current PRM M contains both $M[j, k] = 1$ (encoding $p_j \prec p_k$) and $M[i, k] = 0$, the updated PRM will have $M[i, k] = 0$ instead of the inferred $M[i, k] = 1$.

Example. Continuing from the example in Section 3.3, we describe the processing of the question (p_3, p_4).

Three iterations have to be finished before looking for consensus, since *minni* parameter is set to 3. We assign this question to 5, 5, and 10 workers in the first, second, and third iterations, respectively, since *ina* = 5. At the end of the third iteration, we have a total of 20 assignments. Let us assume that there is consensus after the third iteration and that 15 workers stated that $p_3 \prec p_4$, 3 workers stated the opposite, and 2 workers stated that the PoIs are incomparable. The algorithm updates the graph with the answers from the workers, and the updated graph is shown in Figure A.5.
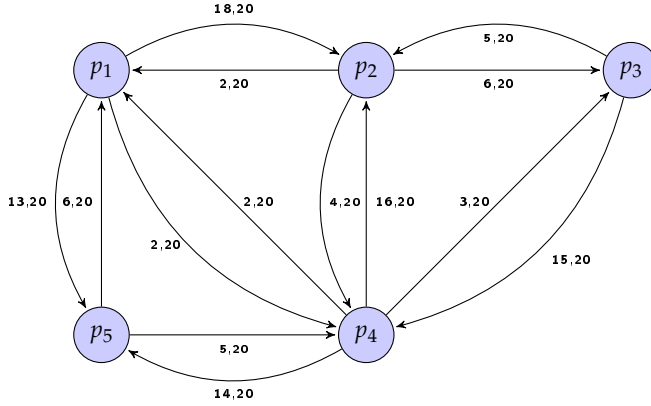
Fig. A.5: Updated Graph

To update the pairwise relevance matrix, we check the probability of the answers. Since the answer with the maximum probability is 1 and its probability is 0.75, which exceeds the *pt* parameter (0.6), we set the answer to 1. Then, the pairwise relevance matrix is updated with respect to this answer. This answer leads to inferring $p_3 \prec p_2$ and $p_3 \prec p_5$ since the matrix contains the pairwise relevances $p_4 \prec p_2$ and $p_4 \prec p_5$. The updated PRM is shown in Table A.2, with the changed values in bold.

	p₁	**p₂**	**p₃**	**p₄**	**p₅**
p₁	0	1	2	0	1
p₂	-1	0	**-1**	-1	2
p₃	2	**1**	0	**1**	**1**
p₄	0	1	**-1**	0	1
p₅	-1	2	**-1**	-1	0

Table A.2: Updated PRM

3.5   Worst Case Complexity

PointRank generates $C(n, 2)$ ($\Theta(n^2)$) questions in the worst case if it cannot eliminate any questions by using previous answers of workers. The number of assignments is affected by the algorithm parameters. In the worst case, the algorithm does not reach consensus before *maxni* iterations for each pairwise relevance question. As a result, the number of assignments per question is $2^{maxni} \cdot ina$ and the total number of assignments is $C(n, 2) \cdot 2^{maxni} \cdot ina$.

4 Experimental Evaluation

We proceed to evaluate the proposed algorithm. First, we present the experimental setup. Then we present experimental results exploring the effect of the parameters of the algorithm. Finally, we present an experimental comparison with the baseline algorithm.

4.1 Experimental Setup

We use simulation for the experimental evaluation. First, we generate the assumed ground-truth ranking on a given number of PoIs. Both total and partial rankings are generated in order to study different ranking types. A total ranking is created as a random permutation of the set of PoIs. To generate partial rankings, a total ranking is created first. Then, we iteratively introduce incomparable pairs. In each iteration, a random pair is selected. If changing this pair to incomparable does not lead to an inconsistency, it is changed. Otherwise, another random pair is selected. In this way, the generated ground-truth ranking does not contain any inconsistencies.

To simulate worker behavior, we assign a reliability value to each worker. The reliability is the probability of giving the correct answer to a pairwise relevance question. The correct answer is the answer that corresponds to the assumed ground truth. If a simulated worker has a reliability of 0.8, the worker will produce the correct answer with probability 0.8. Each of the two wrong answers will be produced with probability 0.1.

Each point in the performance graphs corresponds to the average results of 10 executions of the algorithms.

Kendall Tau Distance. To evaluate the quality of a produced pairwise relevance relation, we compute the *Kendall tau distance* [24] between the result of using our model and the ground-truth ranking. As shown in Equations A.1 and A.2, the Kendall tau distance is the relative number of pairs of objects for which there is a disagreement between the two rankings. Here prm_1 and prm_2 refer to the two pairwise relevance matrices encoding the rankings, and P refers to the set of the pairs of objects.

$$K(prm_1, prm_2) = \frac{\sum\limits_{(i,j) \in P} \bar{K}_{i,j}(prm_1, prm_2)}{|P|} \tag{A.1}$$

The penalty $\bar{K}_{i,j}$ is given in Equation A.2.

$$\bar{K}_{i,j}(prm_1, prm_2) = \begin{cases} 0 & \text{if } prm_1[i,j] = prm_2[i,j] \\ 1 & \text{if } prm_1[i,j] \neq prm_2[i,j] \end{cases} \tag{A.2}$$

Average Number of Assignments. The main performance measure of a crowdsourcing algorithm is the number of assignments performed by the

algorithm. The results of crowdsourcing are cheaper if they can be achieved with fewer assignments.

Average Number of Inconsistencies. Let a, b, and c be the PoIs to be ranked. As defined in Section 3.2, if in the pairwise relevance matrix, the values of the cells $[a, b]$ and $[b, c]$ are 1 and the value of the cell $[a, c]$ is 0 or -1, there is an inconsistency since we can infer $a \prec c$ using transitivity. Naturally, the goal is to reduce the number of inconsistencies in the result matrix.

Baseline Algorithm. We compare our method with an algorithm that uses majority-voting and uses a fixed number of assignments for each question. As in PointRank, for each question, three answer options are available including "incomparable". The algorithm assigns each question to a fixed number of workers, and if at least 50% of the workers give the same answer, the answer is selected as the correct answer. Otherwise, the algorithm updates the pairwise relevance matrix with value 3 to show that it is not possible to decide on this question. We choose this baseline algorithm because no other algorithms exist that contend with incomparable PoIs.

4.2 Exploring the Parameters

In this section, we study the effect of the parameters of PointRank presented in Section 3.2. We generated a total of 200 rankings: 100 total rankings and 100 partial rankings (where half of the pairs are incomparable). For each parameter that we vary, we fix the other parameters to their default values. For this set of experiments, the default values of *ina*, *pt*, *pvalue*, *minni*, and *maxni* are 5, 0.6, 0.2, 2, and 15, respectively. Worker reliability is set to 0.7.

Initial Number of Assignments

In this set of experiments, we analyze the effect of changing *ina*, the initial number of assignments.

As can be seen from Figure A.6, the Kendall tau distance decreases when the number of initial assignments increases. This is due to the fact that when the algorithm creates fewer assignments in the initial iterations, the chances of having an early consensus is quite high. In addition, there is a difference between the distance values for total and partial rankings. Our method uses the transitivity rule to infer new pairwise relevances using the questions already asked and this effects the performance for partial rankings. Because of the early consensus described above and the transitivity rule, the algorithm may decide that two objects are comparable even though they are incomparable. As the figure shows, the gap between partial and total rankings decreases when the initial number of assignments increases.
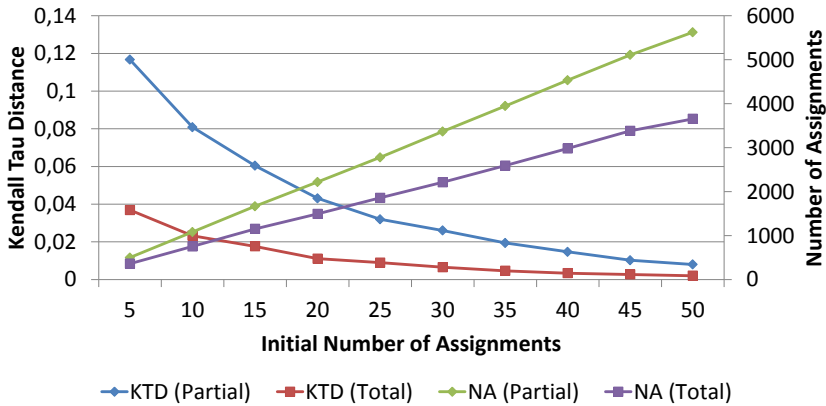
Fig. A.6: Effect of Initial Number of Assignments

As expected, Figure A.6 shows that the total number of assignments increases when the initial number of assignments increases. There is a gap between the assignment counts of total rankings and partial rankings since when the ground truth is a total ranking, the algorithm uses transitivity to decrease the number of questions, which in turn results in fewer assignments. Figure A.6 also demonstrates that our algorithm produces a pairwise relevance relation that is close to the ground truth with a reasonable number of assignments for both total and partial rankings. To illustrate, for *ina* = 25, the total number of assignments is between 2000 and 3000, which is 45 to 65 assignments for each pairwise relevance question.

The experiments also show that unless the initial number of assignments is set to a very low value, the algorithm does not produce any inconsistencies (not shown in the figure). Even for the lowest value of this parameter, the average number of inconsistencies per execution is below 0.2.

Probability Threshold

In this set of experiments, we analyze the effect of changing *pt*, the probability threshold.

Figure A.7 shows that the Kendall tau distance increases as the probability threshold increases since the algorithm cannot decide on the relevances if the probability threshold is too high.

Figure A.7 also shows that the probability threshold parameter does not significantly influence the total number of assignments. This is as expected since the algorithm stops creating new assignments for a pairwise relevance question when consensus is reached, and the probability threshold is not
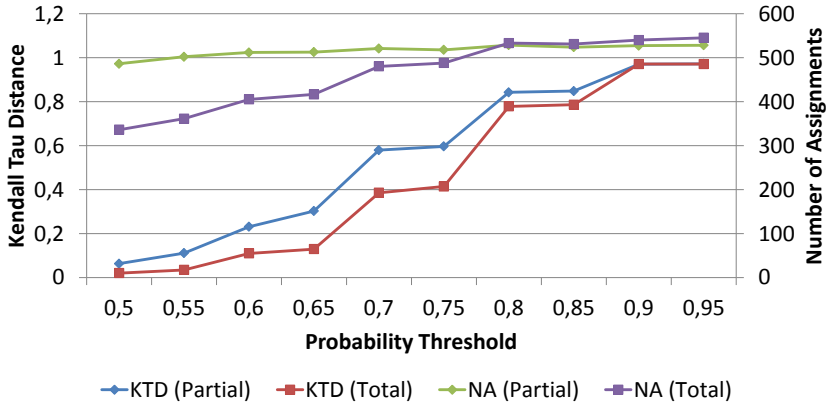
Fig. A.7: Effect of Probability Threshold

used in this stopping criterion. Nevertheless, the number of assignments for total rankings slightly increases when the probability threshold increases. This is because the transitivity property can be used less effectively when, due to high probability threshold, fewer relevances are set based on the answers from the crowd.

If the probability threshold is greater than or equal to 0.6, the algorithm does not produce any inconsistencies (not shown in the figure).

P-value

In this set of experiments, we analyze the effect of changing the *pvalue* parameter.

Figure A.8 shows that the p-value has little effect on the Kendall tau distance. However, there is a slight decrease in distance when the p-value increases since increasing the p-value decreases the chance of having an early consensus which slightly increases the quality of the result. For the same reason, when the p-value increases, the total number of assignments also increases. For high p-values, the algorithm has to create a large number of assignments to achieve consensus. The graphs also show that the difference between the number of assignments for total rankings and partial rankings is negligible.

Finally, the results of experiments show that the P-value selection does not have a noticeable influence on the number of inconsistencies.
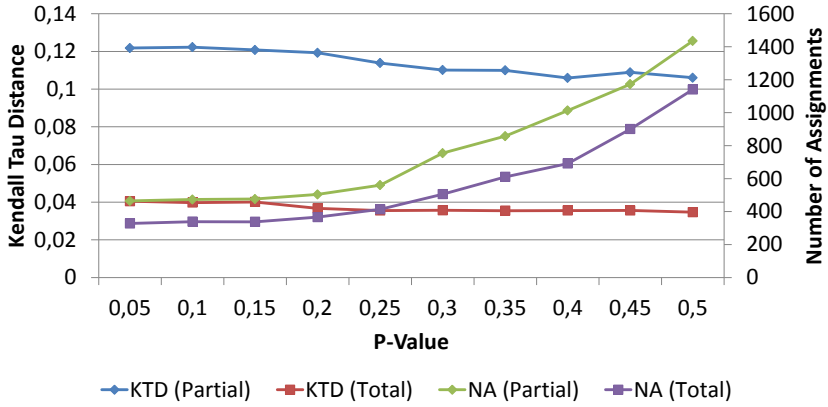
Fig. A.8: Effect of P-value

Minimum Number of Iterations

In this set of experiments, we analyze the effect of changing *minni*: the minimum number of iterations.
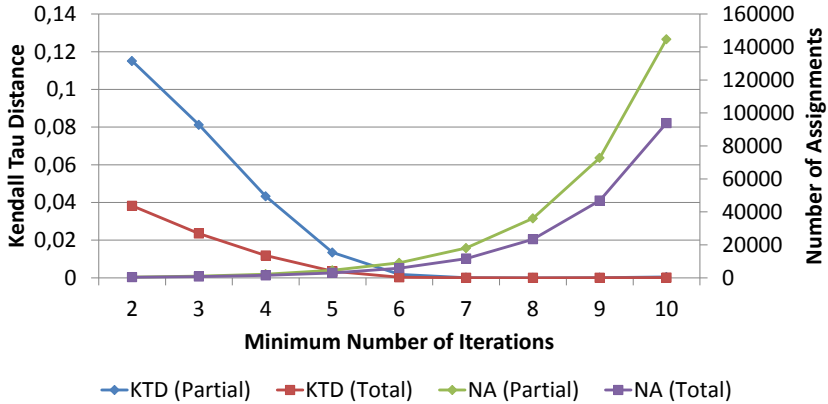
Fig. A.9: Effect of Minimum Number of Iterations

Figure A.9 shows that when the minimum number of iterations increases, the Kendall tau distance decreases. This is because a high minimum number of iterations avoids incorrectly deciding on relevances due to premature consensuses. Also, as expected, when the minimum number of iterations increases, the total number of assignments increases as well.

The experiments also show that if the minimum number of iterations is set to 3 or more, there are no inconsistencies, as the algorithm effectively avoids early consensuses leading to incorrect relevances and, thus, inconsistencies.

Maximum Number of Iterations

Our experiments show that changing the maximum number of iterations does not have a significant effect on the Kendal tau distance, the total number of assignments, and the number of inconsistencies. As the worker reliability is set to a relatively high value of 0.7, the algorithm usually reaches a consensus before reaching the maximum number of iterations.

4.3 Comparison with the Baseline Algorithm

In this section, we present the results of a comparison of PointRank and the baseline algorithm. We generated 50 total and 50 partial rankings (with half of the pairs incomparable). Based on the results of the experiments in Section 4.2, we set *ina*, *minni*, *maxni*, *pt*, and *pvalue* to 5, 4, 15, 0.5, and 0.2, respectively. The main parameter of the baseline algorithms is n, the number of assignments generated for each pairwise relevance question. We show the results for $n = 40, 70$, and 100.

Two main factors may affect the performance of the algorithms: the number of PoIs and the worker reliability. We vary these two parameters and report the Kendall tau distance, the average number of assignments, and the average number of inconsistencies for both algorithms.

Number of Places

Figures A.10 and A.11 show the performance of the algorithms when the number of PoIs is changed. PointRank produces a lower Kendall tau distance than the baseline algorithm with 40 assignments with nearly the same number of assignments. Figure A.10 also shows that our method has the same performance regardless of the number of PoIs. Figure A.12 shows that the baseline algorithm with 40 assignments causes more inconsistencies than PointRank.

Worker Reliability

In this section, the performance of PointRank and the baseline algorithm are reported for different worker reliability settings. The ability to get the ranking correctly even with less reliable workers is crucial for a crowdsourcing method since one cannot always be sure about the reliability of crowd workers.
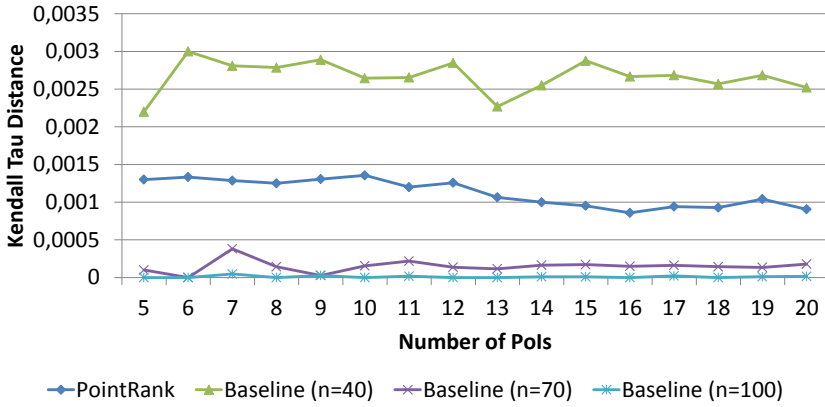
Fig. A.10: Kendall Tau Distance vs Number of PoIs
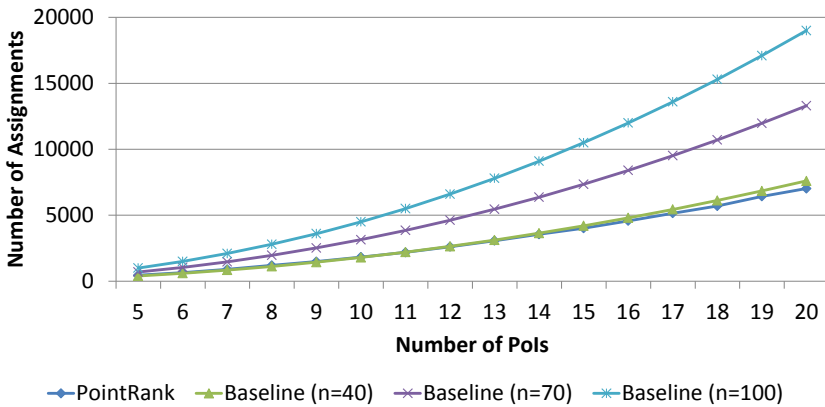
Fig. A.11: Number of Assignments vs Number of PoIs

Figures A.13 and A.14 show that PointRank produces better rankings than the baseline algorithm with 40 assignments, with the same number of assignments. It can be also seen that the Kendall tau distance of PointRank decreases when the worker reliability increases. Figure A.14 also shows that the number of assignments in our algorithm decreases when the worker reliability increases. In other words, our algorithm can tune the number of assignments according to the worker reliability. This is an expected outcome since we use a statistical significance test to check for consensus. When the workers are highly reliable, the algorithm stops assigning questions early.
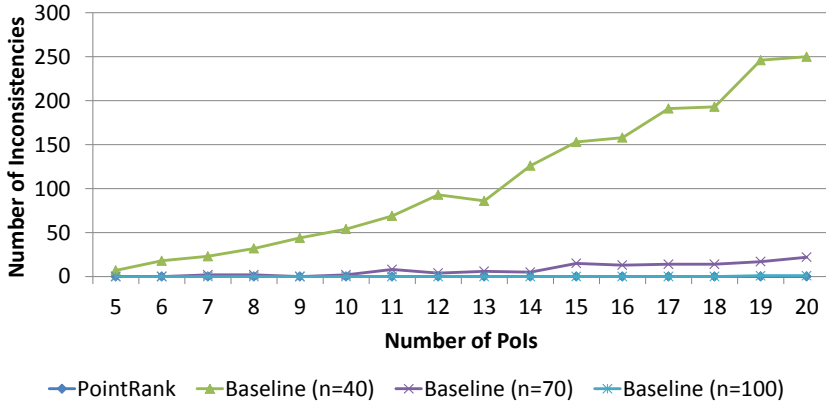
Fig. A.12: Number of Inconsistencies vs Number of PoIs
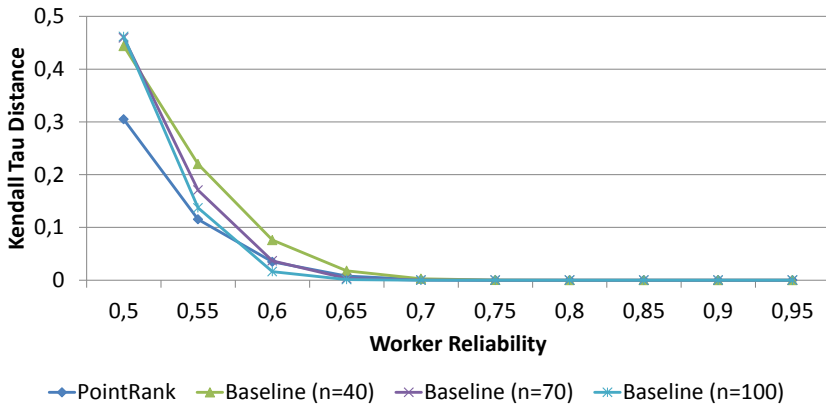
Fig. A.13: Kendall Tau Distance vs Worker Reliability

Figure A.15 shows that PointRank does not cause any inconsistencies even for the lower worker reliability values.

5 Conclusion

A spatial keyword query takes keywords and a user location as arguments and returns nearby points of interest that are relevant to the query keywords. Such queries rely fundamentally on ranking functions. We propose the Point-
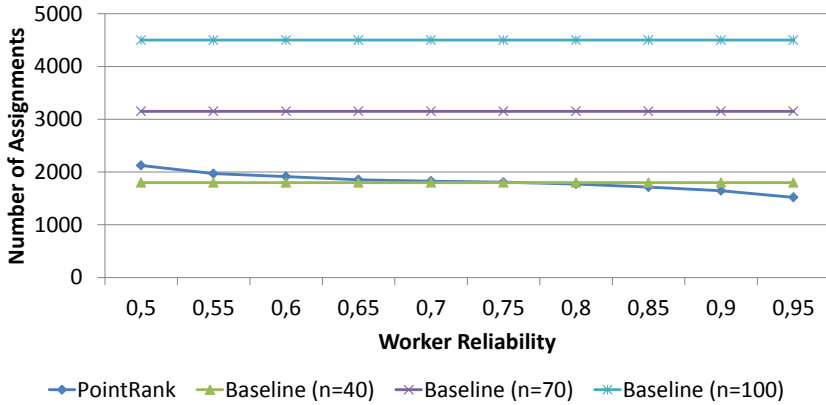
Fig. A.14: Number of Assignments vs Worker Reliability
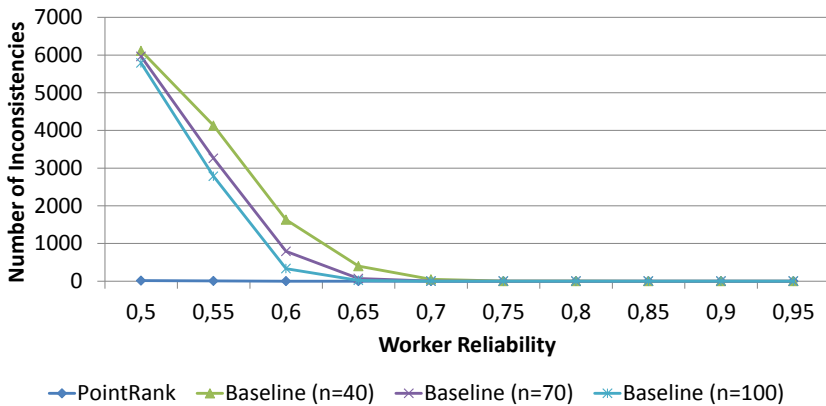
Fig. A.15: Number of Inconsistencies vs Worker Reliability

Rank model that enables evaluation of the quality of such ranking functions. PointRank synthesizes answers to crowdsourced pairwise relevance questions to rank a set of points of interest. The resulting rankings can then be used to assess the rankings produced by ranking functions. Using an innovative evaluation methodology, we evaluate the quality of the synthesized rankings achieved by PointRank, showing that PointRank is capable of producing better rankings than an approach based on majority voting.

The proposed algorithm represents a step towards the evaluation of ranking functions for the spatial keyword queries. As future work, it is of inter-

est to use the proposed model to study hypotheses about spatial keyword queries. For example, by making use of the model, it is possible to study the effect of the types of keywords in a query. A user querying for "furniture" may be willing to travel longer than a user querying for "burger", which means that the weight assigned to the distance should be different for different keywords. It is also possible to study more advanced ranking functions. For instance, to accommodate ranking functions that take into account user context such as gender and age, it is of interest to ensure that workers who evaluate answers satisfy the context assumed in the answers.

References

[1] (2015, Jun.) Google annual search statistics. [Online]. Available: http://www.statisticbrain.com/google-searches/

[2] G. Sterling. (2015, May) It's official: Google says more searches now on mobile than on desktop. [Online]. Available: http://searchengineland.com/its-official-google-says-more-searches-now-on-mobile-than-on-desktop-220369

[3] ——. (2010, Nov.) Microsoft: 53 percent of mobile searches have local intent. Available online at http://searchengineland.com/microsoft-53-percent-of-mobile-searches-have-local-intent-55556.

[4] X. Cao, L. Chen, G. Cong, C. S. Jensen, Q. Qu, A. Skovsgaard, D. Wu, and M. L. Yiu, "Spatial keyword querying," in *Proceedings of the 31st International Conference on Conceptual Modeling (ER 2012)*. Springer, 2012, pp. 16–29.

[5] J. Howe, *Crowdsourcing: Why the Power of the Crowd Is Driving the Future of Business*, 1st ed. Crown Publishing Group, 2008.

[6] J. Stoyanovich, M. Jacob, and X. Gong, "Analyzing crowd rankings," in *Proceedings of the 18th International Workshop on Web and Databases (WebDB '15)*. ACM, 2010, pp. 41–47.

[7] J. Yi, R. Jin, S. Jain, and A. Jain, "Inferring users' preferences from crowdsourced pairwise comparisons: A matrix completion approach," in *Proceedings of the First AAAI Conference on Human Computation and Crowdsourcing (HCOMP 2013)*, 2013, pp. 207–215.

[8] X. Chen, P. N. Bennett, K. Collins-Thompson, and E. Horvitz, "Pairwise ranking aggregation in a crowdsourced setting," in *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining (WSDM '13)*. ACM, 2013, pp. 193–202.

[9] J. Urbano, J. Morato, M. Marrero, and D. Martín, "Crowdsourcing preference judgments for evaluation of music similarity tasks," in *Proceedings of the SIGIR 2010 Workshop on Crowdsourcing for Search Evaluation (CSE 2010)*, 2010, pp. 9–16.

[10] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin, "Crowddb: Answering queries with crowdsourcing," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD '11)*. ACM, 2011, pp. 61–72.

[11] A. Parameswaran and N. Polyzotis, "Answering queries using humans, algorithms and databases," in *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research (CIDR '11)*, 2011, pp. 160–166.

[12] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller, "Crowdsourced databases: Query processing with people," in *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research (CIDR '11)*, 2011, pp. 211–214.

[13] T. Yan, V. Kumar, and D. Ganesan, "Crowdsearch: Exploiting crowds for accurate real-time image search on mobile phones," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys '10)*. ACM, 2010, pp. 77–90.

[14] A. Bozzon, M. Brambilla, and S. Ceri, "Answering search queries with crowdsearcher," in *Proceedings of the 21st International Conference on World Wide Web (WWW '12)*. ACM, 2012, pp. 1009–1018.

[15] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller, "Human-powered sorts and joins," *Proc. VLDB Endow.*, vol. 5, no. 1, pp. 13–24, 2011.

[16] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom, "Crowdscreen: Algorithms for filtering data with humans," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD '12)*. ACM, 2012, pp. 361–372.

[17] S. Guo, A. Parameswaran, and H. Garcia-Molina, "So who won?: Dynamic max discovery with the crowd," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD '12)*. ACM, 2012, pp. 385–396.

[18] S. B. Davidson, S. Khanna, T. Milo, and S. Roy, "Using the crowd for top-k and group-by queries," in *Proceedings of the 16th International Conference on Database Theory (ICDT '13)*. ACM, 2013, pp. 225–236.

[19] O. Alonso and R. Baeza-Yates, "Design and implementation of relevance assessments using crowdsourcing," in *Proceedings of the 33rd European Conference on Information Retrieval (ECIR 2011)*. Springer, 2011, pp. 153–164.

[20] R. Blanco, H. Halpin, D. M. Herzig, P. Mika, J. Pound, H. S. Thompson, and T. Tran Duc, "Repeatable and reliable search system evaluation using crowdsourcing," in *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '11)*. ACM, 2011, pp. 923–932.

[21] G. Kazai, J. Kamps, M. Koolen, and N. Milic-Frayling, "Crowdsourcing for book search evaluation: Impact of hit design on comparative system ranking," in *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '11)*. ACM, 2011, pp. 205–214.

[22] R. A. Bradley and M. E. Terry, "Rank analysis of incomplete block designs: I. the method of paired comparisons," *Biometrika*, vol. 39, no. 3/4, pp. 324–345, 1952.

[23] R. Schumacker and S. Tomek, "Chi-square test," in *Understanding Statistics Using R*, 2013, pp. 169–175.

[24] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee, "Comparing partial rankings," *SIAM Journal on Discrete Mathematics*, vol. 20, pp. 47–58, 2004.

References

Paper B

CrowdRankEval: A Ranking Function Evaluation
Framework for Spatial Keyword Queries

Ilkcan Keles, Christian S. Jensen, Simonas Šaltenis

Abstract

We demonstrate CrowdRankEval, a novel framework for the evaluation of ranking functions for top-k spatial keyword queries. The framework enables researchers to study hypotheses regarding ranking functions. CrowdRankEval uses crowdsourcing for synthesizing results to top-k queries and is able to visualize the results and to compare them to the results obtained from ranking functions, thus offering insight into the ranking functions.

1 Introduction

Location-based services are gaining in importance with the increase in the use of mobile, geo-positioned devices and the amount of geo-tagged web content. One core function of location-based services is top-k spatial keyword querying. Such top-k queries take a user location, keywords, and k as the arguments and return a ranked list of k points of interest (PoI) according to a ranking function [1]. Most ranking functions are a linear combination of the textual relevance of the PoIs to the query keywords and the spatial proximity of the PoIs to the query location, i.e., of the form $rank(o, q) = \alpha \cdot sp(q.loc, o.loc) + (1 - \alpha) \cdot tr(q.keywords, o.doc)$, where α, o, and q are the weighting parameter, the PoI, and the query, respectively; and sp and tr are the spatial proximity function and textual relevance function, respectively. However, existing studies provide no or little empirical evidence of the quality of the ranking functions. We believe that the lack of means of evaluating ranking functions is a major obstacle to the goal of developing high quality and advanced ranking functions.

The evaluation of ranking functions requires user feedback since there is no mathematical formulation of the best results of top-k spatial keyword queries. In fact, the best result is the one that users prefer. In this setting, the evaluation of a ranking function refers to the comparison of the ranking function with user preferences. The evaluation results show which ranking function performs better according to the user feedback. To be able to obtain feedback on the ranking functions, we use crowdsourcing [2].

We demonstrate a ranking function evaluation framework called Crowd-RankEval for top-k spatial keyword queries. The workflow of the framework is presented in Figure B.1. The framework is designed to enable researchers to evaluate the ranking functions used for spatial top-k queries. The user must choose a small set of queries on which the ranking functions are evaluated. The user must also supply a query result for each query and ranking function. Given this input, the framework synthesizes a ranking of the PoIs contained in supplied query results for each query by asking pairwise relevance questions to the crowdsourcing workers. A pairwise relevance question contains a query and a pair of PoIs and it asks a worker which of the two PoIs is more relevant to the query. Upon completion of the crowdsourcing, the framework synthesizes and displays a ranking as well as a comparison between the synthesized ranking and the rankings produced by the ranking functions. Thereby, the framework offers insight into how well the ranking functions perform.

The framework has four modules: the user interface module, the data preparation module, the PointRank module, and the evaluation module. The user interface module is the entry point of a user to the framework. The data
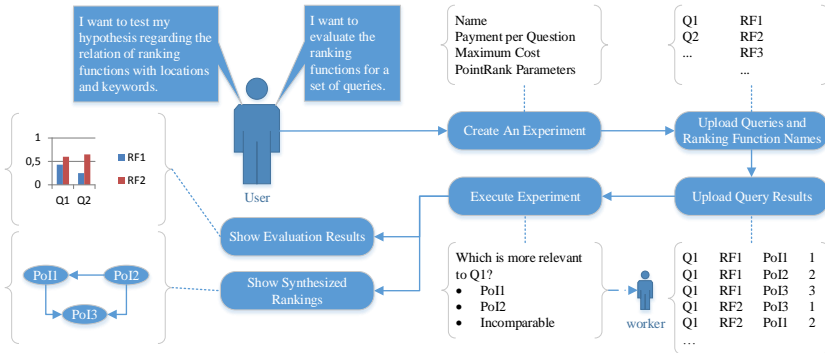
Fig. B.1: Workflow of CrowdRankEval

preparation module allows a user to upload data. It also preprocesses the data to be able to perform the evaluation. The PointRank module simply provides an implementation of the PointRank algorithm that synthesizes the rankings for the queries using crowdsourcing [3]. Our framework is built to use CrowdFlower[1] as the crowdsourcing platform. Finally, the evaluation module is responsible for performing the evaluation of the ranking functions and for visualizing the synthesized rankings and the results of the evaluation.

In summary, the framework to be demonstrated contributes in these aspects:

- The framework is able to evaluate ranking functions for top-*k* spatial keyword queries, and to visualize the results.

- The framework provides an implementation of the PointRank algorithm to synthesize rankings using crowdsourcing.

- The framework is built to connect with CrowdFlower to publish crowdsourcing tasks.

The rest of the paper is arranged as follows: Section 2 presents the framework and gives detailed information regarding the modules, and Section 3 presents the workflow and the demonstration details. Section 4 concludes the paper.

---

[1]http://www.crowdflower.com/

2   The CrowdRankEval Framework

The building block of the framework is that of an experiment. An experiment is defined by a set of queries and ranking functions. To create an experiment, the user must also specify parameters used by the PointRank algorithm. To enable PointRank to evaluate the ranking functions, the user should also upload the corresponding query results. After uploading the results for all queries, the user is able to start the evaluation task. As an example use case, the framework can be used to determine the best weighting parameter (α) for the ranking function given in Section 1 for a specific set of query keywords. To do so, the researcher must upload the queries and their results when using the ranking function with different weighting parameters. According to the evaluation results, the researcher can decide on the best weighting parameter.

The framework is developed using Javascript, HTML, CSS, and PHP. We also used vis.js[2] and Highcharts[3] for visualization purposes.

2.1   User Interface Module

The user interface module is the module that enables interaction with the users. This module is responsible for receiving input and for displaying the experiment details, query details, and evaluation results. It uses HTML elements as well as visualization libraries to handle graphical elements.

2.2   Data Preparation Module

The data preparation module handles the data upload of queries, ranking function names, and the query results. For the upload of queries and ranking function names, it checks whether the uploaded data has the required attributes. For the upload of the query results, it checks whether the corresponding query and ranking function name are uploaded in the current experiment. This module is also responsible for the preprocessing of the data. It creates the merged result set for each query. The merged set is the union of the results of all ranking functions for the corresponding query.

2.3   PointRank Module

This module executes PointRank on the merged result set for each query. First, it checks whether the experiment is ready to execute. In other words, it checks whether each query has results for each ranking function. Then it executes PointRank to synthesize the ranking for each query. To publish

---

[2]http://visjs.org/
[3]http://www.highcharts.com/

crowdsourcing tasks for the execution of PointRank, the module is designed to use the CrowdFlower platform.

PointRank Algorithm

PointRank synthesizes rankings for top-*k* spatial keyword queries using crowd-sourcing. The general flow of the algorithm is given in Figure B.2.
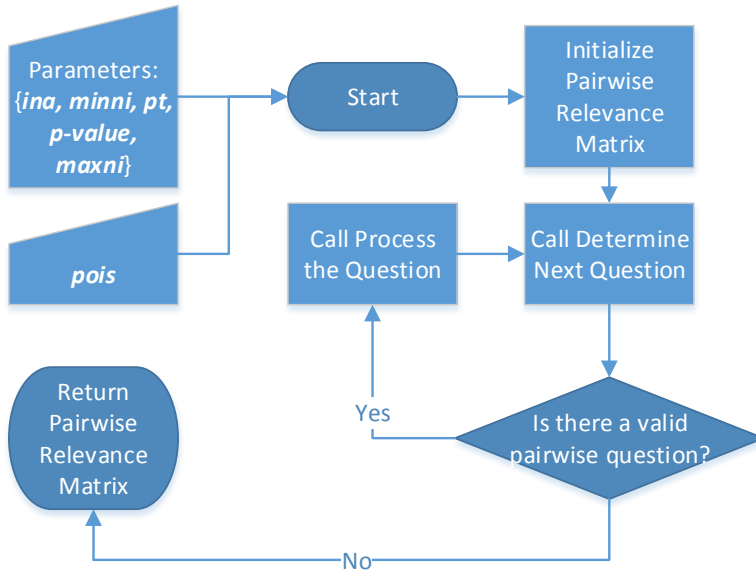
Fig. B.2: General Flow of PointRank

PointRank makes use of pairwise relevance questions that are defined by pairs of PoIs and asks workers which one of the two PoIs in a pair is more relevant to the specified query. The worker can give three different answers: the first PoI is more relevant, the second PoI is more relevant, and the PoIs are incomparable.

Parameter *pois* is the list of PoIs to be ranked. In our framework, it is the merged result set which is created by the data preparation module.

As shown in Figure B.2, the algorithm has two phases: determine the next question and process the question. If each pairwise relevance question is processed, it returns the synthesized partial ranking. To determine the next pairwise relevance question, the algorithm uses a gain function that de-

termines how many further questions can be eliminated with the result of the question. The pairwise relevance question with the maximum gain is chosen as the next question. In the processing phase, the algorithm employs an iterative approach. In each iteration, it assigns the question to the workers. The algorithm uses the initial number of assignments parameter *ina* to determine the number of assignments. It iterates until the iteration number reaches the minimum number of iterations parameter *minni*. Then it checks whether there is a consensus between two iterations. To check for consensus, PointRank uses the *Chi-square(χ^2) test* [4]. The parameter *pvalue* is used as a threshold value for *p-value* in the Chi-square test. If there is a consensus, the probability values of each possible answer is computed. The probability value for an answer is the ratio of the number of workers who gave this answer to the number of all workers who answered the question. The algorithm checks whether the answer with maximum probability exceeds the probability threshold parameter *pt*. If so, it is chosen as the answer to the pairwise relevance question. If there is no consensus, the algorithm continues until there is consensus or the number of iterations reaches the maximum number of iterations parameter *maxni*.

2.4 Evaluation Module
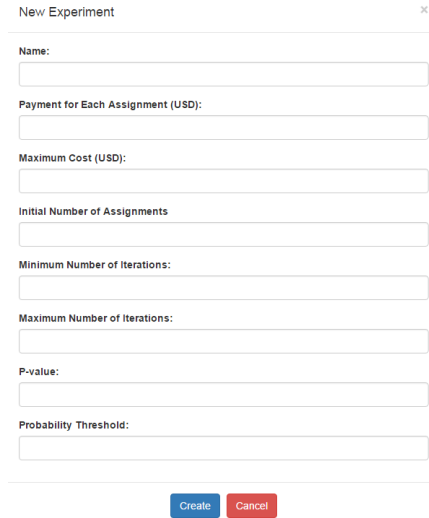
This module handles the evaluation of ranking functions with respect to the synthesized rankings. To evaluate the ranking functions, we compute *Kendall tau distance* [5] between the input rankings and the synthesized ranking. The Kendall tau distance is defined as the proportion of the number of pairs on which two rankings disagree over the total number of pairs. First, for each query and ranking function, the Kendall tau distance is computed between the input ranking and the synthesized ranking. Then, the evaluation module presents Kendall tau distance for four different levels: experiment, query, keyword, and location. At the experiment level, the average Kendall tau distance of a ranking function over all queries is presented. At the query level, the Kendall tau distance for each query is presented. At the keyword and location levels, the average distance of a ranking function is presented for each keyword and each location, respectively. We decided to have these levels since we want the users be able to test hypotheses regarding different queries, keywords, and locations.

3 Workflow and Demonstration Details

3.1 Workflow

As shown in Figure B.1, the workflow of the framework consists of six steps.

Step 1: Creating an Experiment. First, the user must provide the amount to be paid to workers per pairwise relevance question they answer, the maximum total amount available for payments, the parameters of PointRank, as shown in Figure B.3.

Fig. B.3: New Experiment

Step 2: Uploading the Queries and Ranking Function Names. Having created an experiment, the user uploads queries and ranking function names using the form shown in Figure B.4. The input files should be CSV files with required attributes. The attributes of a query are query ID, keywords, latitude and longitude. The attributes of a ranking function name are ranking function ID and name.

Step 3: Uploading the Query Results. In this step, the user is expected to upload the results of applying each query and ranking function to an underlying data set of PoIs. The query result file should also be a CSV file with the following attributes: Query ID, Ranking function ID, Rank, PoI ID, PoI name, PoI latitude, PoI longitude, and PoI description. The framework checks whether the experiment contains the corresponding query and the ranking function. The rank attribute shows the rank of the PoI with respect to the ranking function.

Step 4: Executing the Experiment. After uploading the data, the user can start the experiment execution. During this step, the framework is designed to execute PointRank on CrowdFlower to synthesize the ranking for each query.

Step 5: Showing Synthesized Rankings. When the results are ready, the user can show synthesized rankings for each query. The synthesized ranking

Fig. B.4: Upload Form

is presented as a directed graph as shown in Figure B.5 since the output of PointRank is a partial ranking. A directed edge from one PoI to another means that the first is less relevant to the query than the second. A dashed edge means that the PoIs are incomparable. If there is no edge between a pair of PoIs, the algorithm cannot come to a conclusion about the pairwise relevance of the PoIs.

Fig. B.5: View Synthesized Ranking

Step 6: Showing the Evaluation Results. After the execution is finished,

the user can also see results of the ranking function evaluation. The evaluation result is shown as a column graph as displayed in Figure B.6. Each column corresponds to a ranking function, and the value shown is the Kendall tau distance between the ranking function and the synthesized ranking. The user is able to view the evaluation result at four different levels: experiment, query, keyword, and location. To change the results shown, the user should select the corresponding result type.
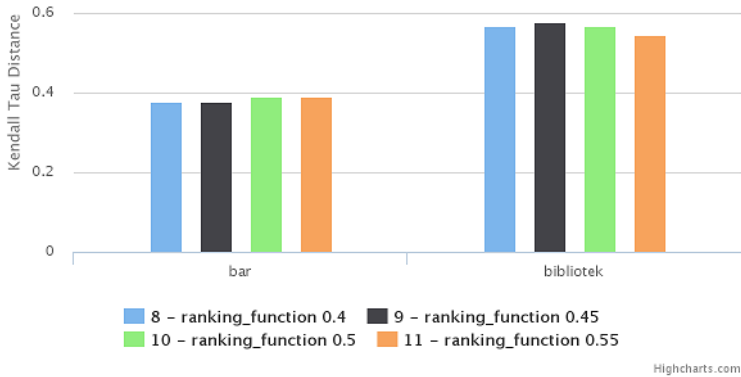
Fig. B.6: Ranking Evaluation

3.2 Demonstration Details

We will demonstrate scenarios where we study a hypothesis regarding keywords. The hypothesis is "In queries with keywords targeting places for eating and drinking, spatial proximity should be weighted higher than textual similarity. In contrast, for queries with keywords targeting specific types of PoIs like churches or libraries, textual similarity should be weighed higher than spatial proximity."

First, we create an experiment by providing the required parameters. Then we will upload 4 queries with 2 different locations and 2 different keywords and 5 ranking function names. One of the locations is in the city center of Aalborg, Denmark, and the other is at the Aalborg University campus. We choose to have different locations to minimize the effect of the location on the evaluation results. The ranking functions are linear combinations of textual relevance and spatial proximity with different weight values. After uploading the data, we upload the query results for each query. Then, we execute the experiment. In the demonstration, since publishing tasks and getting results by means of crowdsourcing takes time, we execute a simulation of PointRank to be able to synthesize rankings. The simulation will get the an-

swers from simulated workers instead of from real crowdsourcing workers. The simulated workers will answer pairwise relevance question according to a generated ground truth ranking and a reliability value. Lastly, we will demonstrate the synthesized rankings and the evaluation results produced by the framework.

4 Conclusion

We demonstrate a novel ranking function evaluation framework for top-*k* spatial keyword queries. The framework is particularly useful for top-*k* spatial keyword queries since it considers incomparability of the points of interest, but it can also be used for general top-*k* queries. The framework includes an implementation of PointRank that synthesizes rankings of points of interest using crowdsourcing. It also provides visualization for the synthesized rankings and comparisons between the synthesized ranking and those obtained from ranking functions. As future work, the framework can be extended to include preprocessing and post-processing techniques for crowdsourcing tasks. For example, a qualification test may be introduced to determine whether workers are qualified to complete the tasks, and post-processing may be used to exclude malicious workers after getting the answers.

5 Acknowledgments

References

[1] X. Cao, L. Chen, G. Cong, C. S. Jensen, Q. Qu, A. Skovsgaard, D. Wu, and M. L. Yiu, "Spatial keyword querying," in *Proceedings of the 31st International Conference on Conceptual Modeling (ER 2012)*. Springer, 2012, pp. 16–29.

[2] J. Howe, *Crowdsourcing: Why the Power of the Crowd Is Driving the Future of Business*, 1st ed. Crown Publishing Group, 2008.

[3] I. Keles, S. Saltenis, and C. S. Jensen, "Synthesis of partial rankings of points of interest using crowdsourcing," in *Proceedings of the 9th Workshop on Geographic Information Retrieval (GIR '15)*. ACM, 2015, pp. 15:1–15:10.

[4] R. Schumacker and S. Tomek, "Chi-square test," in *Understanding Statistics Using R*, 2013, pp. 169–175.

References

[5] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee, "Comparing partial rankings," *SIAM Journal on Discrete Mathematics*, vol. 20, pp. 47–58, 2004.

Paper C

Crowdsourcing Based Evaluation of Ranking
Approaches for Spatial Keyword Querying

Jinpeng Chen, Hua Lu, Ilkcan Keles, Christian S. Jensen

Abstract

Spatial keyword querying has attracted considerable research efforts in the past few years. A prototypical query takes a location and keywords as arguments and returns the k objects that score the highest according to a ranking function. While different scoring functions have been used, how to compare different ranking functions for spatial keyword querying still remains an open question with little investigation. We propose a crowdsourcing-based approach to evaluate and compare ranking functions for spatial keyword search. Given two ranking functions f_1 and f_2, we use a matrix to model all possible binary questions regarding the different results produced by f_1 and f_2. We propose a multi-step process to reduce the number of binary questions, identifying the most important questions to ask. Further, we design a crowdsourcing model that obtains the answers to those important binary questions from crowd workers. We also devise a global evaluation process that is able to quantitatively compare f_1 and f_2 based on a multitude of answers received. According to the results of empirical studies using real data, the proposed approach is efficient and able to draw reliable conclusions in comparing ranking functions for spatial keyword search.

1   Introduction

With the rapid adoption of geo-enabled devices (e.g., smartphones) and the use of location-based social networks (LBSNs, e.g., Foursquare and Facebook Places), users can post geo-tagged information anytime and anywhere, and we are witnessing a proliferation of geo-tagged web content. For instance in 2014, 164 million active users access Twitter from mobile devices each month, while 425 million mobile users access Facebook [1]. While these developments bring convenience to users, they also make it increasingly hard for users to find relevant content. Motivated by this development, research on spatial keyword search has expanded markedly in the past few years. Spatial keyword search is effective in helping users find interesting and relevant content in a range of settings [2–5].

Many different spatial keyword queries [6–8] have been proposed and studied, but most studies focus on query processing for the different types of queries. Most queries considered are distance-sensitive [1], i.e., they concern both the spatial distance and the textual relevance between a query and the set of objects of interest. Typical functionality involves returning the top-*k* spatial objects that are most relevant to a query containing both a location and a set of query keywords. Many different ranking functions can be used to find the top-*k* spatial objects in such queries.

A very natural question to ask is, which ranking function gives the better results for a spatial keyword query? In this study, we aim to provide means of answering this question. The availability of such means is important since a better ranking function increases user satisfaction for location based services. However, providing such means is not straightforward since it is not possible to understand which ranking function performs better without information regarding user preferences for a specific query. We do not consider non-crowdsourced methods because such methods are more likely to fail to find local knowledge compared with crowdsourcing that involves many workers, because such methods are more likely to be biased, and because such methods may require tedious offline surveys or expensive specialized systems. In contrast, Foursquare, TripAdvisor, and Google Places make use of the wisdom of the crowd to associate a rating with a point of interest (POI). However, such ratings cannot be used to decide whether one POI is better than another for a specific spatial keyword query because the rating is in general unrelated to the location in the query. In other words, a POI with a better rating would not be a good option for a query if the POI is distant from the query location.

Thus, we propose a crowdsourcing-based approach. Crowdsourcing can provide solutions for complex tasks that are impossible or too hard to model by means of computer programs, but are relatively easy to accomplish using collective human intelligence. The problem of evaluating ranking functions

falls into this category. For example, it is very difficult for an algorithm to tell whether restaurant A is better than restaurant B for a search request like "find the best pizza shop closest to my current position in Aalborg." It is much easier to draw a conclusion if we ask the question of sufficiently many people who are familiar with Aalborg.

Based on this important observation, we leverage crowdsourcing to collect empirical knowledge to help evaluate ranking functions in spatial keyword querying. Crowdsourcing platforms make it possible to apply a qualification test for workers, which can be used to make sure that workers are familiar with the relevant region. A crowdsourcing based solution faces three major challenges: (1) How to produce questions that can be answered easily by crowdsourcing workers? (2) How to interpret and combine the answers received from the workers to achieve an integrated evaluation? (3) How to determine which ranking function is better based on the evaluation? We design three components, namely a matrix-based question model, a crowdsourcing-based evaluation, and a global evaluation, to tackle the three challenges. The system framework of our approach is shown in Figure C.2.

Given a spatial keyword query q_i, two ranking functions f_1 and f_2 produce two top-k lists l_1 and l_2. We first map the complex comparison of l_1 and l_2 into a series of easy-to-understand binary questions. These questions are represented in a matrix where each cell corresponds to a question like "Is object A better than B for query q_i?". Here, q_i is a concrete query. We use binary questions instead of list-wise questions where workers are asked to provide a complete ranking of the given list. It is likely to be difficult and time-consuming for workers to provide complete rankings since it requires workers to carry out many POI comparisons. Moreover, they cannot provide a proper ranking even if they are not familiar with one of the POIs in the question. We also propose methods to reduce the number of binary questions to ask, as each question comes at a cost in crowdsourcing. Subsequently, we publish a set of selected binary questions to a crowdsourcing platform and obtain answers from workers. Based on those answers, we utilize matrix factorization to obtain answers to all questions in the matrix. Based on the answers, we are able to tell which of l_1 and l_2 is better for q_i. Accordingly, the ranking function that gives the better list is preferred for query q_i. We repeat the process for M queries, and the ranking function that scores overall highest is regarded as the globally better one.

The paper makes the following contributions:

- We establish a crowdsourcing-based framework to evaluate ranking functions used in spatial keyword querying.

- We propose a matrix-based binary question model for two ranking lists generated by two ranking functions for a spatial keyword query. The model is able to control the number of questions to be sent to a crowd-

sourcing platform according to a given budget and the platform's cost model.

- We design a crowdsourcing model that can transform the crowd workers' answers to the binary questions into a comprehensive ranking of the two ranking lists.

- We present a global evaluation method that aggregates the comparison results of the ranking list pairs for a set of spatial keyword queries, which enables us to conclude which ranking function is the better one.

- We use a real data set to conduct experimental studies. The results show that the proposed approach is efficient and effective in evaluating ranking functions in spatial keyword querying.

The rest of the paper is organized as follows. Section 2 gives the problem statement and provides an overview of the proposed framework. Section 3 elaborates on the question model for crowdsourcing. Section 4 details the crowdsourcing model. Section 5 presents the global evaluation for ranking functions. Section 6 reports on the empirical studies. Section 7 reviews related work. Section 8 concludes and discusses future work directions.

2 Problem Formulation and Framework

2.1 Definitions and Problem Statement

The data considered can be abstracted as spatial objects, defined as follows.

Definition C.1 (Spatial Object). *A spatial object o is represented as a 2-tuple $o = (loc, KW)$, where o.loc is o's geo-location and $o.KW = \{kw_1, kw_2, ..., kw_n\}$ is a set of keywords that describe the object. The value of n is not fixed here, and spatial objects can have different number of keywords.*

The spatial objects, or POIs, in a region of interest, e.g., a city like Los Angeles, form a set of spatial objects. In many cases, users want to find a reasonable number of objects that are most relevant or interesting with respect to a location and a set of keywords that describe the user's needs. Thus corresponds to issuing a top-*k* spatial keyword query [2].

Definition C.2 (Top-*k* Spatial Keyword Query). *Given a set O of spatial objects, a top-k spatial keyword query q is formulated as $q = (loc, KW, k)$, where loc is a query location, KW is a set of query keywords, and k is an integer. Query q returns k objects from O that are* most relevant *to the given query location and keywords.*

From a service provider's perspective, a query $q = (loc, KW, k)$ is processed by employing a *ranking function* that quantifies the objects' relevance to the query and by returning the k best objects. A prototypical ranking function scores an object with respect to its distance from the query location and its textual relevance to the query keywords.

Definition C.3 (Ranking Function). *Given a set O of spatial objects, a ranking function $f : O \rightarrow [0,1]$ returns a score between $[0,1]$ for an object. We assume that a lower score means that the object is more relevant to the query.*

Many ranking functions can be defined for top-k spatial keyword queries. In the context of location-based services (LBS), a service provider may have a set of ranking functions that may be used for different inputs or scenarios. Also, different service providers may employ different (sets of) ranking functions in order to deliver distinctive services. For the sake of simplicity, Figure C.1 illustrates a scenario where a single service may have multiple ranking functions, each of which can be used to process a top-k spatial keyword query.
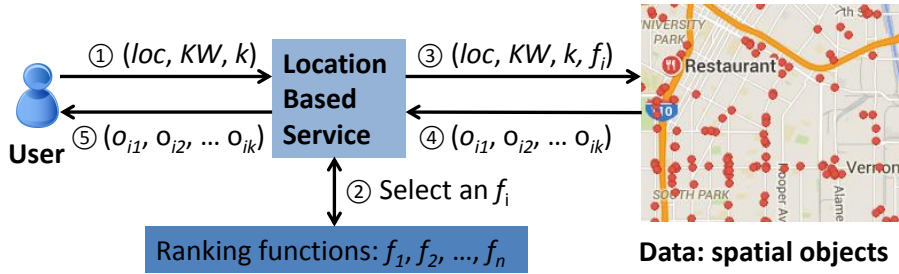
Fig. C.1: Top-k Spatial Keyword Query

Our research does not aim to define new ranking functions, but aims instead to evaluate given ranking functions. For this purpose, we need to involve multiple object sets and different queries. For a given set of objects and a particular ranking function, many queries can be issued. Each query returns a particular top-k list determined by the ranking function employed.

Definition C.4 (Top-k Ranking List). *Given a set O of spatial objects, a query $q = (loc, KW, k)$ $(k \leq |O|)$ processed using a ranking function f returns a top-k ranking list $L_{O,q,f} = (o_1^f, \ldots, o_k^f)$ where*

- $f(o_1^f) \leq \ldots \leq f(o_k^f)$*, and*

- $\forall o \in O \setminus L_{O,q,f} \ (f(o) \geq f(o_k^f))$.

Research Problem. We tackle the problem of evaluating spatial keyword ranking functions using crowdsourcing. Without loss of generality, we focus on evaluating two ranking functions on one spatial object set.

2.2 Framework

Given a spatial object set O and two ranking functions f_1 and f_2, our solution evaluates f_1 and f_2 by considering M ($M > 1$) top-k spatial keyword queries $Q = \{q_1, \ldots, q_M\}$. Without loss of generality, we assume all queries in Q have the same value for k. For ranking function f_1, we get M top-k ranking lists, i.e., $L_{O,q_1,f_1}, \ldots, L_{O,q_M,f_1}$. Likewise, we get M top-k ranking lists $L_{O,q_1,f_2}, \ldots, L_{O,q_M,f_2}$ for ranking function f_2.

Figure C.2 shows the framework of our solution. It consists of M *Local Evaluations*, one for each of the queries in Q, and a *Global Evaluation* that draws the final conclusion based on the M local evaluations.

A local evaluation has three components. The *Ranking Process* accepts a top-k spatial keyword query and outputs two top-k ranking lists l_1 and l_2 by using ranking functions f_1 and f_2, respectively.

Receiving the two ranking lists, the *Question Model* employs a learn-to-rank method to compute an ordered k-list l_{12} that aims to capture important characteristics of l_1 and l_2. Subsequently, the question model uses a $2k$ by $2k$ matrix to represent element pairs from the two lists. It is used for generating considerably fewer than $2k \times 2k$ binary questions, taking into account the given cost budget and three lists. These binary questions are of the form "Do you think place A is better than place B for query q?". The maximum number of objects considered is $2k$, which happens when l_1 and l_2 are disjoint. However, if l_1 and l_2 have objects in common, fewer than $2k$ objects are considered.

The *Crowdsourcing Based Evaluation* component publishes the binary questions on a crowdsourcing platform. Answers from crowd workers are then collected to form a partially instantiated $2k \times 2k$ matrix, which is transformed to a fully instantiated matrix through factorization. The elements in the complete matrix are counted to produce a crowdsourced top-k list l_f.

List l_f as well as the original l_1 and l_2 are passed to the global evaluation, where l_1 and l_2 are compared in terms of their similarity and consistency with l_f. Function f_1 is assigned a score if l_1 is deemed better than l_2; otherwise, f_2 is assigned a score. All such scores for the M queries are given to a score aggregator that draws a conclusion based on the aggregated scores given to functions f_1 and f_2.
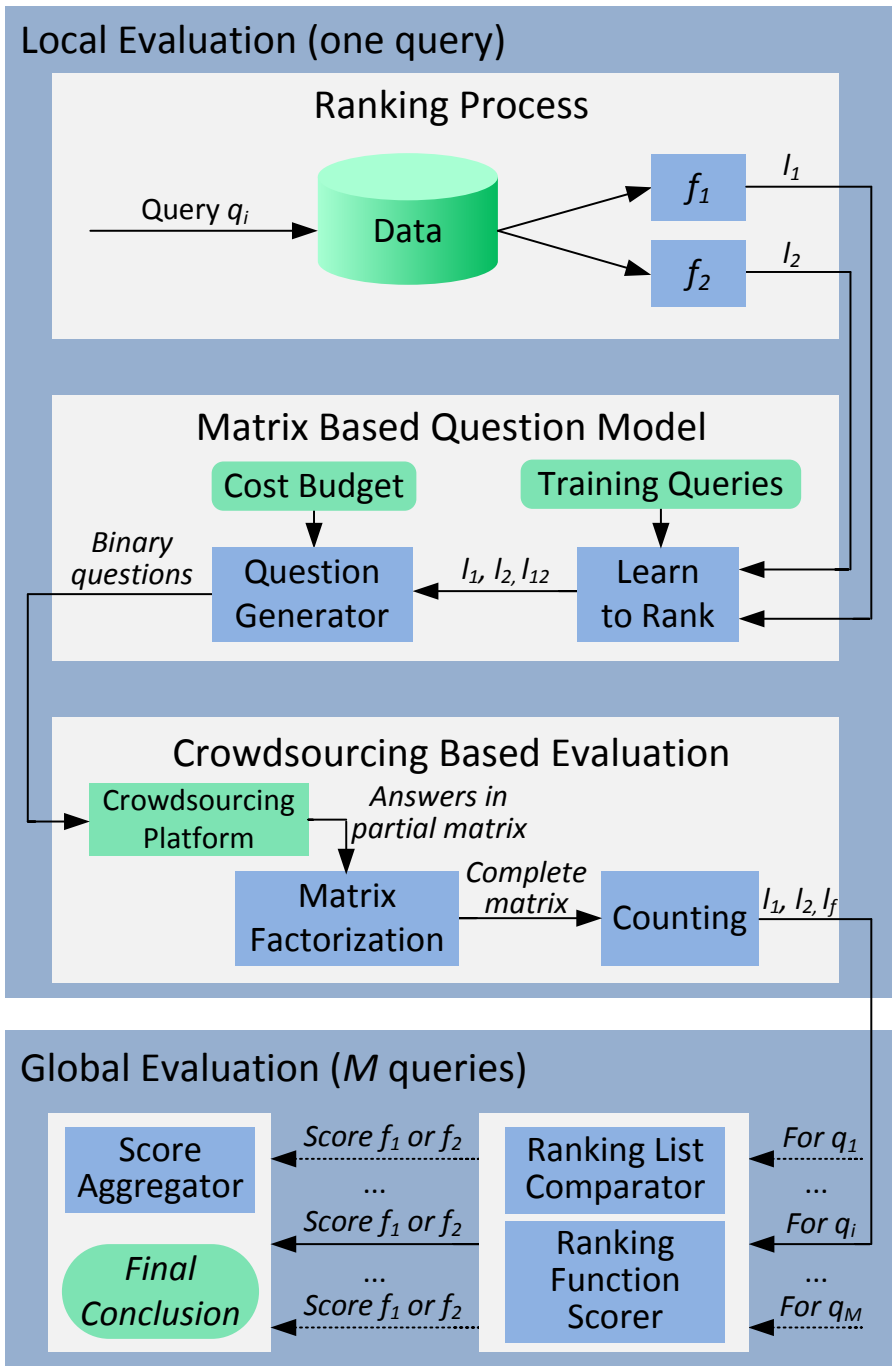
Fig. C.2: System Framework

2.3 Running Example

To better understand the process of generating the binary questions, producing the final list and evaluating two lists, we will tell a case throughout this paper. In the following sections, we will show some examples by which we can summarize the general ideas behind the algorithm.

Table C.1: Properties of dataset. For simplification, we use upper letters in the ID column to represent the spatial objects in the rest part of the paper.

ID	Spatial Object	Location	Keywords
A	Maxim Bar	57.05108, 9.9184	food, bar, Jomfru Ane Gade, Borgergade, maxim
B	D'Wine Bar	57.04828, 9.91696	food, bar, Algade, Boulevarden, Wine
C	Tempo Bodega	57.05112, 9.91646	bar, Borgergade, Jomfru Ane Gade, Boulevarden, Vesterbro
D	Joe & the Juice	57.04821, 9.92145	cafe, coffee, Nytorv, Østeraagad, Jyllandsgade
E	Cafe Azzurra	57.04287, 9.91731	cafe, Jyllandsgade, busterminal, Bornholmsgade, John F. Kennedys Plads
F	Starbucks Coffee	57.04833, 9.92247	cafe, coffee, Bornholmsgade, Salling Stormagasin, Starbucks
G	Visse Pizzaria	56.99109, 9.93112	fastfood, E45, vissevej, Taha Wais, visse
H	Amantes	57.0718, 9.92647	fastfood, E45, 9440, Liam Lund, Forbindelsevejen
I	McDonald's	57.0702, 9.94647	fastfood, 9440, McCafe, Nr.Uttrup Torv, music
J	Hasseris Pizza	57.03565, 9.88452	pizza, music, Fyrrebakken, Thulebakken, Hasseris
K	The Wharf	57.05144, 9.91634	music, pub, Jomfru Ane Gade, beers and ales, fantastic
L	Skotten	56.99117, 9.93107	food, bar, music, vissevej, Skotten
M	Munken	57.05209, 9.90775	food, pub, Kastetvej, restaurent, munken

To illustrate, we show 13 spatial objects located in a query region that covers part of Aalborg City. From these 13 spatial objects, we get four pairs of lists when searching four queries leveraging two ranking functions. The dataset and the search results are shown in Table C.1 and Table C.2, respectively.

Table C.2: Top-5 lists produced by ranking functions

ID	Query	Ranking Function	List
1	food bar	f1	A,B,C,D,E
		f2	C,B,A,F,G
2	food cafe	f1	D,E,F,G,H
		f2	F,E,D,E,I
3	fastfood	f1	G,H,I,J,A
		f2	I,H,G,J,M
4	food pub	f1	A,D,G,J,K
		f2	H,E,B,I,L

3 Question Model for Crowdsourcing

This section details the determination of the questions to be asked to the crowd. The whole process is shown in Figure C.3.

3.1 Matrix Based Question Model

Suppose the ranking process outputs two top-k ranking lists $l_1 = \langle a_1, a_2, \ldots, a_k \rangle$ and $l_2 = \langle b_1, b_2, \ldots, b_k \rangle$ for a given query q and two ranking functions f_1 and f_2.

A naive question model is to use a $2k \times 2k$ matrix $\mathcal{M}$, where the columns (rows) consist of $a_1, a_2, \ldots, a_k, b_1, b_2, \ldots,$ and b_k. Each cell $\mathcal{M}[i, j]$ corresponds to a question like "Is a_i better than b_j?" or "Is b_i better than a_j?". This way, we need to ask $4k^2$ binary questions in order to evaluate lists l_1 and l_2, which is an impractically large and costly number of questions to ask. Also, it may be hard to get sufficiently many crowdsourcing workers to answer so many questions.

Therefore, we propose a cost-aware question model that is able to optimize the result quality under a given cost constraint. In our setting, we employ $|W|$ workers, each of which is expected to answer N_{bq} binary questions. For each binary question a worker answers, we need to pay a cost of C [1]. As a result, we need to pay $C \cdot |W| \cdot N_{bq}$ to accomplish an evaluation task using the crowdsourcing platform. As C is a constant for a selected crowdsourcing platform, we focus on finding a sufficiently small set W and a sufficiently small number N_{bq}.

Essentially, not all cells in the $2k \times 2k$ matrix $\mathcal{M}$ are of equal importance. We need to prioritize the cells in $\mathcal{M}$ and only ask binary questions corre-

---

[1]This cost C may be divided according to a particular ratio between a worker and the crowdsourcing platform. Nevertheless, the concrete low level details do not affect the cost that we have to pay.
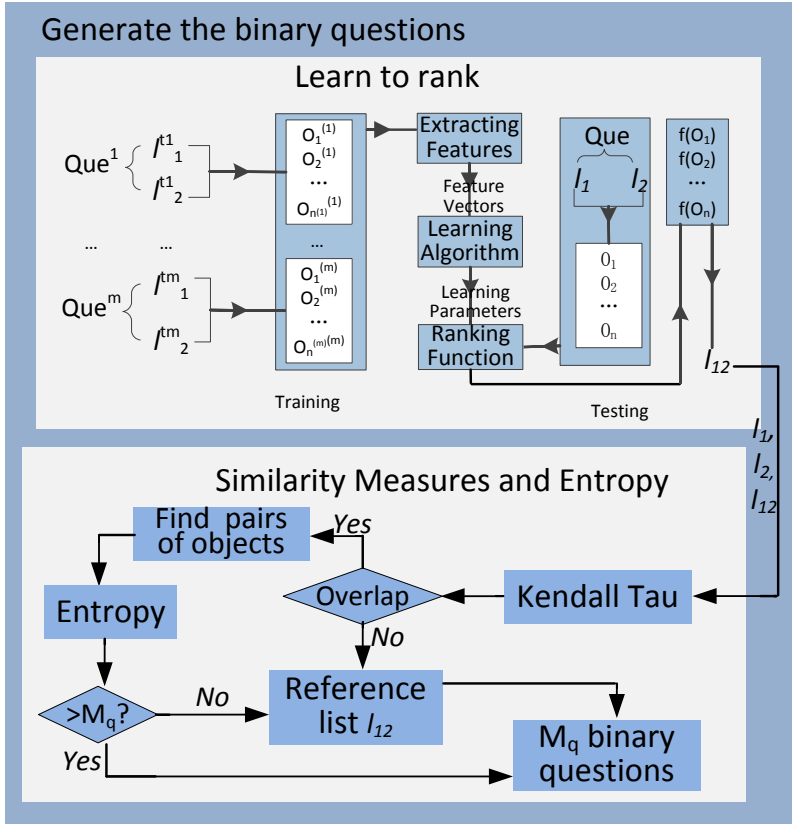
Fig. C.3: The Process of Generating Binary Questions

sponding to the important cells.

A preliminary step is to use a triangular matrix instead of the original full matrix. In particular, we only use the lower triangular part of $\mathcal{M}$, assuming that cells $\mathcal{M}[i,j]$ and $\mathcal{M}[j,i]$ are equivalent. This means that we regard questions "Is c_i better than c_j?" and "Is c_j better than c_i?" as the same. In other words, a "yes" to the first question is equivalent to a "no" to the second question. This way, we only look at cells $\mathcal{M}[i,j]$ where $i \geq j$. This yields $N_{bq} = \frac{2k(2k-1)}{2} = k(2k-1)$.

We proceed to further reduce N_{bq}, i.e., decrease the number of binary questions to ask.

3.2 Binary Question Generation

We generate a sufficiently small number of binary questions in four steps. First, we use a learn-to-rank method to construct a list l_{12} of k most important spatial objects from the two top-k lists l_1 and l_2. Second, we compute the similarity between l_{12} and l_1 (l_2) to decide which objects in l_1 (l_2) to ignore in binary questions. Third, we use spatial object entropy to control the number of binary questions. Fourth, we generate the binary questions for the crowdsourcing platform.

A List by Learn-to-Rank

The ListNet method [9] can solve the ranking problem on the lists of objects, and the features we extract are similar to those the ListNet method requires, so we adopt the ListNet method in this work. As shown in the upper part of Figure C.3, this step has two processes, training and ranking. The training accepts a set of queries $Que = \left\{ Que^{(1)}, Que^{(2)}, ..., Que^{(m)} \right\}$. Each query $Que^{(i)}$ results in two ranking lists generated by the two ranking functions. Then, a list of objects $o^{(i)} = (o_1^{(i)}, o_2^{(i)}, ..., o_{n^{(i)}}{}^{(i)})$ is extracted from these two ranking lists, where $o_j^{(i)}$ is the j-th object in the ranking list, and $n^{(i)}$ is the number of objects in $o^{(i)}$. Furthermore, each list of spatial objects $o^{(i)}$ is associated with a list of scores $g^{(i)} = (g_1^{(i)}, g_2^{(i)}, ..., g_{n^{(i)}}{}^{(i)})$, where $g_j^{(i)}$ is the score of object $o_j^{(i)}$ with respect to a query $Que^{(i)}$. The score $g_j^{(i)}$ represents how relevant an object $o_j^{(i)}$ is to a query $Que^{(i)}$. Here, we manually select several queries from different categories as our training queries and then adopt the following equation to obtain $g_j^{(i)}$ [9].

$$
g_j^{(i)} =
\begin{cases}
\displaystyle\sum_{k=1}^{2} \frac{n^{(k)}}{\text{the position of } o_j^{(i)} \text{ in } l_k}, & \text{if } o_j^{(i)} \in l_1 \cap l_2 \\[3ex]
\displaystyle\frac{n^{(1)}}{\text{the position of } o_j^{(i)} \text{ in } l_1}, & \text{if } o_j^{(i)} \in l_1 \wedge o_j^{(i)} \notin l_2 \\[3ex]
\displaystyle\frac{n^{(2)}}{\text{the position of } o_j^{(i)} \text{ in } l_2}, & \text{if } o_j^{(i)} \in l_2 \wedge o_j^{(i)} \notin l_1
\end{cases}
$$

A feature mapping function $\Psi(\cdot)$ produces a feature vector $x_j^{(i)} = \Psi(Que^{(i)}, o_j^{(i)})$ for each pair of query $Que^{(i)}$ and spatial object $o_j^{(i)}$ [9]. Each list of feature vectors $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, ..., x_{n^{(i)}}{}^{(i)})$ and its associated list of scores

$g^{(i)} = (g_1^{(i)}, g_2^{(i)}, ..., g_{n^{(i)}}{}^{(i)})$ form a training instance $(x^{(i)}, g^{(i)}), i = 1, 2, ..., m$. The training set is given as $\left\{ (x^{(i)}, g^{(i)}) \right\}_{i=1}^{m}$.

The goal of training is to learn a ranking function f that generates a score $f(x_j^{(i)})$ for each feature vector $x_j^{(i)}$. These can be organized into a list of scores $z^{(i)} = (f(x_1^{(i)}), f(x_2^{(i)}), \ldots, f(x_{n^{(i)}}{}^{(i)}))$ for the list of feature vectors $x^{(i)}$. Here, in order to minimize the total loss on the training data, a loss function L based on the two lists of scores $g^{(i)}$ and $z^{(i)}$ is denoted as $\sum_{i=1}^{m} L(g^{(i)}, z^{(i)})$. Specifically, we adopt the method in [9] to define L:

$$L(g^{(i)}, z^{(i)}) = -\sum_{\forall o \in \mathcal{G}_k} P_{g^{(i)}}(o) log(P_{z^{(i)}}(o)), \text{ where } P_{z^{(i)}}(\mathcal{G}_k)) = \Pi_{t=1}^{k} \frac{exp(f(x_{o_t}^{(i)}))}{\sum_{l=t}^{n^{(i)}} f(x_{o_l}^{(i)})}$$

and $\mathcal{G}_k$ is the top-k subgroup containing all the permutations.

In ranking, given a new query $Que^{i'}$ and its corresponding list of spatial objects $o^{i'}$, a list of feature vectors $x^{(i')}$ is produced by using the feature mapping function Ψ. Using this list of feature vectors and the trained ranking function, the corresponding list of scores $g^{(i')}$ is generated. Last, top-k spatial objects $o^{i'}$ are ranked in a list l_{12} in ascending order of the scores.

We adopt the ListNet method [9] in the learn-to-rank step. In ListNet, a cross entropy metric and gradient descent are used as the loss function and learning algorithm, respectively. The gradient of $L(g^{(i)}, z^{(i)})$ with respect to parameter Que of f can be computed as follows: $\Delta Que = \frac{\partial L(g^{(i)}, z^{(i)}(f))}{\partial Que} = -\sum_{j=1}^{n^{(i)}} P_{g^{(i)}}(x_j^{(i)}) \frac{\partial f(x_j^{(i)})}{\partial Que} + \frac{1}{\sum_{j=1}^{n^{(i)}} exp(f(x_j^{(i)}))} \sum_{j=1}^{n^{(i)}} exp(f(x_j^{(i)})) \frac{\partial f(x_j^{(i)})}{\partial Que}$. For simplicity, we also use a linear neural network model to calculate the score of a feature vector $f_{Que}(x_j^{(i)}) = \langle Que, x_j^{(i)} \rangle$, where $\langle \cdot, \cdot \rangle$ denotes an inner product.

Similarity Based Question Reduction

When deciding the binary questions to ask, the similarity between the input ranking lists $l_1 = (a_1, \ldots, a_k)$ and $l_2 = (b_1, \ldots, b_k)$ can give us indications about what not to ask. For example, if we know $a_i = b_j$, it is unnecessary to ask "Is a_i better than b_j?" or the opposite. Let $N_c = |l_1 \cap l_2|$. The original $2k \times 2k$ matrix $\mathcal{M}$ can be reduced to a $(2k - N_c) \times (2k - N_c)$ matrix $\mathcal{M}'$. In particular, matrix $\mathcal{M}'$ is obtained by removing any column or row corresponding to a b_j that appears in l_1. By reducing matrix $\mathcal{M}$ to $\mathcal{M}'$, the total number of possible binary questions is also reduced. Now N_{bq} becomes $\frac{(2k - N_c)(2k - N_c - 1)}{2}$.

If $N_c = |l_1 \cap l_2| \geq 2$, we can further reduce the number of binary questions that we should ask. This is done by using the Kendall Tau Coefficient. Refer to the original $l_1 = (a_1, \ldots, a_k)$ and $l_2 = (b_1, \ldots, b_k)$. If $a_i = b_j$ and $a_{i'} = b_{j'}$ for $1 \leq i < i' \leq k$ and $1 \leq j < j' \leq k$, a_i and $a_{i'}$ are said to be concordant in the

two raking lists. Consequently, it is unnecessary to ask "Is a_i better than $a_{i'}$?" as the two ranking lists agree on this. Accordingly, for each concordant pair a_i and $a_{i'}$ we can reduce N_{bq} by 1. Assuming that there are N_τ concordant pairs, N_{bq} becomes $\frac{(2k-N_c)(2k-N_c-1)}{2} - N_\tau$.

Entropy Based Spatial Object Selection

For the remaining cells in matrix $\mathcal{M}'$, we want to select and ask questions that involve the most important spatial objects. For that purpose, we consider the amount of information a spatial object conveys in its keywords.

Intuitively, if a spatial object has a keyword that is more representative, this spatial object is more likely to be searched for and is thus of higher importance. Also, the more frequent a keyword occurs in spatial objects, the more representative this keyword is. With these considerations, we use entropy to compare spatial objects. The entropy takes into account the frequency of keywords in a certain spatial object as well as the relative proportion of their total distribution across objects. A spatial object will have a higher entropy if more representative keywords are associated with it.

Consider a spatial object set O and an object o. Let f be a ranking function, and let $\kappa(f,o)$ be a subset of $o.KW$ that is relevant to spatial objects obtained by f. We use $\kappa(f,O) = \bigcup_{o \in O} \kappa(f,o)$ to denote the union of such sets over all objects in O. We intend to determine the fraction of all keywords in $\kappa(f,O)$ that come from object o's contribution. This is captured by $P_{\kappa(f,O)}(o) = \frac{|\kappa(f,o)|}{|\kappa(f,O)|}$.

Here, we give an example to classify this point. Given a query q with a keyword q.keywords = (Fast food) and a ranking function f, $O = \{o_1, o_2, o_3, o_4\}$ is the result of top-4 query according to f, where o_1.keywords = (Food, Restaurant, Burger king), o_2.keywords = (Food, Burger king), o_3.keywords = (Food, Restaurant) and o_4.keywords = (Restaurant). Thus, $P_{\kappa(f,O)}(o_1) = \frac{3+3+2}{4} = 2$, $P_{\kappa(f,O)}(o_2) = \frac{3+2}{4} = \frac{5}{4}$, $P_{\kappa(f,O)}(o_3) = \frac{3+3}{4} = \frac{3}{2}$, and $P_{\kappa(f,O)}(o_4) = \frac{3}{4}$. Then, we can find that o_1 is more relevant to q and o_4 is less relevant to q. Further, we can filter some spatial objects in the process of generating binary questions.

Accordingly, a spatial object o's entropy is defined as

$$Entropy(o) = - \sum_{kw \in o.KW} P_{\kappa(f,O)}(o) log P_{\kappa(f,O)}(o)$$

The entropy of objects will be considered when we decide which object pairs should be included in binary questions that we publish to a crowdsourcing platform.

Generating Binary Questions

So far, we obtained a matrix $\mathcal{M}'$ with $N_{bq} = \frac{(2k - N_c)(2k - N_c - 1)}{2} - N_\tau$ cells that correspond to possible binary questions. Now we describe how to generate $M_q < N_{bq}$ binary questions from these cells.

The questions are generated by Algorithm C.1. The algorithm first processes each pair of objects in ranking list l_{12} (lines 2–3) that results from the learn-to-rank step (Section 3.2), as list l_{12} is regarded as having the k most important objects from lists l_1 and l_2. Specifically, the algorithm checks if the entropy difference between a pair of objects is smaller than a threshold σ (line 4). If so, a question is generated for the pair, as their closeness calls for human intelligence to judge whether the ranking between them is appropriate (lines 5–9). If the question budget has been exhausted, the algorithm returns (line 11).

Otherwise, the algorithm continues similarly on the common part of lists l_1 and l_2 (lines 14–26). If there is room for additional questions, the algorithm continues to work on all remaining objects in the union of l_1 and l_2 (lines 27–39). The "if-then-else" statements in the algorithm are used because our setting of binary questions is biased rather than symmetric. Having both directions of comparisons in the binary questions is expected to offset the bias to some extent.

4 Crowdsourcing Model

4.1 Answers from Crowd Workers

Each worker needs to answer M_q questions of the form "Is object o better than object o' for a query Que?" When we get $|W|$ binary (yes or no) answers from $|W|$ workers for each such question, we need to synthesize an overall answer to the question. We use three different methods to make this decision. The amount paid to a single worker is the same for the three voting methods that we use.

Majority voting (MV): MV does not require workers to specify their confidence. It simply counts which answer, yes or no, is the most frequent among the $|W|$ answers, and it uses that as the overall answer.

Voting based on constant confidence (VC): VC requires each worker $w_i \in W$ to specify a constant confidence cc_i from 10% to 100% before starting to answer the questions. After all $|W|$ answers are obtained for the current question, the majority is decided using weighted counting. In particular, let W_y (W_n) be the set of workers that give yes (no). The count of yes votes is $\sum_{w_i \in W_y} cc_i$, and that of no votes is $\sum_{w_i \in W_n} cc_i$.

Voting based on dynamic confidence (VD): VD is more complex than VC in that it requires a worker $w_i \in W$ to specify an individual confidence

---

Algorithm C.1 GenerateQuestions(Ranking lists l_1, l_2, and l_{12}, matrix $\mathcal{M}'$, entropy threshold σ, number of questions M_q)

 1: Question list $lq \leftarrow \varnothing$
 2: **for** each spatial object $o \in l_{12}$ **do**
 3:      **for** each spatial object $o' \in l_{12}$ after o **do**
 4:          **if** $|Entropy(o) - Entropy(o')| < \sigma$ **then**
 5:              **if** o appears before o''s in matrix $\mathcal{M}'$ **then**
 6:                  Add question "Is o' better than o?" to lq
 7:              **else**
 8:                  Add question "Is o better than o'?" to lq
 9:              **end if**
10:          **end if**
11:          **if** $|lq| = M_q$ **then** return lq
12:      **end for**
13: **end for**
14: $l_c \leftarrow l_1 \cap l_2$
15: **for** each spatial object $o \in l_c$ **do**
16:      **for** each spatial object $o' \in l_c$ after o **do**
17:          **if** $|Entropy(o) - Entropy(o')| < \sigma$ **then**
18:              **if** o appears before o''s in matrix $\mathcal{M}'$ **then**
19:                  Add question "Is o' better than o?" to lq
20:              **else**
21:                  Add question "Is o better than o'?" to lq
22:              **end if**
23:          **end if**
24:          **if** $|lq| = M_q$ **then** return lq
25:      **end for**
26: **end for**
27: $l_r \leftarrow (l_1 \cup l_2) \setminus l_c$
28: **for** each spatial object $o \in l_r$ **do**
29:      **for** each spatial object $o' \in l_r$ after o **do**
30:          **if** $|Entropy(o) - Entropy(o')| < \sigma$ **then**
31:              **if** o appears before o''s in matrix $\mathcal{M}'$ **then**
32:                  Add question "Is o' better than o?" to lq
33:              **else**
34:                  Add question "Is o better than o'?" to lq
35:              **end if**
36:          **end if**
37:          **if** $|lq| = M_q$ **then** return lq
38:      **end for**
39: **end for**
40: return lq

$dc_{i,j}$ when answering the j-th question. The overall counting for a particular question is similar to that in VC, except that for the j-th question, the local confidence $dc_{i,j}$ is used.

4.2 Matrix Factorization

After we have decided the overall answers for all the M_q binary questions, matrix $\mathcal{M}'$ (see Section 3.2) still has missing values in its cells. Such cells correspond to questions that we have not sent to crowdsourcing. In other words, as an ideal situation, for a query, we can ask $2k \times 2k$ questions and generate a $2k \times 2k$ matrix. Based on the answers to all these questions, it would be easy to judge which ranking function is better. This option, however, is infeasible due to the high number of questions. As the matrix is incomplete, we use matrix factorization to obtain a complete matrix that contains answers to all the $2k \times 2k$ binary questions. In particular, we use an EM procedure [10] to determine the missing values by maximizing the log-likelihood of the values of all cells. In each iteration, we fill missing cells with the corresponding values in the current model estimate in the expectation step and obtain updated model parameters by performing non-negative matrix factorization on that intermediate matrix in the maximization step.

4.3 The Final List for Objects

By now, we have obtained answers to all binary questions corresponding to cells in matrix $\mathcal{M}'$. Based on these answers, we utilize Borda Count [11] to get the final ranking list for all objects. We use l_f to denote the final list.

Specifically, we use $o \succ o'$ to capture each overall answer that "Object o is better than object o' for query Que." This way, all the binary answers can be captured as a set of such ordered pairs. For each object $o \in O$, we count its occurrences on the left hand side in such ordered pairs. That count is object o's Borda Count. Finally, we rank all objects according to their Borda Counts in a non-ascending fashion.

For example, given a set of objects $O = \{A, B, C, D\}$ and a set of ordered pairs $\{A \succ B, A \succ C, A \succ D, C \succ B, C \succ D, B \succ D\}$, the Borda Counts for all objects are as follows: $g(A) = 3, g(B) = 1, g(C) = 2, g(D) = 0$. Therefore, the final list l_f is $\langle A, C, B, D \rangle$.

5 Global Evaluation for Two Ranking Functions

Sections 3 and 4 explain how to obtain a final ranking list l_f for two ranking lists l_1 and l_2 that result from ranking functions for a query Que. Then we compare l_1 and l_2 with respect to l_f. Specifically, we calculate the Kendall

Tau Coefficients τ_1 between l_1 and l_f and τ_2 between l_2 and l_f. The one with the larger coefficient is regarded as the better ranking list.

We use a set of queries $QUE = \{Que_1, Que_2, ..., Que_M\}$ for evaluating ranking functions f_1 and f_2. For each Que_i, we use the process described so far to judge which of f_1 and f_2 gives a better result. The ranking function that produces the better list is assigned a score. After all queries in QUE are processed, the ranking function with the higher total score is regarded as the better ranking function.

6 Empirical Studies

6.1 Experimental Setup

We employ a real dataset of POIs extracted from OpenStreetMap[2]. All the POIs are from the city of Aalborg, Denmark. Each POI has a place name, a location given by coordinates, and a set of keywords. Also, each POI belongs to one of 41 categories (e.g., pubs, banks, cinemas). Table C.3 lists the dataset properties.

Table C.3: Data Set Properties

Property	Description
Total # of categories	41
Total # of objects	1043
Total # of unique keywords	1028
Total # of keywords	3773
Average # of words per category	39.72
Average # of unique words per category	10.82
Average # of unique words per object	3.62

We generate queries based on the locations and keywords in the dataset. Given a set O of POIs and a positive integer k, we generated a query q with k spatial keywords as in the literature [6]. For the $q.loc$ part, we randomly pick a location from the whole region of O. For the $q.KW$ part, we first sort all the keywords that occur in objects in O in descending order of their frequencies and then randomly pick k keywords among all keywords in the percentile range 10 to 40. This way, we get 91 queries. Further, we manually annotate these queries using semantic information (e.g., pubs, banks, cinemas) and classify them into 41 categories. In addition, we produce two ranking functions by adjusting the parameter (i.e., $\alpha = 0.7(f_1)$, $\alpha = 0.3(f_2)$) of a popular ranking function [2], and we form top-15 ranking lists for each query using

---

[2]http://www.openstreetmap.org

them. The ranking function is as follows.

$$D_{ST}(q,o) = \alpha \frac{D_{\varepsilon}(q.loc, o.loc)}{maxD} + (1-\alpha)(1 - \frac{P(q.KW|o.KW)}{maxP})$$

Here, $\alpha \in [0,1]$ is used to balance between spatial proximity and text relevancy; the Euclidian distance between q and o, $D_{\varepsilon}(q.loc, o.loc)$, is normalized by *maxD*, which is the maximum distance possible between two POIs; and *maxP* is used to normalize the text relevancy score into the range from 0 to 1. The training queries are also chosen randomly because multi-cross validation, which is basically repeating the experiment with different sets of binary questions, is expensive in our setting.

We use CrowdFlower[3] as the crowdsourcing platform. It is chosen for several reasons. First, it allows high throughput with little effort. For example, in preliminary experiments, an entire task was completed in less than one hour. Second, it allows us to test workers with "test questions" before starting to collect real answers from the workers. This helps us identify and avoid irresponsible workers.

Next, we describe the configuration of CrowdFlower. We first provide the workers with the following items: a query that consists of a query location and a set of keywords; a sequence of questions each of which includes a link to Google Maps that allows the workers to view the query location and the spatial objects involved in the question in addition to labels indicating a worker's answer and confidence. An example is shown in the Figure C.4. Then a sequence of binary questions are asked: (a) A question takes the form "Is POI 1 better than POI 2?" Here, "better" means "more relevant to the given query." So we ask whether one POI should be ranked higher than the other in a query result. (b) When workers answer a question, they must indicate their confidence in the answer. We provide three confidence levels: very confident, confident, and low confidence.

The parameters and their values used in the experiments are shown in Table C.4.

6.2 Effectiveness Study

In order to evaluate the performance of our proposal, we execute the following step. We define a set of categories $C = \{C_1, C_2, ..., C_g\}$, where C_i is a set of queries $C_i = \{Que_1, Que_2, ..., Que_{M_i}\}$. Suppose that we have a maximum budget and run the entire process (Figure C.2) for each category C_i using M_i queries. The result for category C_i can be represented as an M_i-dimensional vector $\vec{v} = (v_1, v_2, ..., v_{M_i})$, where v_i is the score of ranking function f_1 as described in Section 6.1.

---

[3]http://www.crowdflower.com

Query: cheap hotel

1. Do you think ''Hotel Hvide Hus (Vesterbro 2, 9000 Aalborg, Denmark)'' is better than
''First Hotel Europa (Vesterbro 12C, 9000 Aalborg, Denmark)'' for the given query?
Click here to see the query location and the two points of interest on Google Maps
○ Yes
○ No
Enter your confidence in your answer.

Select one   ▾

Fig. C.4: An Example in CrowdFlower

Table C.4: Parameter Settings

Parameter	Value
Total budget	$50
Total # of binary questions	1183
Max # of workers per binary question	31
Max # of binary questions per query	21
Average # of workers per binary question	15
Average # of binary questions per query	13
Average cost per binary question	$0.003

However, a very high budget is economically infeasible. Assuming a par-
ticular budget b, we run the entire process (Figure C.2) for each category C_i
using M_i queries. This gives a result that can be represented as another M_i-
dimensional vector $\vec{v}_b$ for ranking function f_1. We use cosine similarity (CS)
to compare how similar $\vec{v}_b$ and $\vec{v}$ are since CS is widely used and also suits
our needs. The more similar they are, the better the result with budget b is,
as budget b achieves a performance comparable to what is achieved with the
maximum budget.

Effect of Training Queries. We study how the training queries affect
the performance of the proposed approach. We vary the number of training
queries from 5 to 20. The results, shown in Figure C.5a, show that the per-
formance improves as the number of training queries increases. Further, the
experimental results advocate aforementioned assumption on the settings.

Effect of budget. Figure C.5b plots the CS against the budget that is
varied in the range [10, 50]. In this experiment, we use voting based on
dynamic confidence (VD in Section 4.1). We set the number of workers per
binary question to 15, and the cost per binary question to $0.003. The figure
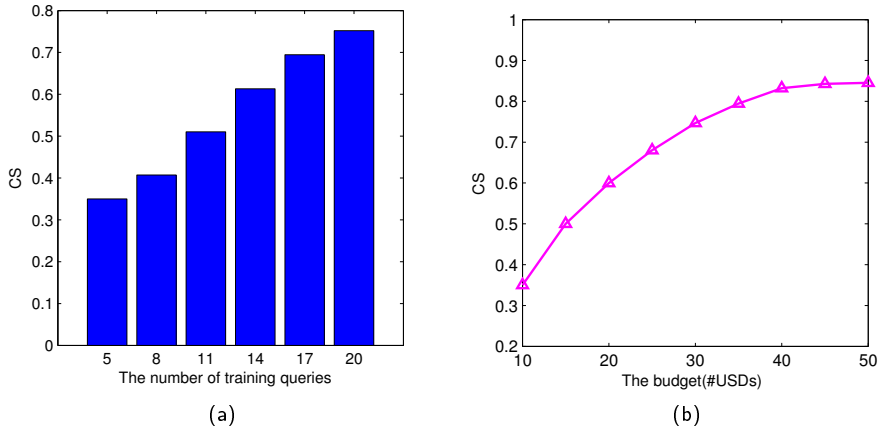
Fig. C.5: Effects of Training Query Number and Budget

shows that the proposed approach is affected by the budget. When we spend a larger budget, we obtain a better effectiveness. Furthermore, we can see that when the budget exceeds 40, the CS of the proposed approach is quite good and maintains a steady state. This indicates that our approach is inexpensive yet effective.
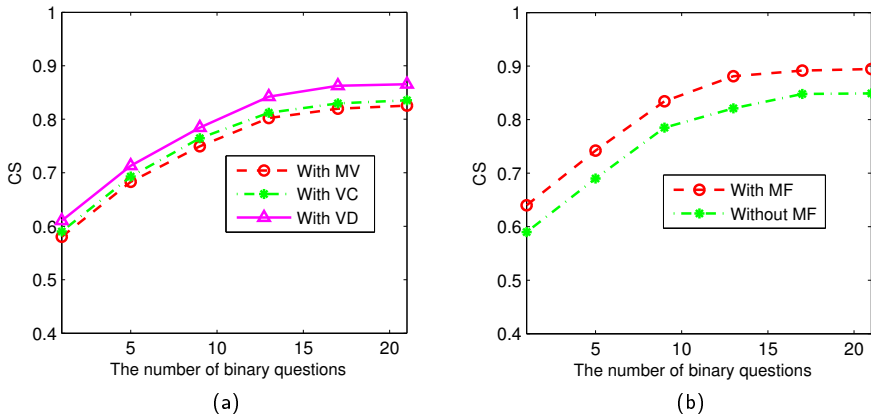
Fig. C.6: Effects of Voting Methods and Matrix Factorization

Effect of voting methods. Next, we compare the three voting methods (cf. Section 4.1). In this experiment, we set the number of workers per binary question to 15, the cost per binary question to \$0.003, and dc_{ij} to 0.5. Figure C.6a plots the CS against the number of binary questions that is varied in

the range [1, 21]. We can see that voting method VD outperforms the other two methods. Specifically, VD can increase the average CS by 3.63% and 2.55% compared to MV and VC, respectively. On the other hand, MV delivers the worst performance. The reason is that MV assumes that all workers have the same quality, and the answers from all workers are treated equally. This is not realistic, because workers are of different quality due to their different backgrounds and experiences. In contrast, VD models workers' quality dynamically for different questions and has the best evaluation performance. It is also noteworthy that VC's performance is in-between, since it is able to capture workers' different confidences, but does not allow workers to vary their confidence from question to question.

Effect of matrix factorization. We also evaluate the effect of matrix factorization (MF). To do so, we implemented two versions of our approach, with and without MF. The results are shown in Figure C.6b, where the x-axis represents the number of binary questions, and the y-axis represents the CS. We can see that the method with MF is clearly better than the one without MF. Through MF, we are able to determine answers for the binary questions that we do not crowdsource due to the budget constraint. This way, we recover more information that is used to evaluate the ranking functions.
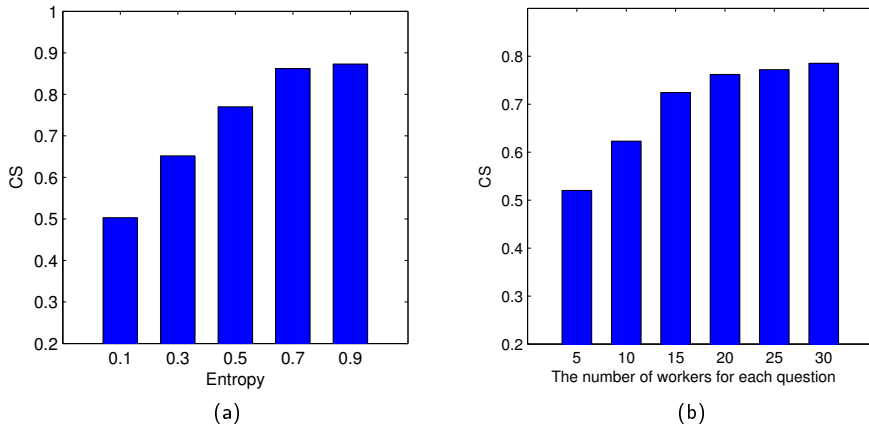
Fig. C.7: Effects of Entropy Threshold and Worker Number

Effect of entropy threshold. In this experiment, we study the effect of different entropy thresholds used for σ in Algorithm C.1. We set the number of workers per binary question to 15, the number of binary question per query to 13, and the cost per binary question to $0.003. The results of varying σ in the range [0.1, 0.9] are shown in Figure C.7a. Clearly, the entropy threshold has a significant impact on the CS. When the entropy threshold increases,

the CS of the proposed approach also increases. But after some point (0.7 in the experiment), CS stays stable. When the entropy threshold increases, object pairs are more likely to be included in crowdsourcing, which in turn tends to exploit more human intelligence in the evaluation. In the remaining experiments, we use the entropy threshold 0.7.

Effect of the number of workers. We also study whether the number of workers for each question affects the performance of the proposed approach. We vary the number from 5 to 30. Figure C.7b shows that the proposed approach is affected by the number of workers per question. As more workers are employed for a question, more human intelligence is exploited, which tends to help achieve a better evaluation. On the other hand, after some point, using more workers does not further improve the evaluation. In this experiment, for instance, when the number of workers per question exceeds 20, the CS of the proposed approach changes only little.
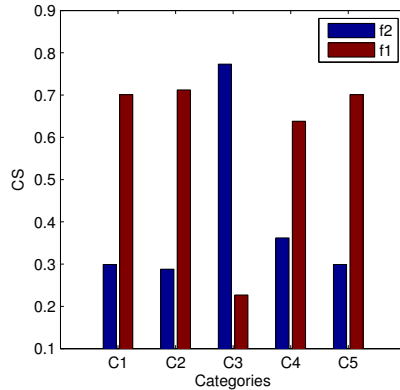
Fig. C.8: Effect of Category

Performance for different categories. Next, we investigate whether the query keywords used affect performance. We classify the query keywords into five categories: "Accommodation" (C1), "Education" (C2), "Tourist" (C3), "Food" (C4), and "Shop" (C5). The experimental results are shown in Figure C.8. Clearly, different categories result in different evaluation performance. Category C3 is of special interest as its evaluation result is opposite to those of the other four categories. The possible reason is that tourists always visit some places according to their interest, and therefore the textual similarity is more important than the spatial location. In this situation, f_2 works better because it gives more weight to the textual part in the ranking. For the other four categories, the query location is more important than the textual similarity, and f_1 works better as it considers the spatial aspect to be more important in the ranking.
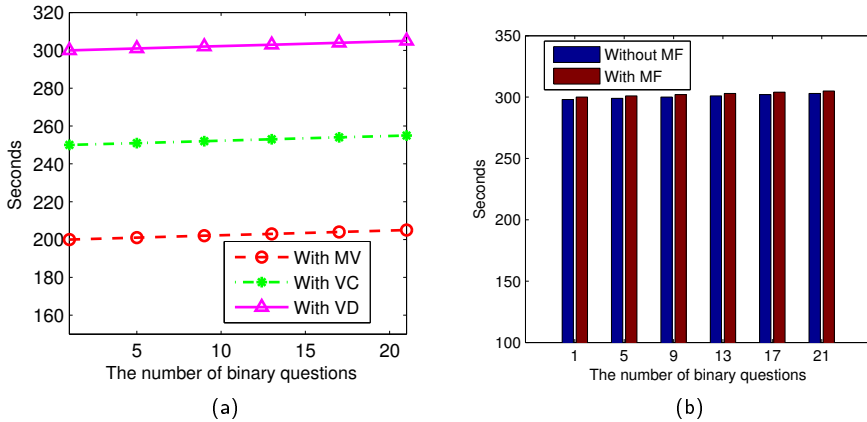
Fig. C.9: The Time Cost of Voting Methods and With/Without MF

6.3 Efficiency Studies

We also examine the time efficiency of the proposed approach. We set the number of workers per binary question to 20, the number of binary question per query to 13, and the cost per binary question to \$0.003.

First, we compare the average time cost of our proposal with the different majority voting methods (Section 4.1). As shown in Figure C.9a, the method with MV is fastest, VD is the slowest, and VC is in-between. The difference is attributed to the different processes of these methods. MV is the simplest without any worker-specified confidence, VD is the most complex with varying confidence from worker to worker and from question to question, and VC is again in the middle with different confidence for workers, but not for questions.

Second, we study the time cost of our proposal with and without MF. The results are reported in Figure C.9b. The approach with MF takes longer time (300 seconds at most) than the one without MF, since more answers of binary questions need to be determined after crowdsourcing. As the evaluation is not time-critical, the time difference is acceptable given that MF results in a clearly better evaluation.

7 Related Work

7.1 Spatial Keyword Search

Existing research on spatial keyword search focuses mainly on how to compute the top-*k* most relevant POIs when taking into account spatial location

and text relevancy.

Wu et al. [12] and Li et al. [13] propose a hybrid indexing framework to facilitate computing the top-k spatial web objects. Cao et al. [7] define the Location-aware top-k Prestige-based Text retrieval query that retrieves top-k spatial objects by considering both the location proximity of an object to the query and the prestige of the object. Junior et al. [14] design a new index named Spatial Inverted Index and related algorithms to perform top-k spatial keyword queries.

Cao et al. [6] define the problem of finding a group of objects that match the query keywords, minimizing intra-group distance and the distance among the objects in the group and the query location. Their method is restricted to Boolean keyword queries and Euclidean distance. Differently, Zhang et al. [4] study the collective keyword search that detects the spatially closest objects that match m user-specified keywords. Wu et al. [8] study efficient, joint processing of multiple top-k spatial keyword queries and design an index structure for efficiently finding the object closest to the query location among all objects that cover all the keywords specified in the query. In order to address scalability problems, Long et al. [15] use efficient exact and approximate algorithms to process two types of collective spatial keyword queries.

Our research studies spatial keyword search from a different angle. Specially, we develop a spatial keyword query evaluation system based on crowdsourcing. Our proposal can be used to evaluate the ranking functions used in the previous research on spatial keyword search.

A recent study [16] also employs crowdsourcing to evaluate ranking functions used in spatial keyword search. Given a set of spatial objects, it obtains partial rankings of them by synthesizing the answers of crowdsourcing workers to binary questions. In contrast, this paper uses a different overall framework (Figure C.2) that does not rely on partial rankings of spatial objects. Also, the question model and crowdsourcing model in this paper make use of learning and information processing techniques. We do not experimentally compare with the method in [16] because it takes a set of POIs as input whereas our method takes two ranked lists as input.

7.2 Crowd-based Query Processing

Crowdsourcing is widely used in the process of answering queries that are hard for machines but relatively easy for human intelligence.

Many studies on crowdsourcing focus on quality control. Joglekar et al. [17] propose an evaluation of worker quality by designing techniques that generate confidence intervals for worker error rate estimates. Supposing the rate between false positive and false negative for each worker is known, Parameswaran et al. [18] investigate the number of assignments that are re-

quired for a task. In order to accurately estimate a worker's quality, Feng et al. [19] propose a worker model to compute the worker's quality and a question model to compute a question's result using the worker model. Zheng et al. [20] aim to improve the result quality of generated questions by designing online task assignment approaches. They incorporate evaluation metrics into assignment strategies and consider quality measures. In order to increase the speed of query processing and reduce the operation cost, Wang et al. [21] propose a method based on data cleaning to reduce the effect of dirty data on aggregate query answers.

Other studies employ crowdsourcing to implement crowd-based operators, e.g., join [22], sort [23], max [24], graph search [25], and path selection [26]. In general, our work also falls into this category, but we focus on a specific topic, namely how to evaluate different ranking functions used in spatial keyword search.

8 Conclusion and Future Work

In this paper, we propose a crowdsourcing-based approach to evaluating ranking functions used in spatial keyword querying. Our approach consists of a series of steps. We use a matrix to model all possible binary questions concerning the different results returned by two ranking functions f_1 and f_2. As such a matrix is typically too large for a given budget, we use multiple steps to reduce the number of binary questions to be sent to crowdsourcing. Furthermore, we design a crowdsourcing model that obtains the answers to the binary questions sent to crowdsourcing. Finally, we devise a global evaluation process that compares f_1 and f_2 based on a multitude of sets of answers from crowdsourcing. We conduct extensive empirical studies on our proposals with real data. The results show that the proposal is efficient and able to deliver effective evaluations of ranking functions in spatial keyword querying.

Several directions exist for future work. First, it is possible to extend the binary question model to ask more complex questions that may also be helpful for the evaluation. Second, other economic models may be used in determining the number of questions for crowdsourcing for a given budget. Note that the proposed framework can accommodate such alternative economic models without substantial changes. Third, it is of interest to extend the proposal so that more than two ranking functions can be evaluated simultaneously. Fourth, it is also of interest to adapt the approach to tasks different from evaluating ranking functions in spatial keyword querying.

Acknowledgements

References

[1] D. Zhang, C.-Y. Chan, and K.-L. Tan, "Processing spatial keyword query as a top-k aggregation query," in *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR '14)*. ACM, 2014, pp. 355–364.

[2] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 337–348, Aug. 2009.

[3] I. D. Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *Proceedings of the 24th International Conference on Data Engineering (ICDE 2008)*, 2008, pp. 656–665.

[4] D. Zhang, Y. M. Chee, A. Mondal, A. K. H. Tung, and M. Kitsuregawa, "Keyword search in spatial databases: Towards searching by document," in *Proceedings of the 25th International Conference on Data Engineering (ICDE 2009)*, 2009, pp. 688–699.

[5] D. Zhang, B. C. Ooi, and A. K. H. Tung, "Locating mapped resources in web 2.0," in *Proceedings of the 26th International Conference on Data Engineering (ICDE 2010)*, 2010, pp. 521–532.

[6] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi, "Collective spatial keyword querying," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD '11)*. ACM, 2011, pp. 373–384.

[7] X. Cao, G. Cong, and C. S. Jensen, "Retrieving top-k prestige-based relevant spatial web objects," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 373–384, Sep. 2010.

[8] D. Wu, M. L. Yiu, G. Cong, and C. S. Jensen, "Joint top-k spatial keyword query processing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 10, pp. 1889–1903, 2012.

[9] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: From pairwise approach to listwise approach," in *Proceedings of the 24th International Conference on Machine Learning (ICML '07)*. ACM, 2007, pp. 129–136.

[10] S. Zhang, W. Wang, J. Ford, and F. Makedon, "Learning from incomplete ratings using non-negative matrix factorization," in *Proceedings of the 2006 SIAM International Conference on Data Mining*, pp. 549–553.

[11] J. A. Aslam and M. Montague, "Models for metasearch," in *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '01)*. ACM, 2001, pp. 276–284.

[12] D. Wu, G. Cong, and C. S. Jensen, "A framework for efficient spatial web object retrieval," *The VLDB Journal*, vol. 21, no. 6, pp. 797–822, 2012.

[13] Z. Li, K. C. K. Lee, B. Zheng, W. C. Lee, D. Lee, and X. Wang, "IR-tree: An efficient index for geographic document search," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 4, pp. 585–599, April 2011.

[14] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg, "Efficient processing of top-k spatial keyword queries," in *Proceedings of the International Symposium on Spatial and Temporal Databases (SSTD 2011)*. Springer, 2011, pp. 205–222.

[15] C. Long, R. C.-W. Wong, K. Wang, and A. W.-C. Fu, "Collective spatial keyword queries: A distance owner-driven approach," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13)*. ACM, 2013, pp. 689–700.

[16] I. Keles, S. Saltenis, and C. S. Jensen, "Synthesis of partial rankings of points of interest using crowdsourcing," in *Proceedings of the 9th Workshop on Geographic Information Retrieval (GIR '15)*. ACM, 2015, pp. 15:1–15:10.

[17] M. Joglekar, H. Garcia-Molina, and A. Parameswaran, "Evaluating the crowd with confidence," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '13)*. ACM, 2013, pp. 686–694.

[18] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom, "Crowdscreen: Algorithms for filtering data with humans," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD '12)*. ACM, 2012, pp. 361–372.

[19] J. Feng, G. Li, H. Wang, and J. Feng, "Incremental quality inference in crowdsourcing," in *Proceedings of the International Conference on Database*

Systems for Advanced Applications (DASFAA 2014). Springer, 2014, pp. 453–467.

[20] Y. Zheng, J. Wang, G. Li, R. Cheng, and J. Feng, "Qasca: A quality-aware task assignment system for crowdsourcing applications," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15).* ACM, 2015, pp. 1031–1046.

[21] J. Wang, S. Krishnan, M. J. Franklin, K. Goldberg, T. Kraska, and T. Milo, "A sample-and-clean framework for fast and accurate query processing on dirty data," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD '14).* ACM, 2014, pp. 469–480.

[22] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng, "Leveraging transitive relations for crowdsourced joins," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13).* ACM, 2013, pp. 229–240.

[23] X. Chen, P. N. Bennett, K. Collins-Thompson, and E. Horvitz, "Pairwise ranking aggregation in a crowdsourced setting," in *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining (WSDM '13).* ACM, 2013, pp. 193–202.

[24] P. Venetis, H. Garcia-Molina, K. Huang, and N. Polyzotis, "Max algorithms in crowdsourcing environments," in *Proceedings of the 21st International Conference on World Wide Web (WWW '12).* ACM, 2012, pp. 989–998.

[25] A. Parameswaran, A. D. Sarma, H. Garcia-Molina, N. Polyzotis, and J. Widom, "Human-assisted graph search: It's okay to ask questions," *Proc. VLDB Endow.*, vol. 4, no. 5, pp. 267–278, 2011.

[26] C. J. Zhang, Y. Tong, and L. Chen, "Where to: Crowd-aided path selection," *Proc. VLDB Endow.*, vol. 7, no. 14, pp. 2005–2016, Oct. 2014.

References

Paper D

Extracting Rankings for Spatial Keyword Queries
from GPS Data

Ilkcan Keles, Christian S. Jensen, Simonas Šaltenis

Abstract

Studies suggest that many search engine queries have local intent. We consider the evaluation of ranking functions important for such queries. The key challenge is to be able to determine the "best" ranking for a query, as this enables evaluation of the results of ranking functions. We propose a model that synthesizes a ranking of points of interest (PoI) for a given query using historical trips extracted from GPS data. To extract trips, we propose a novel PoI assignment method that makes use of distances and temporal information. We also propose a PageRank-based smoothing method to be able to answer queries for regions that are not covered well by trips. We report experimental results on a large GPS dataset that show that the proposed model is capable of capturing the visits of users to PoIs and of synthesizing rankings.

1   Introduction

A very large number of searches are performed by search engines like Google or Bing each day. One source [1] reports that Google processes more than 7 billion queries per day. A recent study [2] of users' local search behavior indicates that 4 in 5 users aim to find geographically related information. It also shows that 50% of the users who conducted mobile search and 34% of the users who used a computer or tablet visit a point of interest (PoI) on the same day. These statistics indicate the importance of location-based web querying.

To support queries with local intent, the research community has proposed many different spatial keyword functionalities to find relevant nearby PoIs [3]. A prototypical spatial keyword query takes a set of keywords and a location as arguments and returns a list of PoIs ranked with respect to a range of signals. Example signals include PoI ratings, properties of the neighborhoods of the PoIs, the distances of the PoIs to the query location, the textual relevances of the PoIs to the query keywords, and the relative expensiveness of the PoIs. These signals can be combined in multiple ways to obtain a ranking function. Most studies focus on indexing and efficient retrieval and thus evaluate the computational efficiency of proposed techniques. In contrast, the evaluation of the quality of the ranking functions is not covered well. We think that evaluation of the ranking functions is crucial since it is an important step towards increasing user satisfaction with location-based services; however, it is difficult to assess the quality of a ranking function when there is no yardstick ranking to compare against. The goal of this study is to propose a framework for constructing such baseline rankings that reflect the preferences of the users. Future studies will then be able to use the constructed rankings to evaluate the quality of different ranking functions.

A few studies ([4–7]) consider the use of crowdsourcing to synthesize rankings for objects and they can be used for spatial keyword queries. However, crowdsourcing-based approaches are expensive since workers need to be paid for each crowdsourcing task. They are also time consuming since there is a need to wait for the workers to complete the tasks. Further, it may be difficult to recruit workers who know about the spatial region of the query. Therefore, as a supplement to crowdsourcing, we focus on the use of GPS data to synthesize rankings.

We propose a method to build rankings for spatial keyword queries based on historical trips extracted from GPS data. We define a trip as a pair of consecutive stops extracted from a GPS trajectory. The stops represent the source and the destination of a trip, and we are interested in trips where the destination is a PoI. While the GPS data does not include spatial keyword queries, we can reasonably assume that a recorded trip to a PoI corresponds to issuing

a spatial keyword query at the starting location of the trip with a keyword that is part of the textual description associated with the PoI. For instance, if a user visited a restaurant r starting from a location l, we assume that the user issued a spatial keyword query at l with the keyword "restaurant" and that r is the preferred restaurant. Further, a PoI is considered to be relevant to the users in a region if many trips starting in that region visit the PoI. To the best of our knowledge, this is the first study of using GPS data to synthesize rankings for spatial keyword queries.

To synthesize rankings, we first extract stops of users from available GPS data. Then, we assign the stops to the PoIs that were visited. Furletti et al. [8] propose a PoI assignment method based on the distance between a stop and a PoI. We extend their method by taking into account temporal patterns of the users' visits to PoIs. Next, we extract all trips to PoIs.

Using the trips, we build a grid structure, where each cell records two values for each PoI, namely the number of trips from the cell to the PoI and the number of distinct users involved. To address the issue that some cells may have few or no trips, we adopt a personalized PageRank [9] based algorithm to smooth the values. The intuition behind using PageRank is that nearby grid cells should have similar values just like web pages linking to each other should have similar values. The resulting grid structure is used to form a ranking for a given spatial keyword query. First, the grid cell that contains the query location is found. Then the PoIs are filtered with respect to the query keywords. Finally, the PoIs are ranked according to the number of trips and the number of distinct users. The resulting ranking reflects the preferences of the users for PoIs, and a ranking function that produces a ranking more similar to the synthesized ranking is more preferable. Although a given collection of GPS data is limited in its geographical coverage and its coverage of users, we are still able to produce rankings in the particular settings where the GPS data offers good coverage.

To summarize, the main contributions are: (i) A method for synthesizing rankings of PoIs from GPS data that is able to produce results for regions without GPS data and that employs the number of trips and distinct users to rank PoIs, (ii) A stop assignment algorithm that employs users' temporal patterns when assigning stops to PoIs, (iii) PageRank-based algorithm to smooth the values for grid cells, (iv) An evaluation using a dataset of some 0.4 billion GPS records obtained from 354 users over a period of nine months.

The remainder of the paper is organized as follows. Section 2 covers preliminaries, related work, and the problem definition. The proposed model is covered in Section 3. Section 4 covers the evaluation, and Section 5 concludes and offers research directions.

2 Preliminaries

2.1 Data Model

The proposed method uses GPS records collected at one hertz from GPS devices installed in vehicles.

A GPS record G is a four-tuple $\langle u, t, loc, im \rangle$, where u is the ID of a user, t is a timestamp, loc is a pair of Euclidean coordinates representing the location, and im is the vehicle ignition mode. Even though im is not part of a GPS measurement, it is included in our dataset as a useful automotive censor measurement. An example GPS record is $\langle 5, 2014\text{-}03\text{-}01\ 13\text{:}44\text{:}54, (554025, 6324317), \text{OFF} \rangle$, where the coordinates of the location are given in the UTM coordinate system. Next, a trajectory TR of a user is the sequence of GPS records from this user ordered by timestamp t, $TR = G_1 \to ... \to G_i \to ... \to G_n$. We denote the set of all trajectories by S_{TR}.

We are interested in the locations where a user stopped for a longer time than a predefined threshold. We extract all such stops from S_{TR}. Specifically, a stop S is a three-tuple $\langle G, a_t, d_t \rangle$, where G is a GPS record, a_t is the arrival time at $G.loc$, and d_t is the departure time from $G.loc$. When we say the location of a stop, we refer to the location of G. Next, a point of interest (PoI) P is a three-tuple $\langle id, loc, d \rangle$, where id is an identifier, loc is a location, and d is a document that contains the textual description of the PoI.

We assume that a significant portion of users' stops are visits to PoIs, so when a user makes a stop, it is probable that the user did so to visit a PoI. We define an assignment A as a pair $\langle S, P \rangle$ of a stop S and a PoI P, indicating that a user stopped at the location of S to visit P. We are unable to assign all stops to PoIs, so only some stops have a corresponding PoI.

Having extracted all the stops of a user, we obtain the user's location history. In particular, the location history H of a user is defined as the sequence $H = S_1 \to ... \to S_i \to ... \to S_m$ of the user's stops ordered by a_t. A user's location history captures the user's trips. Specifically, a trip T is a pair $\langle S_i, S_j \rangle$ of a source and a destination stop. Given a trip $T = \langle S_i, S_j \rangle$ and an assignment $A = \langle S_j, P \rangle$, we say that T is a trip to PoI P.

Our goal is to use the trips extracted from GPS records to synthesize ranking of PoIs for spatial keyword queries.

Definition D.1 (Top-k Spatial Keyword Query). *Let S_P be a set of PoIs. A top-k spatial keyword query $q = \langle l, \phi, k \rangle$ on S_P is a three-tuple, where l is a query location, ϕ is a set of query keywords, and k indicates the number of results. The query q returns k PoIs from S_P that rank the highest according to a given ranking function. A frequently used ranking function is a weighted combination of the proximity of the PoI location to $q.l$ and the textual relevance of the PoI to $q.\phi$ [3].*

Problem Statement. We assume a set S_G of GPS records obtained from

vehicles and a set S_P of PoIs. Given a top-k spatial keyword query, we solve the problem of constructing a top-k ranking of PoIs included in S_P using S_G.

2.2 Related Work

Some studies propose crowdsourcing to obtain rankings of items ([4–7]). Yi et al. [4] propose a method based on pairwise comparisons and matrix completion. Chen et al. [5] also use pairwise comparisons and propose an active learning method that takes worker reliability into account to synthesize rankings. Stoyanovich et al. [6] use list-wise comparisons and build preference graphs for workers and combine these to obtain a global ranking. Keles et al. [7] propose a method based on pairwise comparisons in order to rank PoIs for a given query without assuming a total ranking on the PoIs. Crowdsourcing-based approaches are hard to apply in large-scale evaluations since they are expensive and time-consuming. In the context of spatial keyword queries, it is a challenge to recruit workers familiar with the relevant region and PoIs.

Some studies use GPS data to identify stops, visited PoIs, and interesting places ([8, 10–20]). An important place is one where users stop for a while. In these studies, a stop is generally defined either as a single GPS record corresponding to the loss of satellite signal when a user enters a building or a set of GPS records where a user remains in a small geographical region for a time period.

Alvares et al. [10] enrich GPS trajectories with moves and stops. They require a predefined set of possible stop places that is then used for annotating trajectories. Palma et al. [11] enable the detection of stops when no candidate stops are available. They use a variation of DBSCAN [21] that considers trajectories and speed information. The main idea is that if the speed at a place is lower than the usual speed, the place is important. We use GPS data collected from vehicles, and we have a specific signal telling whether the engine is on or off. This simplifies the detection of stops.

Many clustering-based methods have been proposed to identify significant locations from GPS data. Ashbrook et al. [12] use a variation of k-means clustering to identify locations. Kang et al. [13] propose a time-clustering method. Zhou et al. [14] propose a density based clustering algorithm to discover personally meaningful locations. Zheng et al. [15] propose a hierarchical clustering method to mine interesting locations. They employ a HITS [22] based inference model on top of the location histories of the users to define the interestingness of the locations by considering users as hubs and locations as authorities. Cao et al. [16] employ a clustering method to identify semantic locations. They enhance the clustering using semantic information provided by yellow pages. They propose a ranking model that utilizes both location-location relations and user-location relations as found in trajectories.

They also consider the stay durations and the distances traveled. Montoliu et al. [17] propose time-based and grid-based clustering to obtain places of interest. We are not using clustering because we are not interested in regions; instead, we want to identify the specific PoIs that are visited by users.

Some studies use different strategies to extract significant places from GPS data. Bhattacharya et al. [19] propose a method based on bearing change, speed, and acceleration for walking GPS data. In a recent study [20], they make use of density estimation and line intersection methods to extract places. Their work requires walking GPS data and polygon information for each PoI. Their method is not applicable in our setting.

Finally, methods have been proposed that identify visits to PoIs from GPS data. Given a stop, the goal is to identify a PoI. Spinsanti et al. [18] annotate stops with a list of PoIs based on the distance between the stop and the PoIs and the average durations people spend at the PoIs. Their method requires average stay durations for each PoI, which are provided by experts. This information is not available in our setting. Furletti et al. [8] propose a method that also forms a set of possibly visited PoIs by taking walking distance and opening hours into account. We extend their stop assignment strategy. Shaw et al. [23] consider the use of learning-to-rank methods to provide a list of possible PoIs when a user checks in. They use historical check-ins to form a spatial model of the PoIs. They also make use of PoI popularity information and user-specific information like a user's check-in history and information about a user's friends that have already checked in at the PoI. Kumar et al. [24] and Gu et al. [25] model the geographic choice of a user taking into account the distance between the stop and a PoI as well as the number of possible PoIs and their popularity when multiple PoIs are possible. They use labeled data (direction queries and check-ins) to train their model. Since we have no personal information or check-in data, we are unable to use their method when assigning stops to PoIs.

3 Proposed Method

3.1 Overview

The method consists of two phases: model-building and ranking-building.

The model-building phase takes a set of GPS records and a set of PoIs as the input and outputs a regular grid that partitions the underlying geographical space. Each grid cell records two values for each PoI: the number of trips from the cell to the PoI and the number of distinct users making trips.

Using the GPS records, we first extract stops. Then we determine the home and work locations of the users and assign non-home/work stops to PoIs. In the next step, we extract the set of all trips to the PoIs and we com-

pute the number of trips and distinct users for each cell and PoI. Finally, we smooth the values of the grid cells using an algorithm based on personalized PageRank [9].

The ranking-building phase uses the grid structure constructed to synthesize rankings for top-*k* spatial keyword queries. Given a query, we first locate the cell that contains the query location. Then the PoIs that have values in this cell are filtered according to the query keywords. The remaining PoIs are sorted according to the scores produced by a scoring function that is a weighted combination of the number of trips and the number of distinct users of a PoI in the cell of a query. The first *k* PoIs constitute the output.

3.2   Stop Extraction

To extract stops, we use the ignition mode attribute. Similar to Cao et al. [16], we employ a duration threshold parameter Δ_{th} to check whether an OFF record represents a stop. If the duration between consecutive OFF and ON records exceeds Δ_{th}, a stop is formed from the first GPS record. Arrival-time attribute a_t and departure-time attribute d_t of the stop correspond to the timestamp attributes of the OFF and ON records, respectively.

Since GPS readings might be inaccurate or missing, we augment the procedure with a distance threshold d_{th}. Only if the distance between the location of the ON record and the location of the OFF record is below d_{th} and the time difference exceeds Δ_{th}, the arrival record is classified as a stop.

To exclude stops at traffic lights but include short stops, e.g., to pick up kids at a kindergarden, Δ_{th} should be set to a value between 5–30 minutes. Parameter d_{th} can be set to a value in the range 100–500 meters.

If the GPS dataset does not contain an ignition mode attribute, the stops can be extracted by the methods mentioned in Section 2.2. In other words, all the subsequent steps of the proposed method are applicable to GPS trajectories in general.

3.3   Determining Home/Work Locations

Home and work locations are not of interest to our study, so a first step is to eliminate stops that relate to such locations.

To determine home/work locations, we employ an algorithm based on DBSCAN [21], which is a density-based clustering algorithm with two parameters, *eps* and *minPts*. If a point *p* has more than *minPts* points in its *eps*-neighborhood, that is in the circular region centered at *p* with a radius of *eps*, it is a core point. The points in the *eps*-neighborhood of a core point *p* are said to be directly reachable from *p*. A point *q* is reachable from *p* if there is a sequence of points $\langle p_1, ..., p_n \rangle$ with $p_1 = p$ and $p_n = q$, where each p_{i+1}

is directly reachable from p_i. The objects reachable from a core point forms a cluster.
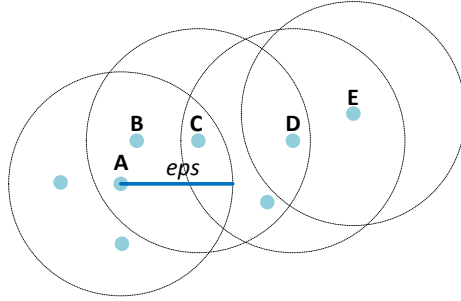
Fig. D.1: An Example DBSCAN Cluster

Figure D.1 shows an example cluster with *minPts* set to 4. Here, *A* is a core point since there are 5 points within its *eps*-neighborhood. Points *B* and *C* are directly reachable from *A*, and *E* is reachable from *A* since there is a sequence $\langle A, C, D, E \rangle$, where all of the preceding points of *E* are core points and each point in the sequence is directly reachable from the preceding point. All points reachable from *A* form the cluster.

The parameters are set to different values for each user since the total number of stops differs among users. For a given user, we set *minPts* to a value that is proportional to the number of distinct days this user has stops, and we introduce a parameter p_{hw} as the constant of proportionality. For instance, a p_{hw} value of 4/7 means that the user should have at least four stops a week to consider clustering them into a home/work cluster.

To determine the *eps* parameter, we first compute the distances between the locations of each stop belonging to the user and its nth nearest neighbor stop with $n = minPts$. Then we sort these distance values and eliminate those that exceed the globally defined distance threshold parameter (dn_{th}). Finally, for each distance value v_i, we compute the percentage of increase of the next distance value v_{i+1}: $(v_{i+1} - v_i)/v_i$. The distance value with the maximum percentage of increase becomes the *eps* parameter.

Having found the DBSCAN parameters of a user, we cluster his/her stops with respect to the location and compute the average stay duration for each cluster. If the duration exceeds a threshold Δ_{hw}, we conclude that the cluster represents a home/work location, and we mark the stops in the cluster as home/work stops. The intuition behind using a duration threshold to determine home/work locations is that people typically spend a long duration at home and work.

3.4 Stop Assignment to PoIs

The next step is to assign the remaining stops to PoIs. The goal is to assign as many stops to PoIs as possible while being conservative, thus getting assignments that are true with high certainty. To achieve this, we propose two methods: *distance based assignment* and *temporal pattern enhanced assignment*.

Distance Based Assignment (DBA). Furletti et al. propose a stop annotation method [8] that uses a maximum walking distance parameter and creates a list of PoIs that are within the maximum walking distance from the location of the stop and that have opening hours matching the time of the stop. Similarly, our DBA method searches for candidate PoIs in a circular region centered at the location of a stop with radius ad_{th}, a distance threshold that captures the maximum walking distance from the location of a stop to a PoI. In addition, the DBA method employs a parameter *lim* that sets an upper limit on the number of PoIs in the considered region. This avoids assigning a stop to a PoI when there are too many nearby PoIs. In such situations, it is not clear which nearby PoI was visited. Our goal is to make those assignments that we can make with relatively high certainty, so that the preferences of the users are captured while trying to avoid errors.

To assign a stop S to a PoI, we find the set of PoIs within the region defined by the location of S and parameter ad_{th}. Then we check whether the opening hours of the PoIs, if available, match with the arrival and departure time attributes of S. If the cardinality of the result set is below *lim*, we assign S to the closest PoI. Otherwise, we do not assign S to any PoI.

Temporal Pattern Enhanced Assignment (TPEA). The output of DBA might contain unassigned stops. These occur when there are either too many or no PoIs around the stops. We utilize temporal visit patterns to assign the unassigned stops.

For each user, we cluster non-home/work stops with respect to their locations using DBSCAN with *minPts* equal to the *lim* and *eps* equal to ad_{th} from DBA. If a cluster contains stops that are assigned to PoIs, we construct a so-called *visit-pattern matrix* for the cluster.

In this 2D matrix, the first dimension represents different days, and the second represents different times of a day. The value in a cell is the number of PoIs that the user visited during the corresponding time period. We use three levels of groupings of weekdays: top, weekdays/weekends, day. At the top level, we do not use the day information and the matrix has only one row and groups PoI visits by periods of a day. At the weekdays/weekends level, the matrix contains one row for weekdays and one for weekends. At the day level, we build seven rows, one for each day of the week. An example matrix for the weekdays/weekends level is shown in Table D.1.

Next, for each unassigned stop in the cluster, we check the number of PoIs for the corresponding cell starting from the top level of day grouping. If

Table D.1: Example Visit-Pattern Matrix

	00:00 06:00	06:00 12:00	12:00 18:00	18:00 00:00
Weekdays	0	0	2	0
Weekends	0	0	1	0

(a)

	00:00 06:00	06:00 12:00	12:00 18:00	18:00 00:00
	0	0	3	0

(b)
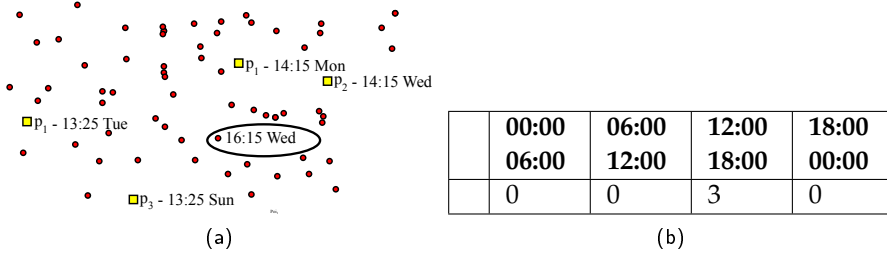
Fig. D.2: Cluster of Stops and Visit-Pattern Matrix - Top Level

there is only one PoI, we assign the stop to this PoI. If there is more than one PoI, we proceed to the weekdays/weekends level and, finally, the day level. Otherwise, we conclude that we cannot assign the stop.

Example. Figure D.2a represents one of the clusters of a single user after the assignment with DBA. Yellow rectangles and red circles represent the assigned and unassigned stops, respectively. The corresponding PoIs are denoted by p_1, p_2, and p_3. We want to assign the stop in the ellipse, and we break a day into four 6-hour periods.

We first check the matrix at the top level. We only have visits to PoIs in the time period between 12:00 and 18:00, and 3 distinct PoIs are visited. The top level matrix is shown in Table D.2b. Since the relevant cell value (3) exceeds 1, we consider the weekdays/weekends level. We have 3 stops on weekdays with 2 different corresponding PoIs, and we have 1 stop on weekends. The matrix is shown in Table D.1. The value of the relevant cell is 2, so we move to the day level, which is shown in Table D.2.

Table D.2: Visit-Pattern Matrix - Day Level

	00:00 06:00	06:00 12:00	12:00 18:00	18:00 00:00
...				
Wednesday	0	0	1	0
...				

Now the value of the relevant cell (Wednesday, 12:00-18:00) is 1, so the user visited only p_2 on a Wednesday during the time period containing 16:15.

Therefore, we assign the stop to p_2.

3.5 Computing Values of Grid Cells

We use Danske Kvadratnet[1], which is the official geographical grid of Denmark, as the underlying grid structure. The grid consists of equal-sized square cells of size 1 km^2 and it contains 111,000 cells.

Initializing Values of Grid Cells. Next, for each PoI p_i, we form the set T_{p_i} of trips to p_i. Using these sets, we initialize a grid structure for each PoI. For each cell, the number of trips from the cell to the PoI and the number of distinct users making these trips are computed and recorded.

Table D.3: Number of Cells out of 111,000 cells with Non-zero Values for Top-5 PoIs

PoI ID	Before Smoothing	After Smoothing
1	148	2,021
2	142	1,521
3	115	2,123
4	98	2,148
5	98	1,652

Smoothing the Values. After the initialization, many PoIs have sparse grids, where many cells have no trips to the PoIs. Table D.3 shows the number of cells with non-zero values for top-5 PoIs. Only 3 PoIs have more than 100 cells with non-zero values after initialization. Sparse grids are a problem since this reduces the number of spatial keyword queries that we can construct rankings for. If neighboring cells of an empty cell have non-zero values, it is reasonable to assume that trips starting in these cells are also relevant for the empty cell. So, the neighboring cell values can be used for smoothing to address the sparsity problem. The smoothing also helps reduce noise in cell values.

As the smoothed grids of multiple PoIs will be used in the ranking building phase, a smoothing method should have two properties. First, for a PoI, a smoothing algorithm should not change the sum of all the values in the grid for that PoI. Inflating or deflating the sum of values would unfairly promote or demote the PoI in relation to other PoIs in a constructed ranking. Second, the ordering of the values for all PoIs in a specific cell before and after smoothing should be similar in order to reduce distortion of the original spatial popularity data for the PoIs.

The literature contains some smoothing and interpolation methods for spatial grid based data. The inverse distance weighting (IDW) method [26] is proposed to interpolate missing values using distance-based weighting.

---

[1]http://www.dst.dk/da/TilSalg/produkter/geodata/kvadratnet

This method builds on the intuition that the effect of a cell's value on the value of an originally empty cell should depend on how close the cell is to the empty cell. However, IDW does not contain a smoothing method, and it changes the sum of values in the grid. Therefore, we do not utilize IDW in our work. Kernel-based methods [27] have also been proposed for smoothing. For these, it is possible to preserve the sum of the values since they produce a probability distribution as output. The sum of the values can then be distributed according to this distribution. However, they might introduce changes to the ordering of grid cells since kernel-based methods yield continuous functions that might not reflect the original properties of the data.

We use a smoothing algorithm based on personalized Pagerank [28] to interpolate values for cells with no trips. The PageRank algorithm was proposed for web graphs, where web pages are the vertices and hyperlinks are the edges. The algorithm assigns a page rank value to each website to indicate the relative importance of it within the set. A web page is considered important if other important web pages link to it. The algorithm can be described as a random walk over a directed graph $G = \langle V, E \rangle$. A random walker starts from a randomly chosen vertex. Then, with probability $1 - \alpha$, it follows an outgoing edge, and with probability α, it teleports to another randomly chosen vertex $y \in V$, where α has the same value for each web page and $0 < \alpha < 1$. The PageRank of a vertex is the probability that a random walker will end up at the vertex.

Personalized PageRank [9] was proposed in order to incorporate personalized preferences. This is achieved by changing the uniform probability distribution of teleportation to a random web page to a personalization parameter that is basically a distribution based on user preferences. We use this parameter to utilize the initial values of the grid cells while smoothing the values.

The PageRank algorithm is a good candidate for smoothing, since, if a cell is close to another cell, they should have similar values just like the page rank values for the web pages linking to each other. The main idea is that if a PoI is of interest to drivers leaving from a cell, it might also be of interest to drivers leaving from nearby cells.

We first convert the underlying grid into a directed graph. For each cell, we introduce a vertex. Then, we add edges from a "cell" to the neighboring "cells" with weight $w = 1/d^2$, where d denotes the distance between the centers of the cells. The edge weights define how the page rank value of a vertex should be distributed to the adjacent vertices. In the initial version of PageRank, it is equally distributed. In our case, we use weights based on distance to make sure that the page rank value is distributed inversely proportional with the distance between the grid cells corresponding to the vertices. Then we apply PageRank to this graph for both number of trips and number of

users values. We use the initial cell values obtained after the initialization to determine the personalization parameters. The probability of teleportation to a vertex is set proportional to the actual value of the corresponding grid cell.

The procedure yields a probability distribution that indicates the likelihood that a random walker will end up at a particular vertex. We distribute the total number of trips and the total number of distinct users to the cells proportional to the output probability distribution. For instance, assume that we are smoothing the numbers of trips and that the total number of trips to the PoI is 100. A cell with probability 0.23 then gets the value 23. Note that this smoothing procedure is done for each PoI. Table D.3 shows that smoothing provides a significant increase in the number of cells with non-zero values.

Example. Let G be a grid with cells $c_1, c_2, c_3, c_4, c_5, c_6$. The grid structure is shown in Figure D.3a. The first value represents the number of trips from each cell to a PoI before smoothing.

c_1 - 0, 0.126, 1.890	c_2 - 10, 0.277, 4.155	c_3 - 2, 0.141, 2.115
c_4 - 2, 0.135, 2.025	c_5 - 1, 0.201, 3.015	c_6 - 0, 0.120, 1.800

(a)

(b)
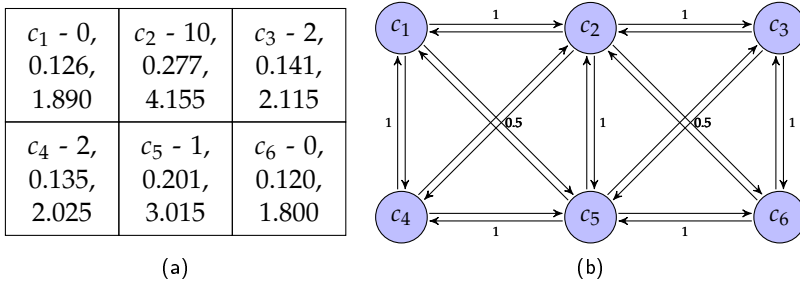
Fig. D.3: Grid Structure and Corresponding Graph

The graph representing the grid is given in Figure D.3b. Each vertex has an edge to each vertex that represents a neighboring cell. Each edge is assigned a weight as explained above. For instance, the distance between c_1 and c_2 is 1 unit, and the distance between c_1 and c_5 is $\sqrt{2}$ units, so the weight of edge (c_1, c_5) is $1/d^2 = 0.5$.

Then, we apply personalized PageRank using the initial values as the personalization input. The second value of each cell in Figure D.3a represents the resulting probability of the cell.

Finally, we distribute the sum of the values according to the pagerank values. The third value of each cell in Figure D.3a represents the smoothed value. Here, c_5 has the second largest value because it is closer to the cell with the largest value (c_2) than c_4 and c_6, and it has more edges than c_3 and c_1 since it is in the middle column. The effect of the number of edges is not an issue when smoothing is applied on a large grid since the grid cells, except the cells on the boundary of the grid structure, have the same number

of edges.

3.6 Extracting Rankings for Queries using the Model

To form a ranking for a given top-k spatial keyword query, we use the grid model.

Algorithm D.1 The Algorithm for Ranking-building Phase

Input: q - top-k spatial keyword query, *model* - the grid model
Output: r_k - a ranked list of PoIs
 1: $c \leftarrow$ the corresponding grid cell for $q.l$ in *model*
 2: $p \leftarrow$ the set of PoIs that have values in c
 3: $p_f \leftarrow$ the PoIs in p which are filtered using the query keywords $q.\phi$
 4: Rank the PoIs in p_f with respect to their values in c and assign it to r_{all}
 5: **if** r_{all} has more than k elements **then**
 6:     $r_k \leftarrow$ the first $q.k$ elements of r_{all}
 7:     **return** r_k
 8: **else**
 9:     **return** r_{all}
10: **end if**

The algorithm, given in Algorithm D.1, first finds the cell that contains the query location l. Then it filters the PoIs with values in the cell with respect to the query keywords to exclude the PoIs that do not contain query keywords. Here, we assume that all PoIs with descriptions that do not contain any of the query keywords will not be among the popular ones for this query. The ranking is computed using the ranking function given in Equation D.1, where n denotes the number of trips, d denotes the number of distinct users, and β is the weighting parameter.

$$s = \beta \times n + (1 - \beta) \times d \text{ where } 0 \leq \beta \leq 1 \tag{D.1}$$

If there are more than k relevant elements in the ranking then the top k elements form the output of the algorithm, as shown in Lines 5–7. Otherwise, the ranking is the output, as shown in Line 9. It is important to note that the output ranking might contain fewer than k elements. Although the algorithm does not produce a complete ground-truth ranking, the partial ranking is still useful for evaluation purposes as it provides valuable information about the expected result.

4 Experimental Evaluation

In Section 4.1, we report on studies aimed at understanding effects of parameter settings. In Section 4.2, we compare the stop assignment methods with a baseline method. Finally, we study the effect of smoothing and weighting parameters on the output rankings in Section 4.3. We do not present a complexity analysis since we think that it is not a real concern due to the fact that model-building is performed only once.

In the experiments, we use 0.4 billion GPS records collected from 354 cars traveling in Nordjylland, Denmark during March to December 2014. The PoI dataset used in the experiments contains around 10,000 PoIs of 88 categories. All of the PoIs are located in or around Aalborg.

4.1 Exploring the Parameters

To explore the effects of changing the parameters on the outputs of the different steps, we vary one parameter at a time while fixing other parameters to their default values. The parameters are described in Table D.4.

Stop Extraction

Here, we study the effect on stop extraction of varying Δ_{th} and d_{th}.

(a)   (b)
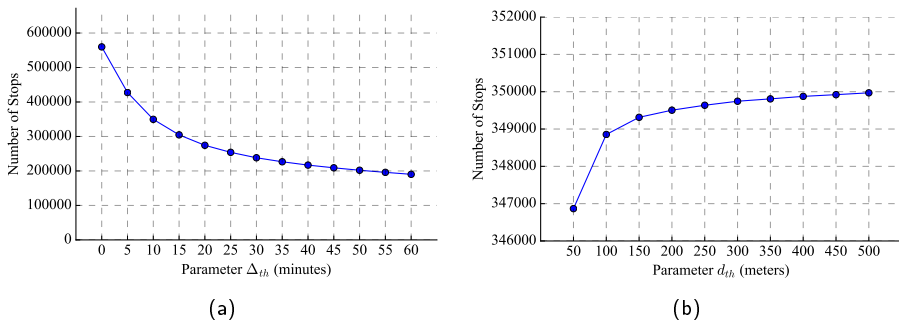
Fig. D.4: Effects of Parameters Δ_{th} and d_{th}

As shown in Figure D.4a, the number of stops decreases as Δ_{th} increases, as expected. The decrease is smooth. In order to capture meaningful stops from GPS data, parameter Δ_{th} should be set to a value in the range of 5–30 minutes since this setting would exclude quite short stops which might not be a visit to a PoI and as we see from the figure, there are a lot of quite short stops.

As can be seen in Figure D.4b, the number of stops increase when d_{th} increases, which is as expected. Although this parameter has some effect on the

Table D.4: Parameters

Notation	Step	Explanation	Default Value
Δ_{th}	Stop extraction	Minimum duration between two GPS records to consider the first one as a stop.	10 minutes
d_{th}		Maximum distance between two GPS records to consider that their locations match.	250 meters
Δ_{hw}	Determining home and work locations	Minimum average stay duration in home/work locations.	240 minutes
p_{hw}		Minimum fraction of days in a week a person is expected to visit home/work.	Three days a week (3/7)
ad_{th}	Stop assignment to PoIs	Maximum distance between a PoI and the location of a stop.	100 meters
lim		Maximum number of PoIs within the region bounded by the location of a stop and the ad_{th} parameter.	5 PoIs

number of stops extracted, the increase in the number of stops is negligible. This parameter is introduced to eliminate inaccurate GPS readings, and the results suggest that there are only few inaccurate readings in our dataset.

Determining Home/Work Locations

Here, we analyze the effect of p_{hw} and Δ_{hw} on determining home/work locations.

As shown in Figure D.5a, both the number of home/work locations and the number of stops assigned to home/work locations decrease when p_{hw} increases, because of the fact that more weekly stops are required to form a cluster. The decrease in the former is sharper than the decrease in the latter which is a consequence of that parameter p_{hw} is used to limit the number of stops in a cluster to be considered as a home/work location. The clusters that are left out when p_{hw} increases are the clusters with a small number of stops.
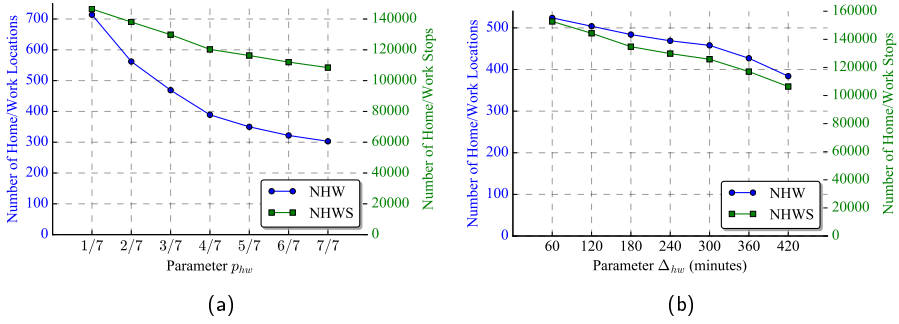
Fig. D.5: Effect of Parameters p_{hw} and Δ_{hw}

This is why the decrease in the number of stops assigned to home/work locations are not as sharp as the decrease in the number of home/work locations.

Figure D.5b shows that the numbers of home/work locations and stops decrease as Δ_{hw} increases, as expected. Unlike when increasing p_{hw}, the patterns of decrease are quite similar for both when increasing Δ_{hw}. This suggests that the average durations that users spend inside clusters are not correlated with the number of stops in the clusters.

Stop Assignment to PoIs

Here, we analyze the effect of parameters ad_{th} and *lim* on the assignment of stops to PoIs. In addition, we assess the effect of ad_{th} on the distance between stop location and the assigned PoI for TPEA. Figure D.6 shows the effect of varying ad_{th} on the number of stops that can be assigned to PoIs and the number of PoIs that receive assignments of stops when *lim* is set to 5 or 10.
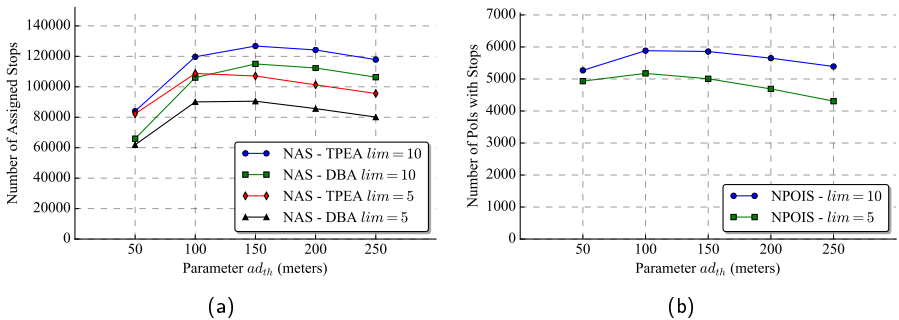
Fig. D.6: Effect of Parameter ad_{th}

Figure D.6a shows that for both *lim* values, the number of assigned stops

increases up to a point and then decreases when ad_{th} increases. When ad_{th} is small, it is impossible to assign some stops since there are no PoIs in the region defined by the location of the stop and the parameter. However, when ad_{th} increases, the number of PoIs within the bounded region increases as well. At some point, the number of PoIs starts to exceed the value of *lim*, and we are unable to assign the stop to a PoI. When this occurs, the number of stops starts to decrease.

Figure D.6b shows that the number of PoIs with stops assigned to them follows a very similar pattern. The decrease after an increase is also the result of having too many PoIs ($> lim$) inside the region bounded by parameter ad_{th}.
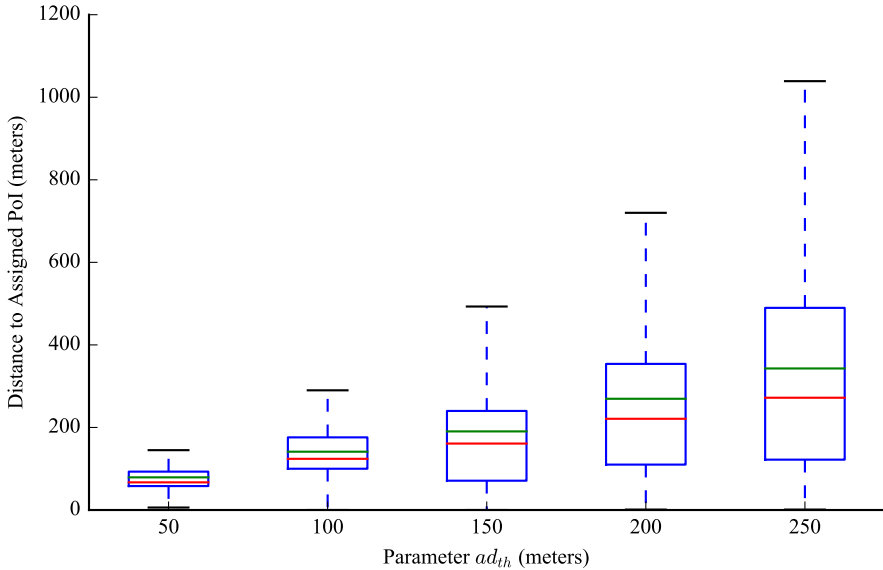
Fig. D.7: Effect of Parameter ad_{th} on Distance Distribution for TPEA

Figure D.7 shows the effect of varying ad_{th} on the distribution of the distance between the stop location and the assigned PoI for TPEA when *lim* is set to 5. The red lines show the medians, and the green lines show the means of the distance values. For this experiment, we only consider the stop locations assigned by TPEA. The figure shows that the distance between the stop and the assigned PoI increases when ad_{th} increases, as expected since parameter *eps* is set to ad_{th} parameter in the density based clustering. This figure also shows that although no specific distance threshold parameter is employed by TPEA, the distance between the stop and the assigned PoI does not exceed a few multiples of ad_{th} since it employs density-based clustering to form the visit pattern matrices. In other words, the assigned PoIs when TPEA is employed are still within reasonable distance from the stops.
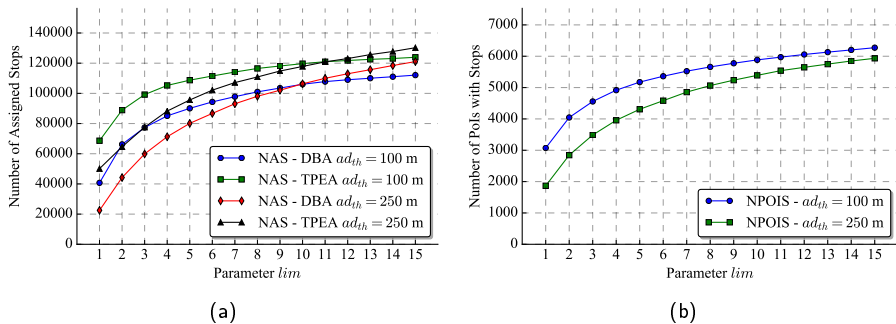
Fig. D.8: Effect of Parameter *lim*

Figure D.8 shows the effect of varying parameter *lim* on the number of assigned stops and the number of PoIs in assignments when ad_{th} is set to 100 or 250 meters. As expected, both the number of stops and the number of PoIs increase when *lim* increases because the algorithm is able to assign stops that it could not assign for smaller values of *lim*.

Figures D.6 and D.8 also show that TPEA is able to assign additional stops using the temporal patterns of the users found in the initial assignment with the DBA method. For instance, the number of stops assigned using TPEA is around 110,000 while it is 90,000 using DBA in Figure D.6a for $ad_{th} = 100$ and $lim = 5$.

4.2 Evaluation of Stop Assignment

To evaluate the accuracy of stop assignment methods, we use the home/work locations extracted from GPS data using the method explained in Section 3.3 since we do not have access to a proper ground-truth data. To extract home/work locations, we use the default values given in Table D.4 for parameters Δ_{hw} and p_{hw}. Then, the extracted home/work locations are inserted to the PoI database. The assignments of home/work stops to the newly inserted home/work PoIs forms the ground-truth dataset for this experiment. In other words, no stops are assigned to any regular PoI in this ground truth. However, we use the set of all PoIs (regular PoIs and home/work PoIs) in this experiment. We assign the complete set of stops using the proposed methods and compare our methods with the closest assignment method (CA) that assigns the stop to the closest PoI regardless of the number of PoIs around it.

In particular, we report the precision and recall [29] for the stop assignment methods. The true positives are the stops marked as home/work stops that are assigned to the correct home/work PoI, and the false positives are the non-home/work stops that are assigned to a home/work PoI. The false

negatives are the home/work stops that are assigned to a PoI different from the ground-truth PoI or are not assigned at all.

(a)　　　　　　　　　　　　　　　　(b)
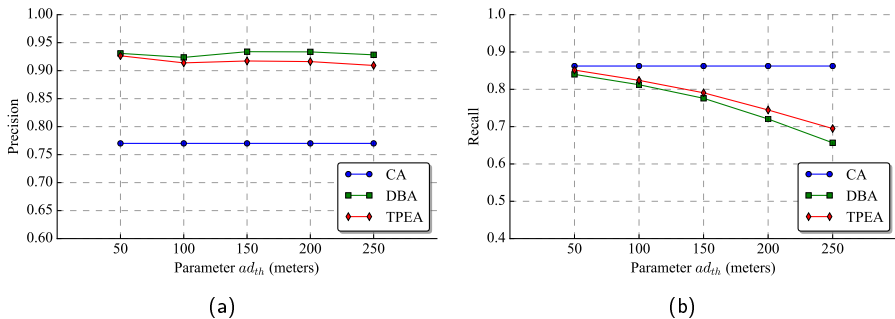
Fig. D.9: Precision and Recall

Figure D.9a shows that precision values of DBA and TPEA are higher than that of CA. The precision of DBA is slightly higher than that of TPEA since utilization of the temporal visit patterns of the user can introduce false positives. The precision, at 0.93, indicates that DBA and TPEA are able to assign home/work stops and the remaining stops almost correctly.

Figure D.9b shows that CA has better recall than DBA and TPEA that cannot assign all the home/work stops due to the constraints set by parameters ad_{th} and *lim*. Since the unassigned home/work stops are false negatives, DBA and TPEA have lower recall than CA. For $ad_{th} = 50$ and $ad_{th} = 100$, DBA and TPEA achieve a recall above 0.8. TPEA achieves a slightly higher recall because of an increase in the true positives and a decrease in the false negatives compared to DBA.

4.3 Exploring the Effect on Output Rankings

We proceed to study the effect of the grid smoothing method and the weighting parameter of the ranking function (β) on the output top-*k* rankings.

Smoothing, described in Section 3.5, changes the original values of the grid as well as introduces non-zero values in cells that lack data. This, in effect, extrapolates the available data to wider geographical areas, but it may also distort the original data. To observe the effect of the smoothing, we compute the top-10 PoIs for the grid cells that have initial values for at least 10 different PoIs, before and after smoothing, and we report the distribution of the Kendall tau distance [30] between them. The top-10 lists are formed according to the ranking function in Equation D.1.

To explore the effect of the weighting parameter (β) in Equation D.1, we present the average Kendall tau distance between the rankings constructed

for top-*k* spatial keyword queries using different β values. The set of queries used in this experiment consists of top-*k* queries with $k = 10$ and $k = 15$. The set of keywords used in top-*k* queries is {"restaurant", "supermarket","store"}, and the set of locations include the centers of grid cells that contain values for at least *k* PoIs.

Kendall Tau Distance. The distance is defined in Equation D.2 [30], where R_1 and R_2 denote the rankings that are compared and P is the set of pairs of the PoIs.

$$K(R_1, R_2) = \frac{\sum\limits_{(p,q)\in P} \bar{K}_{p,q}(R_1, R_2)}{|P|} \tag{D.2}$$

Function $\bar{K}_{p,q}$ is given in Equation D.3. If R_1 and R_2 agree on the ranking of PoIs p and q, the function evaluates to 0; otherwise, it evaluates to 1.

$$\bar{K}_{p,q}(R_1, R_2) = \begin{cases} 0 & \text{if } R_1 \text{ and } R_2 \text{ agree on } p, q \\ 1 & \text{if } R_1 \text{ and } R_2 \text{ do not agree on } p, q \end{cases} \tag{D.3}$$

Effect of Smoothing

We report the Kendall tau distance distributions between top-*k* rankings obtained before and after smoothing for different β values.
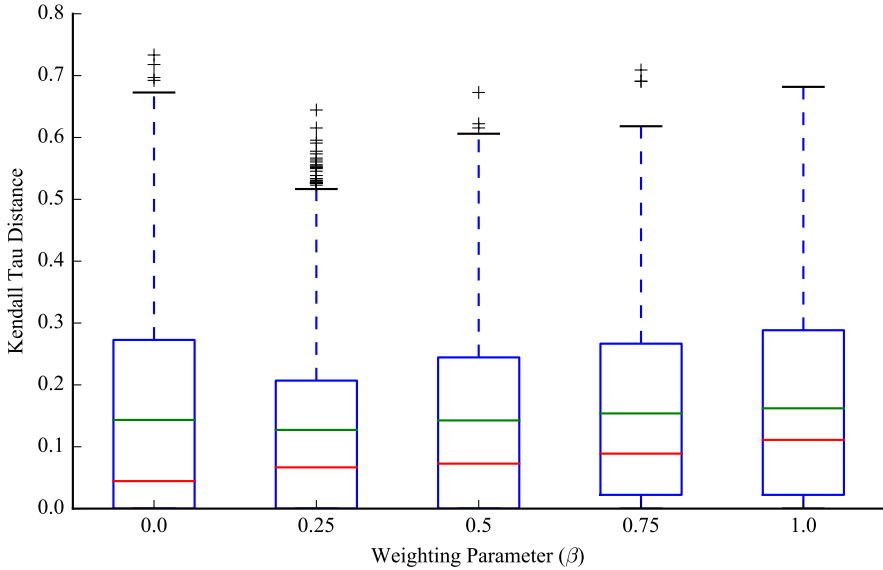
Fig. D.10: Kendall Tau Distance Distribution

The results are shown in Figure D.10, where the red lines show the medians and the green lines show the means of the distance values. The points denoted by a plus sign shows outlier distance values. On average, we achieve a Kendall tau distance around 0.15, which means that we can capture 85% of the relations between pairs after smoothing. We can also see that for all β values, the resulting distribution is right-skewed. We achieve a Kendall tau distance less than 0.1 for half of the grid cells and a Kendall tau distance around 0.3 for 75% of the grid cells. Further, for β values 0, 0.25, and 0.5, the smoothing does not introduce any changes in top-k PoIs for at least 25% of the grid cells.

Effect of Weighting Parameter

Figure D.11 reports the average Kendall tau distance between the top-k rankings produced using different β values for top-10 and top-15 queries. For instance, in Figure D.11a, the green bar on the group $\beta = 0$ indicates that the average Kendall tau distance between the rankings produced with $\beta = 0$ and $\beta = 0.25$ is around 0.12. The distances between rankings produced with different β values are less than 0.2. This suggests that the proposed model to extract output rankings is not overly sensitive to the weighting parameter.

(a) $k = 10$ (b) $k = 15$
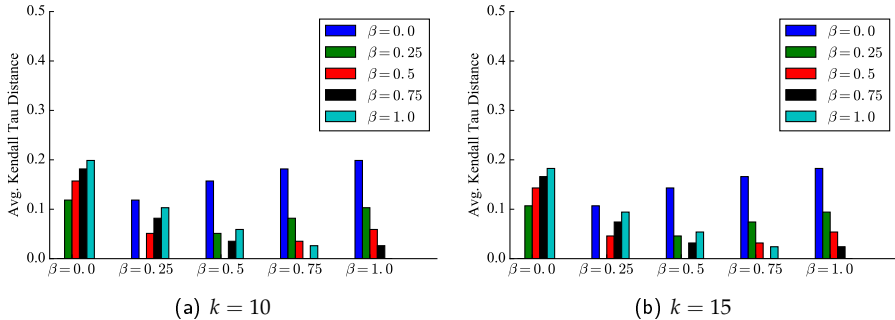
Fig. D.11: Avg. Kendall Tau Distance for Top-k queries

5   Conclusion and Future Work

The paper proposes a model with two phases, model-building and rank-building, to synthesize rankings for top-k spatial keyword queries based on historical trips extracted from GPS data. We propose a novel stop assignment method that makes use of the distances between the locations of the stops and the PoIs as well as temporal information of the stops to obtain the trips. We also propose a Pagerank-based smoothing method in order to extend the

geographical coverage of the model. Experiments show that the model is able to produce rankings with respect to the visits of the users, and that the output rankings produced by the model are relatively insensitive to variations in the parameters.

In future work, it is of interest to use the methods proposed here for evaluation of the ranking functions for spatial keyword queries, as this is the motivation behind this work. Another future direction is to explore probabilistic stop assignment in order to contend better with dense regions since the proposed methods use a conservative distance based approach when assigning stops to PoIs. In other words, we assign a stop to a PoI if it is highly probable that the visit occurred. As a result, it is not possible to assign stops in regions with many PoIs. It is also of interest to try to employ more advanced home/work identification methods to be able to determine home/work locations more accurately. It is also of interest to combine data sources like geo-coded tweets and check-ins with GPS data to form rankings for spatial keyword queries.

References

[1] (2015, Jun.) Google annual search statistics. [Online]. Available: http://www.statisticbrain.com/google-searches/

[2] Google. (2014, May) Understanding consumers' local search behavior. Accessed: 2018-03-06. [Online]. Available: https://goo.gl/7TrZNl

[3] X. Cao, L. Chen, G. Cong, C. S. Jensen, Q. Qu, A. Skovsgaard, D. Wu, and M. L. Yiu, "Spatial keyword querying," in *Proceedings of the 31st International Conference on Conceptual Modeling (ER 2012)*. Springer, 2012, pp. 16–29.

[4] J. Yi, R. Jin, S. Jain, and A. Jain, "Inferring users' preferences from crowdsourced pairwise comparisons: A matrix completion approach," in *Proceedings of the First AAAI Conference on Human Computation and Crowdsourcing (HCOMP 2013)*, 2013, pp. 207–215.

[5] X. Chen, P. N. Bennett, K. Collins-Thompson, and E. Horvitz, "Pairwise ranking aggregation in a crowdsourced setting," in *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining (WSDM '13)*. ACM, 2013, pp. 193–202.

[6] J. Stoyanovich, M. Jacob, and X. Gong, "Analyzing crowd rankings," in *Proceedings of the 18th International Workshop on Web and Databases (WebDB '15)*. ACM, 2010, pp. 41–47.

[7] I. Keles, S. Saltenis, and C. S. Jensen, "Synthesis of partial rankings of points of interest using crowdsourcing," in *Proceedings of the 9th Workshop on Geographic Information Retrieval (GIR '15)*. ACM, 2015, pp. 15:1–15:10.

[8] B. Furletti, P. Cintia, C. Renso, and L. Spinsanti, "Inferring human activities from GPS tracks," in *Proceedings of the 2nd ACM SIGKDD International Workshop on Urban Computing (UrbComp '13)*. ACM, 2013, pp. 5:1–5:8.

[9] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, TR 1999-66, 1999.

[10] L. O. Alvares, V. Bogorny, B. Kuijpers, J. A. F. de Macedo, B. Moelans, and A. Vaisman, "A model for enriching trajectories with semantic geographical information," in *Proceedings of the 15th Annual ACM International Symposium on Advances in Geographic Information Systems (GIS '07)*. ACM, 2007, pp. 22:1–22:8.

[11] A. T. Palma, V. Bogorny, B. Kuijpers, and L. O. Alvares, "A clustering-based approach for discovering interesting places in trajectories," in *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC '08)*. ACM, 2008, pp. 863–868.

[12] D. Ashbrook and T. Starner, "Using GPS to learn significant locations and predict movement across multiple users," *Personal Ubiquitous Comput.*, vol. 7, no. 5, pp. 275–286, Oct. 2003.

[13] J. H. Kang, W. Welbourne, B. Stewart, and G. Borriello, "Extracting places from traces of locations," in *Proceedings of the 2Nd ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots (WMASH '04)*. ACM, 2004, pp. 110–118.

[14] C. Zhou, D. Frankowski, P. Ludford, S. Shekhar, and L. Terveen, "Discovering personal gazetteers: An interactive clustering approach," in *Proceedings of the 12th Annual ACM International Workshop on Geographic Information Systems (GIS '04)*. ACM, 2004, pp. 266–273.

[15] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma, "Mining interesting locations and travel sequences from GPS trajectories," in *Proceedings of the 18th International Conference on World Wide Web (WWW '09)*. ACM, 2009, pp. 791–800.

[16] X. Cao, G. Cong, and C. S. Jensen, "Mining significant semantic locations from GPS data," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 1009–1020, Sep. 2010.

[17] R. Montoliu, J. Blom, and D. Gatica-Perez, "Discovering places of interest in everyday life from smartphone data," *Multimedia Tools and Applications*, vol. 62, no. 1, pp. 179–207, Jan 2013.

[18] L. Spinsanti, F. Celli, and C. Renso, "Where you stop is who you are: Understanding peoples' activities," in *Proceedings of the 5th BMI Workshop on Behaviour Monitoring and Interpretation (BMI'10)*, 2010, pp. 38–52.

[19] T. Bhattacharya, L. Kulik, and J. Bailey, "Extracting significant places from mobile user GPS trajectories: A bearing change based approach," in *Proceedings of the 20th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL '12)*. ACM, 2012, pp. 398–401.

[20] ——, "Automatically recognizing places of interest from unreliable GPS data using spatio-temporal density estimation and line intersections," *Pervasive Mob. Comput.*, vol. 19, no. C, pp. 86–107, May 2015.

[21] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*. AAAI Press, 1996, pp. 226–231.

[22] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *J. ACM*, vol. 46, no. 5, pp. 604–632, Sep. 1999.

[23] B. Shaw, J. Shea, S. Sinha, and A. Hogue, "Learning to rank for spatiotemporal search," in *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining (WSDM '13)*. ACM, 2013, pp. 717–726.

[24] R. Kumar, M. Mahdian, B. Pang, A. Tomkins, and S. Vassilvitskii, "Driven by food: Modeling geographic choice," in *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining (WSDM '15)*. ACM, 2015, pp. 213–222.

[25] Q. Gu, D. Sacharidis, M. Mathioudakis, and G. Wang, "Inferring venue visits from GPS trajectories," in *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'17)*. ACM, 2017, pp. 81:1–81:4.

[26] D. Shepard, "A two-dimensional interpolation function for irregularly-spaced data," in *Proceedings of the 1968 23rd ACM National Conference (ACM '68)*. ACM, 1968, pp. 517–524.

References

[27] T. Hastie, R. Tibshirani, and J. Friedman, *Kernel Smoothing Methods*. Springer, 2009.

[28] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," in *Proceedings of the Seventh International Conference on World Wide Web (WWW7)*. Elsevier, 1998, pp. 107–117.

[29] R. A. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[30] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee, "Comparing partial rankings," *SIAM Journal on Discrete Mathematics*, vol. 20, pp. 47–58, 2004.

References

Paper E

Extracting Visited Points of Interest from Vehicle Trajectories

Ilkcan Keles, Matthias Schubert, Peer Kröger, Christian S. Jensen, Simonas Šaltenis

Abstract

Identifying visited points of interest (PoIs) from vehicle trajectories remains an open problem that is difficult due to vehicles parking often at some distance from the visited PoI and due to some regions having a high PoI density. We propose a visited PoI extraction (VPE) method that identifies visited PoIs using a Bayesian network. The method considers stay duration, weekday, arrival time, and PoI category to compute the probability that a PoI is visited. We also provide a method to generate labeled data from unlabeled GPS trajectories. An experimental evaluation shows that VPE achieves a precision@3 value of 0.8, indicating that VPE is able to model the relationship between the temporal features of a stop and the category of the visited PoI.

1   Introduction

Mobile devices and vehicles are often able to track their movements using GPS and other localization methods. The resulting trajectories can be used in several fields such as traffic management, security, and location based services. As the amount of available trajectory data increases day by day, it becomes increasingly important to be able to assign semantic information to this data. Indeed, a number of studies consider the enrichment of trajectories with semantic information. SMoT [1] and CB-SMoT [2] annotate stops. SMoT requires a predefined set of candidate places to stop while CB-SMoT uses clustering and speed information to detect stops that are not predefined. Further, many proposals [3–8] use some form of clustering to extract interesting and significant locations. Bhattacharya et al. [9] propose a method based on bearing change, speed, and acceleration to identify interesting locations. Some of the methods proposed [6, 7] also support ranking of output locations. We focus on the identification of visits to PoIs because identified visits can provide valuable insights about the behavior of users. In particular, this information can help understand the popularity or the importance of PoIs, which in turn is useful when location-based services, such as spatial keyword search, are designed and evaluated [10].

There are a few studies on extracting visited PoIs and activities from GPS trajectories. Nishida et al. [11] propose a probabilistic PoI identification method for GPS and Wi-Fi trajectories. The method is semi-supervised and employs a hierarchical Bayesian model that makes use of personal preferences, stay locations, and stay times for each PoI category to assign a stop to a PoI. Bhattacharya et al. [12] propose a two-phase algorithm to assign stops to PoIs given a PoI database containing a polygon for each PoI and mobile phone trajectory data. The first phase includes generation of random points according to an error distribution for each GPS record and kernel density estimation on the latitude, longitude, and time dimensions. The second phase ranks the PoIs using a line segment intersection based approach. However, both of these studies require mobile trajectory data that includes walking GPS data and Wi-Fi data. In contrast, extracting visits to PoIs from vehicle trajectories is different since drivers often park their vehicle some distance from the location of the visited PoI and walk to the PoI, with walking not being part of the trajectory. Another complication is that while a mobile phone is personal, a car is often shared. Thus, it is difficult to personalize car trajectories. Furletti et al. [13] and Keles et al. [10] use a distance based approach to assign stops to PoIs. However, these methods are not accurate when stops are surrounded by many nearby PoIs. The assumption that the closest PoI is visited often does not apply in this scenario.

We propose a method to extract visited PoIs from vehicle GPS trajectories

that takes into account temporal aspects. The method employs a time-based Bayesian network with distance based filtering to determine the category of a visited PoI. In order to learn the Bayesian network, we use a labeled dataset constructed using distance based assignment based on cases where there is only a single possible PoI within the vicinity of a stop. The Bayesian network takes weekday, arrival time, and stay duration into account to determine the category of the visited PoI.

To summarize, we present a method for the extraction of visited PoIs from vehicle trajectories. The main contributions are: (i) An algorithm using a temporal feature-based Bayesian network in combination with distance-based filtering to assign a stop to a PoI, (ii) A distance-based method to construct labeled temporal feature vectors from vehicle trajectories, (iii) An evaluation using a dataset of about 0.4 billion GPS records obtained from 354 users over a period of nine months.

The remainder of the paper is organized as follows. Section 2 covers preliminaries and the problem definition. The proposed method is presented in Section 3. Section 4 covers the evaluation, and Section 5 concludes the paper.

2 Preliminaries

2.1 Data Model

The method in this paper relies on GPS data collected from cars and a point of interest (PoI) database to find visited PoIs for each user.

Definition E.1 (Point of Interest). *A point of interest (PoI) P is a five-tuple $\langle id, x, y, c, o \rangle$, where id is an identifier, x and y are coordinates, c is the set of categories of the PoI, and o is the opening hours of the PoI for each day of the week. A category can be one of the place types supported by the Google Places API [1] and there are no hierarchical relations among categories.*

An example PoI is $\langle$Louvre Museum, 451327, 5412229, museum, {09:00–18:00, 09:00–22:00, 09:00–18:00, 09:00–22:00, 09:00–18:00, 09:00–18:00}$\rangle$, where coordinates are given in the Universal Transverse Mercator (UTM) coordinate system, and the opening hours information is a set containing opening hours for each weekday.

Definition E.2 (GPS Record). *A GPS record G is a five-tuple $\langle u, t, x, y, im \rangle$, where u is a user ID, t is a timestamp, x and y are coordinates, and im is the ignition mode of the vehicle obtained from GPS device preinstalled on the vehicle.*

---

[1]`https://developers.google.com/places/supported_types`

The trajectory of a user is defined by sorting GPS records of the user with respect to the timestamp t.

Definition E.3 (Trajectory). *A trajectory TR of a user is the sequence of GPS records from this user ordered by timestamp t: $TR = G_1 \rightarrow ... \rightarrow G_i \rightarrow ... \rightarrow G_n$. We denote the set of all trajectories by S_{TR}.*

We are interested in the locations where a user stopped and spent longer time than a predefined threshold. We extract such stops for all users from S_{TR}.

Definition E.4 (Stop Location). *A stop location S is a three-tuple $\langle G, a_t, d_t \rangle$, where G is a GPS record that represents the stop, a_t is the arrival time at the stop location, and d_t is the time of departure from the stop location.*

We assume that a user stops at a location to visit a PoI.

Definition E.5 (Assignment). *An assignment A is a pair $\langle S, P \rangle$ of a stop location S and a PoI P, indicating that the user who stopped at location S visited PoI P. The set of all assignments S_A might not contain all stop locations due to various reasons.*

2.2 Problem Statement

Let S_{TR} denote a set of GPS trajectories and S_P denote a database of PoIs in the geographical region covered by S_{TR}.

We tackle the problem of identifying visits of users whose trajectories are given in S_{TR} to the PoIs contained in S_P. This problem can be divided into two sub-problems: Identifying the stops in trajectories, and assigning the stops to PoIs with the available information. We focus on the second problem.

3 Visited PoI Extraction

We propose Visited PoI Extraction (VPE), a method that employs a Bayesian network to extract visited PoIs. Section 3.1 gives the overview of VPE, and Sections 3.2 and 3.3 detail how we build the Bayesian network and how it is used for assignment, respectively.

3.1 Overview

VPE uses a Bayesian network in combination with distance-based filtering to determine the most likely PoI category of a stop location. The Bayesian network represents the relationship between a stop (based on the weekday, time, and duration of the stop) and the category of the PoI. To learn the

Bayesian network, VPE includes a method to construct a labeled dataset from GPS records because human labeled datasets like check-ins are usually not available for vehicle trajectories. The assignment phase assigns a given stop location to a PoI using the Bayesian network built in the previous phase. It computes the joint probability of the category and stop location and outputs the category with the maximum probability. Additionally, the set of possible categories is restricted to the categories of nearby PoIs.

3.2   Building the Bayesian Network
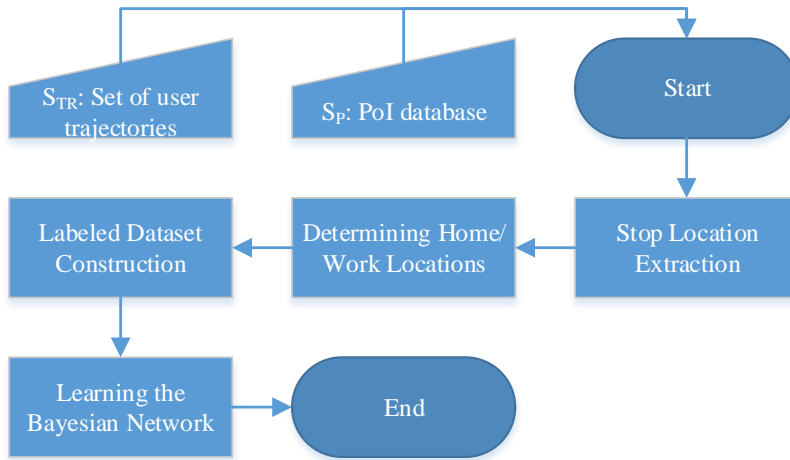
Fig. E.1: Flowchart of Building the Bayesian Network

The flowchart for building the Bayesian network is given in Figure E.1. It consists of four phases: stop location extraction, determining home/work locations, labeled dataset construction, and learning the Bayesian network. We need to determine home/work locations in order not to assign stops which correspond to home/work place to any PoIs. In other words, it excludes stops that are related to home/work locations, which is a necessary step to identify true visits to PoIs.

Stop Location Extraction and Determining Home/Work Locations

To extract stop locations and to determine home/work locations of users, we use methods employed in previous work [10]. For stop location extraction, we

148

use a duration threshold parameter Δ_{th} and a distance threshold parameter d_{th} to infer whether a GPS record with ignition mode IGNITION-OFF corresponds to a stop location. If the time difference between an IGNITION-OFF record and the next IGNITION-ON record exceeds Δ_{th} and the spatial distance between them is below d_{th}, the location of the IGNITION-OFF record is classified as a stop location.

To determine the home/work locations of a user, we employ a density based clustering approach. First, we cluster the stop locations of a user with DBSCAN [14]. The parameters of DBSCAN are determined with respect to a proportionality parameter (p_{hw}). For instance, a p_{hw} value of 5/7 means that the user should have at least 5 stop locations per week for a set of stop locations to form an output cluster. If the average stay duration of the GPS records in an output cluster exceeds a duration threshold (Δ_{hw}), we conclude that the stop locations forming the cluster are home/work stops.

Labeled Dataset Construction

To learn a Bayesian network, we need labeled stop locations to estimate distributions of stop locations over weekdays, arrival times, and stay durations for each PoI category. A labeled stop is an assignment that maps the stop location to the visited PoI. Unfortunately, labeled stops are typically not available for vehicle trajectories. Thus, VPE includes a method for extracting labeled stops directly from the trajectories. To generate labeled stops, we rely on distance based assignment (DBA) [10] with extreme parameter settings. The DBA method takes a stop location, a distance threshold parameter (ad_{th}), and a limit parameter (*lim*) as input. It assigns the stop location to the closest PoI if the number of PoIs in the circular region centered at the location of the stop location with a radius of parameter ad_{th} is below parameter *lim*. We use $lim = 1$ that makes it highly probable that the PoI a stop is assigned to is the actual visited PoI because DBA only assigns a stop location to a PoI if the PoI is the only PoI within the region surrounding the stop. Thus, the correctness of resulting labeled stops should be sufficiently large to derive distributions for the temporal parameters in the Bayesian network. Let us note that the labeled set is drawn from a different feature base, i.e., it is completely distance-based, and thus the derived temporal samples are labeled based on an independent feature space.

Learning the Bayesian Network

The Bayesian network contains four nodes. One node is the category of the PoI that the stop is assigned to. The three remaining nodes correspond to three attributes that can be inferred from a stop location: time index, day index, and stay duration, which is the difference between the a_t and d_t attributes of a stop location. The time and day index values are determined

from the arrival time (a_t) at the stop location according to the parameters of time period duration tp and day granularity dg. The day is divided into equal time slots of duration tp, and the time index is a value identifying the time slot. For instance, if tp is 2 hours and a_t is 08:23, the time index value is 4. Possible values of dg are 1, 2, and 3 corresponding to a daily level, a weekdays-weekend level, and no distinction, respectively. So, if dg is set to 1, we have 7 possible values for the day index, and if dg is set to 2, we have 2 possible values.
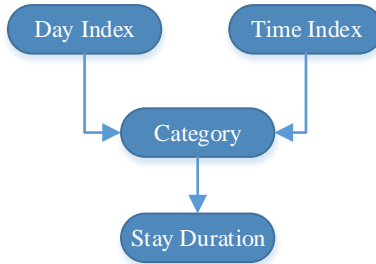
Fig. E.2: Structure of the Bayesian Network

The structure of the Bayesian network is shown in Figure E.2. Each node refers to an attribute inferred from a stop location and contains a conditional probability table. A directed edge from node A to node B shows that the value of attribute B is dependent on the value of attribute A. The structure is based on the preliminary analysis of the labeled dataset constructed by the method explained in Section 3.2 with parameter $ad_{th} = 100$ m.
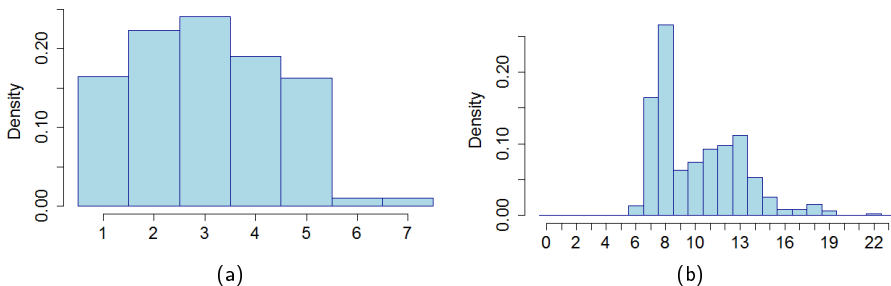
Fig. E.3: Day and Time Index Distributions for Universities

The day and time distributions for the university category when $tp = 1$ and $dg = 1$ are illustrated in Figure E.3. According to these distributions,

people tend to visit universities on weekdays (1–5), arriving during the time period 08:00–10:00. These distributions also show that the day of the week and the time have an affect on the probability of visiting a category, and this is why we have edges from the time index and day index nodes to the category node in the Bayesian network.
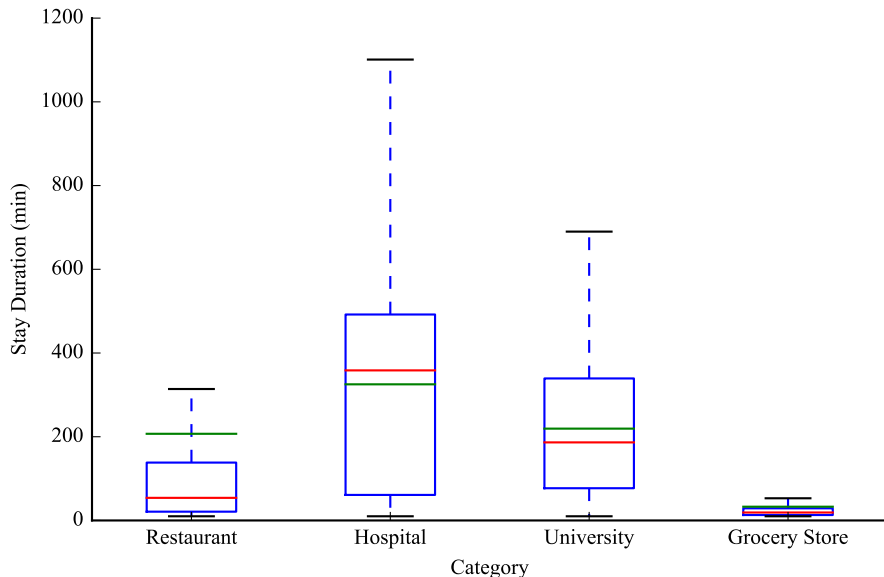
Fig. E.4: Stay Duration Distribution for Different Categories

The stay duration distribution for some categories is shown in Figure E.4, where the green and red lines represent mean and median values, respectively. This figure shows that the category has a significant impact on the stay duration. For this reason, we have an edge from the category node to the stay duration node. We model the stay duration values of each category as a log-normal distribution [11]:

$$sd \sim \ln \mathcal{N}(\mu, \sigma^2),$$

where μ and σ are the mean and standard deviation of the stay duration values for a category. For the stay duration node, we compute the observed mean and standard deviation for each category while learning the Bayesian network. The observed distribution is then used to compute conditional probability of a stay duration value given a PoI category.

After deciding the structure, we learn the Bayesian network with the labeled dataset. The learning here means forming conditional probability tables for the nodes of Bayesian network.

3.3   Assignment using Bayesian Network

We proceed to explain how the Bayesian network is used to assign a stop location to a PoI. The assignment method employs distance-based filtering, which reduces the number of possible categories. It does so by filtering the PoIs according to a distance factor parameter (df) that together with ad_{th}, introduced in Section 3.2, determines a distance threshold.

Algorithm E.1 The Assignment Algorithm

Input: bn - Bayesian network, sl - Stop location, ad_{th} - distance threshold for labeled dataset construction, df - distance factor, tp - time period duration, dg - day granularity

Output: p - visited PoI

  1: $ti \leftarrow$ time index of sl wrt tp
  2: $di \leftarrow$ day index of sl wrt dg
  3: $sd \leftarrow$ stay duration of sl
  4: $adt \leftarrow ad_{th} \cdot df$ ▷ calculation of distance threshold
  5: $pSet \leftarrow$ set of PoIs within adt meters of sl
  6: $cSet \leftarrow getCategories(pSet)$ ▷ set of categories of PoIs
  7: $cSelected \leftarrow \mathrm{argmax}_{cat \in cSet} P(cat, ti, di, sd)$ ▷ computation of joint probability using Bayesian network bn
  8: **if** $\exists! p \in pSet; \ (p.c = cSelected)$ **then**
  9:     **return** p
10: **else**
11:     **return**   ▷ unique assignment is not possible
12: **end if**

The algorithm is given in Algorithm E.1. It takes a Bayesian network, a stop location, a distance threshold, a factor, a time period duration, and a day granularity as inputs and outputs a PoI that the stop location is assigned to. First, it computes time index, day index, and stay duration values using the input stop location as shown in lines 1–3. Then it calculates the distance threshold in line 4 and filters the PoIs around the given stop location and forms the set of possible categories, in lines 5–6. The filtering also uses opening hours information if it is available. Then it selects the most probable category from the set of possible categories by computing the joint probability of category, stay duration, day index, and time index using the Bayesian network (line 7). The joint probability is computed using Equation E.1.

$$P(cat, ti, di, sd) = P(di) \cdot P(ti) \cdot P(cat \mid di, ti) \cdot P(sd \mid cat) \qquad \text{(E.1)}$$

If only one PoI has the selected category, that PoI is returned. Otherwise, the stop location is not assigned to any PoIs (line 11).

4 Experimental Evaluation

We continue with evaluating the proposed method. We present the experimental setup in Section 4.1. Then we report on studies aiming to understand the effect of parameters in Section 4.2. In Section 4.3, we present the effect of distance based filtering (DBF) on VPE. Finally, we report on the stay duration distributions of the output assignments in Section 4.4.

4.1 Experimental Setup

We use GPS data collected from 354 cars during the period 01/03/2014–31/12/2014 with a frequency of 1 Hz. The trajectory dataset contains around 0.4 billion GPS records and the PoI dataset contains around $10,000$ PoIs of 88 categories. The majority of GPS records and all of the PoIs are located in or around Aalborg, Denmark. The complete dataset is used in all experiments.

We use default values for the parameters controlling stop location extraction and home/work stop location extraction since our focus is assignment of stop locations to PoIs. The parameters Δ_{th}, d_{th}, Δ_{hw}, and p_{hw} are set to 10 minutes, 250 meters, 240 minutes, and 3/7 (three days a week), respectively. With the default parameters, we obtain $349,637$ stop locations, out of which $129,836$ correspond to home/work stops.

In order to evaluate the proposed VPE method, we construct a ground truth dataset. We use a labeled dataset constructed by using the method explained in Section 3.2. The labeled dataset obtained with $ad_{th} = 100$ m contains $36,691$ assignments. The top-5 PoI categories and the number of stop locations which are assigned to a PoI of this category are as follows: supermarket - $3,961$, store - $3,925$, school - $3,020$, restaurant - $2,832$ and lodging - $2,214$.

To evaluate our algorithm and split the labeled data into training and test datasets, we apply 10-fold cross validation. The training dataset is used to learn the Bayesian network, and the test dataset is used to evaluate it. In order to make sure that the test dataset contains stop locations with more than one possible PoI, we extend the region defined by a stop location and ad_{th} using a parameter df. If the number of PoIs in this region exceeds the minimum PoI count parameter (mpc), the stop location is included in the test set. In other words, the test set contains only stop locations with more than mpc PoIs around them.

For experimental purposes, we modify the assignment algorithm used in VPE. Instead of returning a single PoI, it is set to return a list of possible categories sorted according to the joint probability value obtained from the Bayesian network. We report the following metrics: (i) $p@n$ - Precision at position n: The percentage of stop locations for which the category of the PoI, the stop location is assigned to, is in the first n categories in the output list, (ii)

mrr - Mean Reciprocal Rank: The average position of the category of the PoI a stop location is assigned to in the output list, (iii) *npc* - Number of Possible Categories: The average number of possible categories after distance-based filtering.

We also report the results of applying our algorithm without distance-based filtering to show the effect of this filtering. When the assignment algorithm is applied without distance-based filtering, the set of possible categories *cSet* in line 6 of Algorithm E.1 is set to all possible categories in the PoI database.

4.2 Exploring the Parameters

We first explore the effect of the parameters of the proposed method on the stop location assignment. We vary the value of an explored parameter while fixing all other parameters to their default values. The parameters and their default values are given in Table E.1.

Table E.1: Parameters and Default Values

Notation	Name	Default Value
ad_{th}	Distance Threshold	100 meters
df	Distance Factor	2
tp	Time Period	0.5 hours
dg	Day Granularity	1
mpc	Minimum PoI Count	3 PoIs

(a) on *p@n*

(b) on *mrr* and *npc*
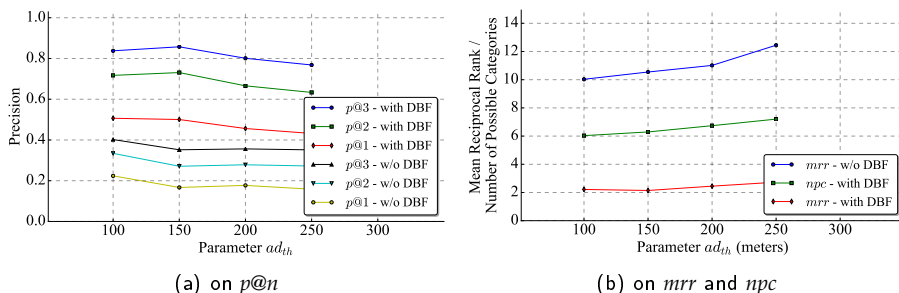
Fig. E.5: Effect of ad_{th}

Figure E.5a shows that precision values decrease when ad_{th} increases since the number of possible categories also increases as shown in Figure E.5b.

However, VPE is still able to achieve a mean reciprocal value of 2 out of 6–8 categories.

(a) on *p@n*

(b) on *mrr* and *npc*
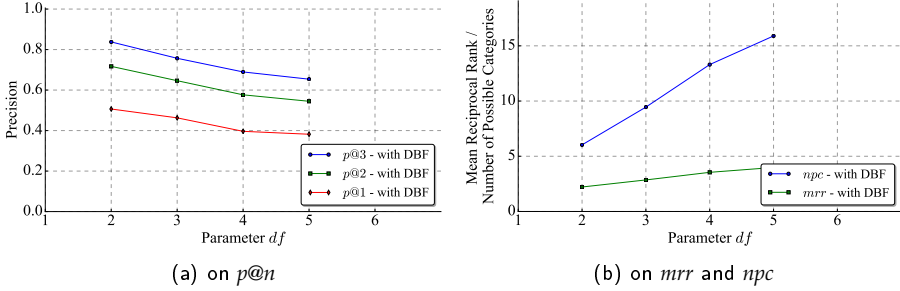
Fig. E.6: Effect of *df*

Figure E.6a shows that the precision decreases when the distance factor increases. This occurs because the number of possible categories increases sharply, as shown in Figure E.6b. The decrease in precision is expected when more categories are possible. However, the increase in number of possible categories is sharper than the increase in mean reciprocal rank, which shows that VPE performs well even though the number of possible categories is high.

(a) on *p@n*

(b) on *mrr* and *npc*
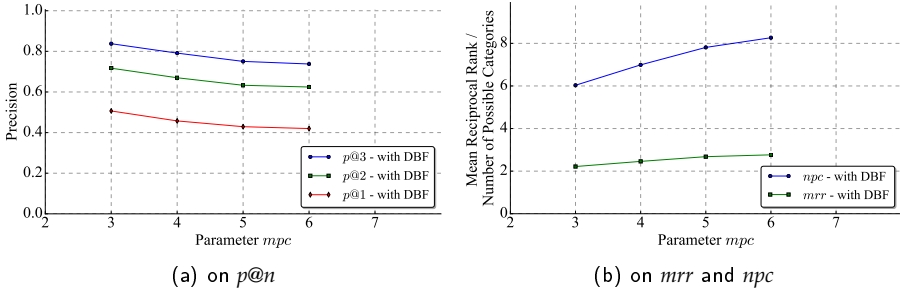
Fig. E.7: Effect of *mpc*

Figure E.7a shows that the precision decreases when the minimum PoI count, and thus the number of possible categories increases,as shown in Figure E.7b. This is also expected since having more PoIs to choose from makes assignment more difficult.

Figures E.8a and E.8b show that time period of a time slot affects the model's performance. We can see that best performance is achieved when the time period is 30 minutes or 1 hour. From these figures, we infer that increasing the time period reduces the proposed method's ability to distinguish

(a) on *p@n*   (b) on *mrr* and *npc*
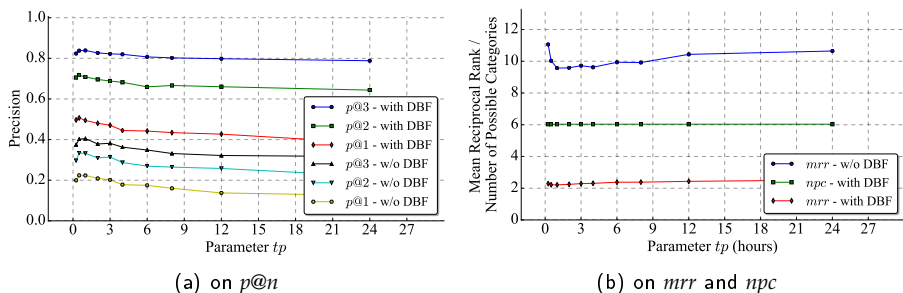
Fig. E.8: Effect of *tp*

PoI categories.

Our experiments show that the day granularity does not have a significant effect on the proposed method. This suggests that even though some specific categories have different day distributions, most of the categories have similar distributions for each day.

4.3 Effect of Distance Based Filtering

Figures E.5a, E.5b, E.8a, E.8b show that the VPE method performs much better with distance-based filtering. For instance, Figures E.5a and E.5b show that it achieves a *p@1* value around 0.5, a *p@3* value around 0.8, and an *mrr* value of between 2 and 2.5 for all ad_{th} values. We can also see that the *p@1* value increases from 0.2 to 0.5 and that the *mrr* value decreases from being in the range 10–12 to being in the range 2–2.5 when we use VPE with distance-based filtering.

4.4 Output Stay Duration Distribution

This set of experiments is designed to check whether the stay duration distributions of the assignments obtained by VPE match with the stay duration distributions in the labeled dataset shown in Figure E.4. For this experiment, we perform assignment of the stop locations that have no assignment in the labeled dataset. We use VPE with the default values and compute the distributions from the assignments obtained. Here, the assignments contain only the assignments obtained using VPE. In other words, the assignments in the labeled dataset are not used when computing the distributions.

The stay duration distributions of the same categories obtained with VPE are shown in Figure E.9. This figure shows that stay duration has a significant effect on the PoI that is assigned to a stop location. For instance, if a stay lasts more than 45 minutes, the probability that the visit is to a grocery
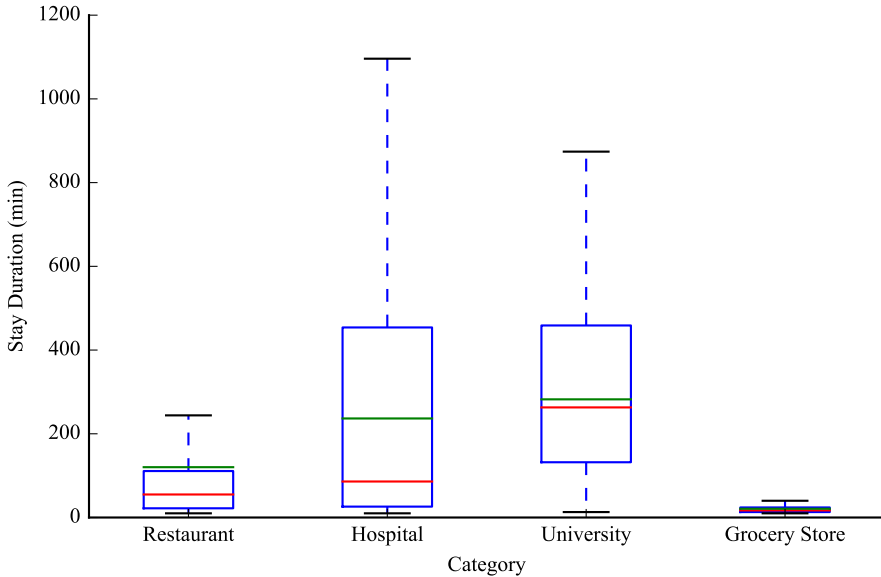
Fig. E.9: Stay Duration Distributions obtained with VPE

store is quite low. Figure E.9 also shows that stay duration distributions of the categories as computed from output assignments are quite similar to the distributions computed from the assignments in the labeled dataset. This suggests that the network structure used in the VPE method together with the log-normal distribution assumption are able to model the relation between stay duration values and categories.

We also designed experiments to explore the degree of importance of the stop location attributes included in the Bayesian network on the PoI category prediction. To explore this, we omit the term in Equation E.1 corresponding to the attribute we are investigating the effect of and then compute the joint probability without it. The experiments show that the stay duration is more important for visited PoI category prediction than the day and time indexes.

5   Conclusion

The paper proposes a visited PoI extraction (VPE) method for vehicle trajectories using a Bayesian network together with distance-based filtering. The Bayesian network represents the relationship between the temporal attributes of a stop and the category of the visited PoI. VPE also includes a method to build a labeled dataset on top of unlabeled GPS records. The experimental results show that the proposed method is capable of detecting the category

of visited PoI from the temporal information, and it achieves a p@3 value of 0.8.

In future work, it is of interest to combine other data sources like check-ins with GPS data in order to achieve a better assignment performance. This might provide more accurate information about arrival day and time distributions.

References

[1] L. O. Alvares, V. Bogorny, B. Kuijpers, J. A. F. de Macedo, B. Moelans, and A. Vaisman, "A model for enriching trajectories with semantic geographical information," in *Proceedings of the 15th Annual ACM International Symposium on Advances in Geographic Information Systems (GIS '07)*. ACM, 2007, pp. 22:1–22:8.

[2] A. T. Palma, V. Bogorny, B. Kuijpers, and L. O. Alvares, "A clustering-based approach for discovering interesting places in trajectories," in *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC '08)*. ACM, 2008, pp. 863–868.

[3] D. Ashbrook and T. Starner, "Using GPS to learn significant locations and predict movement across multiple users," *Personal Ubiquitous Comput.*, vol. 7, no. 5, pp. 275–286, Oct. 2003.

[4] J. H. Kang, W. Welbourne, B. Stewart, and G. Borriello, "Extracting places from traces of locations," in *Proceedings of the 2Nd ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots (WMASH '04)*. ACM, 2004, pp. 110–118.

[5] C. Zhou, D. Frankowski, P. Ludford, S. Shekhar, and L. Terveen, "Discovering personal gazetteers: An interactive clustering approach," in *Proceedings of the 12th Annual ACM International Workshop on Geographic Information Systems (GIS '04)*. ACM, 2004, pp. 266–273.

[6] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma, "Mining interesting locations and travel sequences from GPS trajectories," in *Proceedings of the 18th International Conference on World Wide Web (WWW '09)*. ACM, 2009, pp. 791–800.

[7] X. Cao, G. Cong, and C. S. Jensen, "Mining significant semantic locations from GPS data," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 1009–1020, Sep. 2010.

[8] R. Montoliu, J. Blom, and D. Gatica-Perez, "Discovering places of interest in everyday life from smartphone data," *Multimedia Tools and Applications*, vol. 62, no. 1, pp. 179–207, Jan 2013.

[9] T. Bhattacharya, L. Kulik, and J. Bailey, "Extracting significant places from mobile user GPS trajectories: A bearing change based approach," in *Proceedings of the 20th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL '12)*. ACM, 2012, pp. 398–401.

[10] I. Keles, C. S. Jensen, and S. Saltenis, "Extracting rankings for spatial keyword queries from GPS data," in *Proceedings of the 14th International Conference on Location Based Services (LBS 2018)*. Springer, 2018, pp. 173–194.

[11] K. Nishida, H. Toda, T. Kurashima, and Y. Suhara, "Probabilistic identification of visited point-of-interest for personalized automatic check-in," in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '14)*. ACM, 2014, pp. 631–642.

[12] T. Bhattacharya, L. Kulik, and J. Bailey, "Automatically recognizing places of interest from unreliable GPS data using spatio-temporal density estimation and line intersections," *Pervasive Mob. Comput.*, vol. 19, no. C, pp. 86–107, May 2015.

[13] B. Furletti, P. Cintia, C. Renso, and L. Spinsanti, "Inferring human activities from GPS tracks," in *Proceedings of the 2nd ACM SIGKDD International Workshop on Urban Computing (UrbComp '13)*. ACM, 2013, pp. 5:1–5:8.

[14] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*. AAAI Press, 1996, pp. 226–231.

References

Paper F

Transportation-mode Aware Spatial Keyword Querying for Point of Interest Clusters

Ilkcan Keles, Dingming Wu, Simonas Šaltenis, Christian S. Jensen

Ready for submission

Abstract

Spatial keyword queries retrieve points of interest that are relevant to the supplied keywords and that are close to the query location. We propose a new spatial keyword query, namely the top-k transportation-mode aware spatial textual clusters query (k-TMSTC) that returns k density-based clusters of relevant spatial objects. We propose DBSCAN-based and OPTICS-based algorithms to process k-TMSTC queries. We also propose an algorithm to find the kth nearest neighbor distance for all objects. We evaluate the proposed algorithms on a real dataset and show that our proposal is capable of processing k-TMSTC queries in interactive time.

The layout has been revised.

1 Introduction

With the proliferation of geo-tagged content on the web and the extensive use of mobile phones, the number of people who search for local places using mobile or web applications increase day by day. A recent study on users' search behavior [1] reports that 4 out of 5 users search for geographically related information. In addition, 50% of the mobile users and 34% of the web users who conducted search for points of interest (PoIs) visit one of the PoIs on the same day. These statistics indicate the importance of location-based queries for search engines and mobile applications.

To support queries with local intent, the research community has proposed different variants of spatial keyword queries [2–11]. Most of the existing proposals return lists of individual PoIs that are ordered with respect to a scoring function. The Boolean kNN query [4, 9] and the top-k spatial keyword query [2, 3, 5–7, 10] are examples of this type. However, a user may be interested in small regions containing many relevant PoIs instead of individual PoIs that may be located far away from each other. This might happen when a user wants to visit multiple PoIs or wants to browse or explore different options before making a final decision. For instance, a tourist looking for attractions would prefer attractions that are located close to each other instead of traveling long distances between different attractions. Another example is a user who wants to buy jeans. This user may prefer visiting a region with several shops to check the prices before deciding where to buy.

We propose the **top-k transportation-mode aware spatial textual cluster (k-TMSTC)** query to address browsing or exploratory user behavior. A k-TMSTC query takes a query location, a set of keywords, an integer value k, and transportation modes for both to-cluster and in-cluster travel as arguments. The query identifies k clusters of relevant PoIs to make it possible for the user to browse different relevant PoIs within a cluster. Each cluster contains relevant PoIs with respect to the query keywords, and the clusters are ranked with respect to a cost function. In order to take the user's transportation modes into account, we propose two new cost functions, namely a spatial and a spatio-textual cost function. An example dataset with textual relevance weights is illustrated in Table F.1a. An example 3-TMSTC query with query keyword "jeans" on this dataset and the result clusters are illustrated in Figure F.1b. The grid represents a city, q represents the query location, and the p_is represent shops throughout the city.

Existing studies have proposed query types to target the same user behavior. Skovsgaard et al. [12] propose the top-k groups spatial keyword query that takes a query location, a set of keywords, and a value k as parameters and returns top-k groups of PoIs ordered according to a cost function. The main limitation of this query type is that it does not consider in-group dis-

	jeans	tshirt	hat
p_1	0.25	0.25	0.25
p_2	0.2	0	0
p_3	0.3	0.7	0
p_4	0	0.2	0.5
p_5	0.25	0.4	0
p_6	0.5	0	0.2
p_7	0.45	0	0
p_8	0	0.3	0.1
p_9	0.1	0.4	0.4
p_{10}	0	0	0.5
p_{11}	0.2	0	0.3
p_{12}	0.8	0	0
p_{13}	0.3	0.2	0.1
p_{14}	0	0	0

(a)
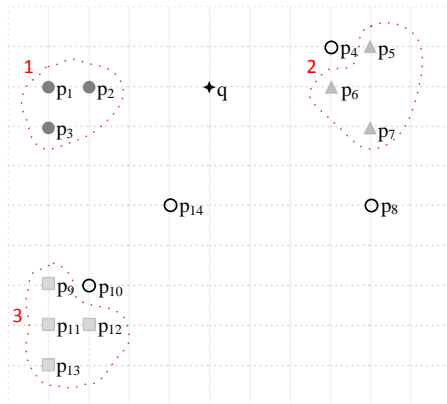
(b)

Fig. F.1: Example Dataset and Example k-TMSTC Query with Keyword "jeans"

tances properly. The cost function utilized by top-k groups query considers the diameter of a group, which is the maximum distance between a pair of PoIs in the group. We find that it would be more convenient for the users to consider the actual distance that the user needs to travel instead of considering the diameter. If the diameter of the group is considered, another PoI can be added to the group without changing the cost if it does not affect the diameter. However, doing so might change the distance the user needs to travel to visit all the PoIs in the group. Wu and Jensen [11] propose the top-k spatial textual cluster (k-STC) query that takes a query location, a set of keywords, a value k, and density-based clustering parameters. It returns top-k clusters ordered with respect to a cost function. The main limitation of this query type is that it expects users to provide density based clustering parameters for a region that they might not have any information about. It may be difficult for a user to provide appropriate values for the clustering parameters since it requires knowledge about the distribution of relevant PoIs. Further, this query type does not consider the user's transportation mode while ranking the clusters. We believe that a user's transportation mode affects the user's preferences on how much they would like to travel to and within a group of PoIs.

We propose two algorithms to process k-TMSTC queries. The first is based on the DBSCAN algorithm [13] and is built on top of the algorithms proposed to process k-STC queries [11]. It first determines the parameters for density-based clustering using the method employed by the VDBSCAN

algorithm [14]. This method finds k^{th} nearest neighbor distances of PoIs and checks the difference between them to determine different density levels. It returns parameters corresponding to the different density levels. The DBSCAN-based algorithm then constructs a set of k-STC queries with these parameters and process these queries in parallel. The output clusters are ranked with respect to proposed spatial and spatio-textual cost functions. The second algorithm is based on the OPTICS algorithm [15] and creates a cluster ordering using the OPTICS algorithm. It determines the parameters while creating the cluster order, and then it constructs clusters corresponding to different density levels using the cluster order. Therefore, the OPTICS-based algorithm does not have a separate parameter determination phase.

We also propose an algorithm called FastKDist that is based on the SGPL index structure that aims to efficiently determine the list of kth nearest neighbor distances for all PoIs in a set of query-relevant PoIs. In other words, the FastKDist algorithm basically processes a k-dist query for each relevant object, which returns the distance of the kth nearest neighbor to the relevant object. For each PoI in the set, it creates a visit order of the grid cells to find the kth nearest neighbor.

To summarize, our main contributions are:

- Definition of the k-TMSTC query to support exploratory and browsing behavior.

- Two algorithms based on the DBSCAN and OPTICS to process k-TMSTC queries.

- An algorithm (FastKDist) that finds k^{th} nearest neighbor distances for all objects in a set.

- An evaluation of the proposed algorithms using a dataset from Yelp[1] that contains around $150,000$ PoIs.

The rest of the paper is organized as follows. Section 2 covers the related work, and Section 3 defines the top-k transportation mode aware spatial textual cluster query. We present the proposed algorithms in Section 4. The DBSCAN-based algorithm and the OPTICS-based algorithm are presented in Sections 4.2 and 4.3, respectively. We report on the experimental evaluation in Section 5, and Section 6 concludes the paper.

2 Related Work

Spatial keyword querying has attracted substantial attention in recent years. A prototypical spatial keyword query [16] takes the following parameters: a

---

[1]https://www.yelp.com/

query location, a set of keywords, and a value k representing the cardinality of the result set. It retrieves spatial web objects, or PoIs, that are close to the query location and textually relevant to the query keywords. Several efficient spatio-textual indexes have been proposed to support spatial keyword querying. Most of these indexes combine the R-tree with a text indexing method like inverted files and signature files such as the IR^2-tree [2], the IR-tree [3, 6] and S2I [5]. These hybrid index structures are capable of utilizing both spatial and textual information to prune the search space when processing a query. We adopt the IR-tree [3] in this work.

Most of the existing spatial keyword query proposals have single-object granularity. These proposals retrieve a list of objects that are close to the query location and textually relevant to the query keywords. Some proposals [4, 9] use the query keywords as Boolean predicates to filter out the objects that are not textually relevant. Then, the remaining objects are ranked with respect to their distances to the query location. Other proposals [2, 3, 5–7, 10] combine textual relevance and spatial proximity to rank the spatial objects. Furthermore, a range of studies investigate variants of the prototypical spatial keyword query that address different use cases. The location aware top-k prestige based text retrieval query [17] retrieves the top-k spatial web objects ranked according to both prestige-based relevance and spatial proximity, where the prestige-based relevance captures both the textual relevance of an object and the effects of nearby objects. Li et al. [18] propose a spatial query that considers the direction of the user. This query also uses the keywords as Boolean predicates and retrieves a list of objects ranked with respect to the spatial proximity to the query location. Wu et al. [19, 20] cover the problem of maintaining the result set of top-k spatial keyword queries while the user (query location) moves continuously.

Some spatial keyword query proposals focus on retrieving a single set of objects. Collective spatial keyword queries [21–24] return a set of objects such that the set collectively covers the query keywords and the cost of the set is minimized. The cost of a set can be defined in terms of the spatial distances of the objects in the set to the query location, the spatial distances between the objects in the set or the inherent costs of the objects within the set. Collective spatial keyword queries cannot be utilized for the browsing user behavior we target in this work since the objects within the set collectively covers the query keywords and the users are interested in browsing alternatives.

Some studies have proposed query types to target browsing user behavior. Skovsgaard et al. [12] propose a top-k groups spatial keyword query that returns top-k groups with respect to a cost function. The cost function takes into account the textual relevance of the PoIs inside the group, the diameter of the group, which is the maximum pairwise distance between two objects in the group, and the minimum distance between an object in the group and the query location. They propose the Group Extended R-tree (GER-tree) to effi-

ciently process top-k groups spatial keyword queries. The GER-tree is based on the R-tree [25], and it maintains information about the textual representation of the objects and their inter-object distances by utilizing compressed histograms. The top-k groups keyword query has two drawbacks. First, the in-group distance is modeled by utilizing the diameter of the group. We find that the in-group distance should be modeled as the actual distance that a user needs to travel to visit all the objects in a group. Second, the cost function utilized by this query does not consider the user's transportation mode to the group and within the group. In their demo paper [26], the authors propose a work-around to take the transportation mode into account by changing the weighting parameter utilized in the cost function. In fact, they adjust the weights given to the spatial distances and the textual relevance to account for a specific transportation mode. We find that it would be more realistic if the cost is defined with respect to the duration needed to visit all the places within the group according to a particular transportation mode.

Wu and Jensen [11] propose the top-k spatial textual cluster (k-STC) query that takes additional density-based clustering parameters together with the query parameters and returns top-k clusters ordered with respect to a cost function. The cost function considers the minimum distance between the query and the cluster and the textual relevance of the objects inside the cluster. They propose a basic DBSCAN-based algorithm to process k-STC queries utilizing the IR-tree [3]. They propose three advanced query processing methods that address inefficiencies of the basic approach. The first method reduces the number of range queries needed by checking whether the neighborhood of an object has already been examined. The second method utilizes a grid-based index structure, named Spatially Gridded Posting Lists (SGPL), to estimate the number of objects that might be included in the result of a range query. A range query is not issued if the estimated number of objects is not high enough. An SGPL on a regular grid is constructed for each term, and each grid cell contains the set of objects that is located inside the cell and that contains the term. The third method utilizes the SGPL data structure to process range queries efficiently. There are two issues with k-STC queries. First, the user is expected to provide the density-based clustering parameters. The problem of expecting these parameters from users is that users are unable to have sufficient information about the region where the query is issued or about the distribution of PoIs around the region. Second, the transportation mode is not taken into account in the cost function. We think that doing so is important when forming and ranking clusters. For instance, a user with a car might not care about travelling 20 km to a cluster, but if the user has a bicycle then 20 km is likely an unattractive distance to travel. In this work, we extend k-STC queries to overcome these issues. We also utilize the methods proposed to process k-STC queries in order to process k-TMSTC queries.

3 Problem Definition

We consider a data set D in which each object $p \in D$ is a pair $\langle \lambda, d \rangle$ representing a point of interest (PoI), where $p.\lambda$ is a point location and $p.d$ is a document containing a textual description of the PoI. The document $p.d$ is represented by a weight vector $(w_1, w_2, ..., w_n)$ in which each dimension corresponds to a distinct term t_i in the data set. The weight w_i of a term t_i in a document can be computed in several different ways, e.g., using tf-idf weighting [27] or language models [28]. We employ the function $tr(p, \psi) = \sum_{t_i \in \psi} p.d.w_i$ to assess the textual relevance of an object (p) to a given set of words (ψ).

Definition F.1. *Given a location λ, a set ψ of keywords, and an upper bound for the distance Δ_{ub}, the **relevant object set** D_ψ satisfies (i) $D_\psi \subseteq D$, and (ii) $\forall p \in D_\psi$ ($tr(p, \psi) > 0 \land \|p\,\lambda\| \leq \Delta_{ub}$), where $\|p, \lambda\|$ denotes the distance between the location λ and the PoI p.*

We adopt the density based clustering model [13], and clusters are query dependent. We proceed to present the relevant definitions from density based clustering. We provide these definitions in order to define top-*k* spatial textual cluster and top-*k* transportation mode aware spatial textual cluster queries. The density based clustering related definitions and the definition of top-*k* spatial textual cluster query are reproduced from [11].

Definition F.2. *The ϵ-**neighborhood** of a relevant object $p \in D_\psi$, denoted by $N_\epsilon(p)$, is defined as $N_\epsilon(p) = \{p_i \in D_\psi \mid \|p\,p_i\| \leq \epsilon\}$.*

Definition F.3. *A **dense** ϵ-**neighborhood** of a relevant object $N_\epsilon(p)$ contains at least minpts objects, i.e., $|N_\epsilon(p)| \geq minpts$. An object whose ϵ-neighborhood is dense is called a **core object**.*

Definition F.4. *A relevant object p_i is **directly reachable** from a relevant object p_j with regard to ϵ and minpts if (i) $p_i \in N_\epsilon(p_j)$ and (ii) p_j is a core object.*

Definition F.5. *A relevant object p_i is **reachable** from a relevant object p_j with regard to ϵ and minpts if there is a chain of relevant objects $p_1, ..., p_n$, where $p_1 = p_i$, $p_n = p_j$, and p_m is directly reachable from p_{m+1} for $1 \leq m < n$.*

Definition F.6. *A relevant object p_i is **connected** to a relevant object p_j with regard to ϵ and minpts if there is a relevant object p_m such that both p_i and p_j are reachable from p_m with regard to ϵ and minpts. The connected relation is symmetric unlike the reachable relation.*

Having the definitions related to density based clustering, we now proceed to define spatial textual cluster and the query types.

Definition F.7. *A **spatial textual cluster** C with regard to ψ, ϵ, and minpts satisfies the following conditions: (i) $C \subseteq D_\psi$ and (ii) C is a maximal set such that $\forall p_i, p_j \in C$, p_i and p_j are connected through dense ϵ-neighborhoods when considering relevant objects. A spatial textual cluster is a density-based cluster [13] found from the relevant object set D_ψ.*

We now proceed to define top-*k* spatial textual cluster query. We utilize this query type to process *k*-TMSTC queries.

Definition F.8. *A **top-k spatial textual cluster (k-STC) query** is defined by Wu and Jensen [11] as $q = \langle \lambda, \psi, k, \epsilon, minpts \rangle$, where λ is a point location , ψ is a set of keywords, k is the number of result clusters, and ϵ and minpts are the density-based clustering parameters. It returns a list of k spatial textual clusters that are sorted in ascending order of their scores computed by a scoring function. The scoring function employed is given in Equation F.1, where $d_{q.\lambda}(C)$ is the minimum spatial distance between the query location and the objects in C and $tr_{q.\psi}(C)$ is the maximum text relevance in C. Parameter α is used as a balancing factor between spatial proximity and textual relevance.*

$$score_q(C) = \alpha \cdot d_{q.\lambda}(C) + (1 - \alpha) \cdot (1 - tr_{q.\psi}(C)) \tag{F.1}$$

Definition F.9. *A **top-k transportation-mode aware spatial textual clusters (k-TMSTC)** query is defined as $q = \langle \lambda, \psi, k, tm_c, tm_i \rangle$, where λ is a point location, ψ is a set of keywords, k is the number of result clusters, and tm_c and tm_i are transportation modes for traveling to the cluster and traveling in the cluster, respectively. A cluster is defined as a subset of D_ψ and a k-TMSTC query identifies the k best maximal density-based clusters with respect to a cost function. The transportation mode can be one of the following: driving, cycling, walking, and public transportation.*

It is important to note that we extend the *k*-STC query definition given by Wu and Jensen [11] by removing the ϵ and *minpts* parameters and taking transportation-mode information into account. We think that it is difficult for a user to state a purposeful pair of ϵ and *minpts* parameters. We also think that a user's transportation modes play an important role in the user's preferences. To illustrate, a tourist who rented a car might be willing to travel farther than a tourist who just walks to visit attractions. Since we are still interested in density-based clusters, a proposal for top-*k* transportation-mode aware spatial textual cluster query should contain a method to determine the ϵ and *minpts* parameters automatically. Further, we allow these parameters to be different for each returned cluster.

Intuitively, the cost function should consider the distance between the query location and the cluster as well as the transportation mode both to the cluster and in the cluster. It should also take the cluster size and the textual relevances of the objects in the cluster into account. In order to achieve

these, we first define a spatial cost function and then extend it with textual relevance. The spatial cost function, given in Equation F.3, only considers the duration of traveling as the cost measure and defines the cost as the duration of traveling needed per PoI. In Equation F.3, $td(q, C)$ refers to the travel duration needed to visit all the PoIs. The definition $td(q, C)$ is provided in Equation F.2, where $md(q.\lambda, C)$ is the minimum distance between the query location and the cluster, ud_{tc} and ud_{ic} are durations corresponding to one distance unit for the transportation modes to the cluster and in the cluster, respectively, and $d(C)$ is the total distance that needs to be traveled in the cluster. The distance in the cluster is the length of a Hamiltonian path visiting all PoIs. For simplicity, we compute such a path greedily. The first PoI in the path is the PoI closest to the query location, and the next PoI is always chosen as the closest unvisited PoI from the previous PoI.

$$td(q, C) = md(q.\lambda, C) \cdot ud_{tc} + d(C) \cdot ud_{ic} \tag{F.2}$$

$$cost_{sp}(q, C) = \frac{td(q, C)}{|C|} \tag{F.3}$$

The spatio-textual cost function is given in Equation F.4, where $tr(p, q.\psi)$ is the textual relevance of p to the query keywords $q.\psi$. It considers both the duration of traveling needed and the textual relevance of the objects within the cluster. The cost in this version is modeled as the duration of traveling needed per textual relevance.

$$cost_{st}(q, C) = \frac{td(q, C)}{\sum\limits_{p \in C} tr(p, q.\psi)/|C|} \tag{F.4}$$

Example: Let us assume that ud_{tc} and ud_{ic} are equal to 1. Then, the travel durations for clusters $C_1 = \{p_1, p_2, p_3\}$ and $C_2 = \{p_5, p_6, p_7\}$ in the example provided in Figure F.1b are $td(q, C_1) = 5$ and $td(q, C_2) = 5 + \sqrt{2}$. The spatial costs are $cost_{sp}(q, C_1) = 5/3$ and $cost_{sp}(q, C_2) = (5 + \sqrt{2})/3$. The average textual relevance within the clusters C_1 and C_2 are 0.25 and 0.4, respectively. So, the spatio-textual costs are $cost_{st}(q, C_1) = 20$ and $cost_{st}(q, C_2) = 16.04$. So, if we utilize the second cost function while ranking the clusters, the ordering should change, and C_2 should be the first cluster.

Problem Statement. Given the definitions above, the problem we tackle is to develop an efficient algorithm to process k-TMSTC queries with a response time that supports interactive search.

4 Proposed Method

In the following, we propose two algorithms to process the k-TMSTC query. We first explain the index structures used for indexing PoIs in Section 4.1. Sections 4.2 and 4.3 present the algorithms for the processing of k-TMSTC queries.

4.1 Indexes

We employ the IR-tree [3] and spatially gridded posting lists (SGPL) [11] to index the objects.

The IR-tree is an R-tree [25] extended with inverted files [29]. An inverted file index has two main components: (i) A vocabulary of all distinct words appearing in the text descriptions of the objects in the data set being indexed, and (ii) a posting list for each word t, i.e., a sequence of pairs (id, w), where id is the identifier of an object whose text description contains t and w is the word's weight in the object. Each leaf node of an IR-tree contains entries of the form $o = (id, \lambda)$, where $e.id$ refers to an object identifier and $o.\lambda$ refers to a minimum bounding rectangle (MBR) of the spatial location of the object. Each leaf node also contains a pointer to an inverted file indexing the documents of all objects stored in the node. Each non-leaf node contains entries of the form $e = (id, \lambda)$ representing the children of the node where $e.id$ is a child node identifier and $e.\lambda$ is the MBR of all entries contained in the child node identified by $e.id$. Each non-leaf node also contains a pointer to an inverted file indexing the text descriptions of the entries stored in the node's subtree.

Example: An IR-tree with fanout 4 indexing the example dataset given in Table F.1a is illustrated in Figure F.2.
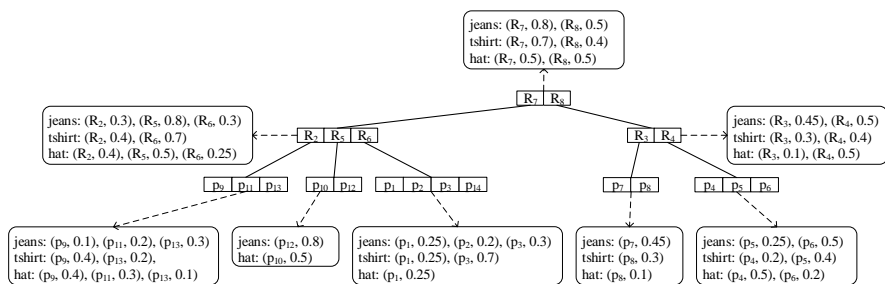
Fig. F.2: Example IR-tree

A SGPL [11] is a grid-based index structure proposed for selectivity estimation and processing of range queries. First, an $n \times n$ grid is created on the data set, and the grid cells are indexed by a space filling curve. Then, for each

word w_i, a SGPL is created. The SGPL of w_i is a sorted list of entries of the form $\langle c_j, S_{w_i,c_j} \rangle$, where c_j is the index value of a grid cell and S_{w_i,c_j} is a set of objects that contain w_i in their documents and are located in cell c_j. Although we only show the identifiers of the PoIs in the example SGPL below for the sake of simplicity, the index structure also stores the location and the textual weight associated with w_i.

jeans	$(0, \{p_{11}, p_{12}, p_{13}\})$, $(1, \{p_9\})$, $(4, \{p_3\})$, $(5, \{p_1, p_2\})$, $(14, \{p_7\})$, $(15, \{p_5, p_6\})$
tshirt	$(0, \{p_{13}\})$, $(1, \{p_9\})$, $(4, \{p_3\})$, $(5, \{p_1\})$, $(14, \{p_8\})$, $(15, \{p_4, p_5\})$
hat	$(0, \{p_{11}, p_{13}\})$, $(1, \{p_9, p_{10}\})$, $(5, \{p_1\})$, $(14, \{p_8\})$, $(15, \{p_4, p_6\})$

(a) 4×4 Grid   (b) SGPL
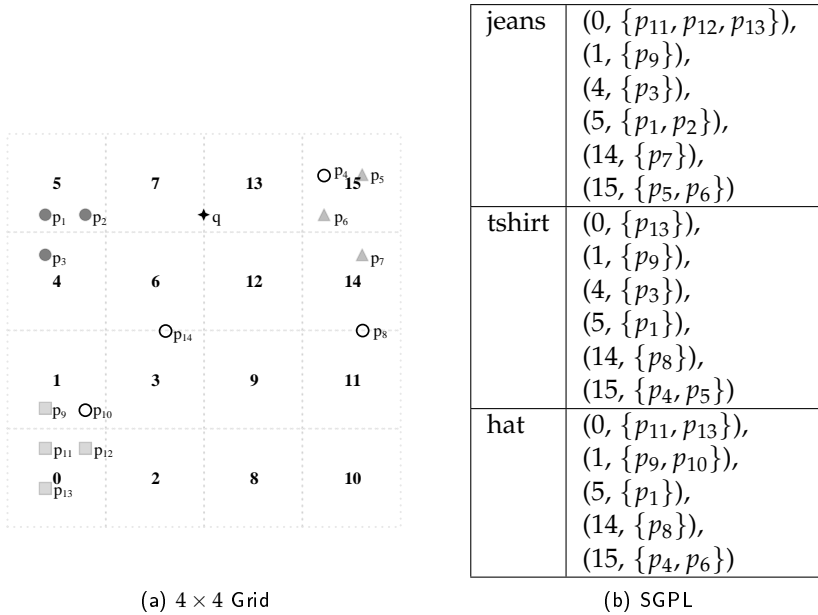
Fig. F.3: Example Spatially Gridded Posting Lists

Example: Figure F.3a illustrates a 4×4 grid on the 14 objects given in Table F.1a. Grid cells are indexed using a 2-order Z-curve. The numbers in bold are the Z-values of the cells. Table F.3b shows the SGPLs for words "jeans", "tshirt", and "hat". For instance, the first entry of "hat" indicates that the grid cell with Z-value 0 has two PoIs (p_{11} and p_{13}) that contain "hat" in their documents.

4.2 DBSCAN-based Algorithm

We present the DBSCAN-based algorithm that uses the IR-tree. Then we present how to use the SGPL instead of the IR-tree for the DBSCAN-based algorithm.

Algorithm

The DBSCAN-based algorithm is built on top of the algorithms proposed to process *k*-STC queries. Given a query, we determine the set of possible ϵ and *minpts* values and construct the set of *k*-STC queries. Then we process these queries in parallel using the methods proposed by Wu and Jensen [11]. The clusters are then sorted with respect to one of the cost functions given in Equation F.3 and Equation F.4, and the top-*k* disjoint clusters are returned as the result.

Algorithm F.1 DBSCAN-based Algorithm

Input: *irtree* - IR-tree, *q* - *k*-TMSTC query, *minmp* - Minimum *minpts* value, *maxmp* - Maximum *minpts* value, inc_{th} - Increase threshold value to determine ϵ values

Output: *clusters* - The list of clusters

1: $\Delta_{ub}, \epsilon_{ub} \leftarrow$ **GetBounds**$(q.tm_c, q.tm_i)$;
2: $D_\psi \leftarrow$ *irtree*.**RangeQuery**$(q.\lambda, q.\psi, \Delta_{ub})$;
3: *dbscanParams* $\leftarrow \varnothing$;
4: **for** *minpts* = *minmp* **to** *maxmp* **do**
5:     *paramList* $\leftarrow$ **GetEpsValues**$(irtree, q, D_\psi, minpts, \epsilon_{ub}, inc_{th})$;   ▷ Gets executed in parallel using fork-join model
6:     *dbscanParams* $\leftarrow$ *dbscanParams*.**Add**(*paramList*);
7: **end for**
8: *queries* $\leftarrow$ The list of *k*-STC queries corresponding to the query *q* and *dbscanParams*;
9: *cSet* $\leftarrow \varnothing$;
10: **for all** *query* in *queries* **do**
11:     *cq* $\leftarrow$ **ProcessKstcQuery**$(query, irtree, D_\psi)$;   ▷ Gets executed in parallel using fork-join model
12:     *cSet* $\leftarrow$ *cSet* $\cup$ *cq*;
13: **end for**
14: *clusters* $\leftarrow$ top-*k* disjoint clusters of *cSet* sorted with respect to the cost;
15: **return** *clusters*;

The DBSCAN-based algorithm is given in Algorithm F.1. Given a query, the algorithm first determines the upper bounds for the distance between the query location and the cluster (Δ_{ub}) and for the ϵ parameter of the DBSCAN algorithm (ϵ_{ub}) according to the transportation mode parameters tm_c and tm_i (line 1). We assume that the algorithm has access to a rule-based method (the **GetBounds** call in the algorithm) to determine the upper bounds according to the transportation mode parameters and the underlying spatial region. For instance, if a user specifies that he plans to walk to the cluster, the Δ_{ub} should not be set to more than 3–4 kilometers. However, if he plans to drive then Δ_{ub}

can be set to 40–50 kilometers. We can determine ϵ_{ub} similarly. For instance, if a user plans to walk within the cluster, the ϵ_{ub} should not be set to more than 0.5–1 kilometers. However, if the user plans to cycle then it can be set to 2–3 kilometers.

Then, the algorithm obtains the relevant object set D_ψ with regard to the query keywords by issuing a range query using the query location, query keywords, and the upper bound for the distance, Δ_{ub} (line 2). The algorithm then determines the possible ϵ values for *minpts* values in the range of [*minmp–maxmp*] and the given query using the relevant object set (lines 3–7). The algorithm has an additional increase percentage threshold inc_{th} to be used in determining ϵ values. We set it to 5% by default. To determine possible DBSCAN parameters, the algorithm first initializes *dbscanParams* as an empty set (line 3). Then the list of $(\epsilon, minpts)$ pairs (*paramList*) is determined for each *minpts* value in parallel. At the end of each parallel execution, the resulting *paramList* is appended to *dbscanParams* (line 6). The default value for the maximum *minpts* value (*maxmp*) is set to 10 since we do not think a user can visit more than 10 PoIs according to a query result. Furthermore, the default value for the minimum *minpts* value (*minmp*) is set to 3 since users are interested in clusters and since we do not want to miss small clusters in the result.

After the DBSCAN parameters are determined, the algorithm constructs the corresponding *k*-STC query for each DBSCAN parameter tuple and processes them in parallel (lines 8–13). It uses a simple fork-join model to process the queries in parallel. The algorithm initializes the set of clusters *cSet* to an empty set (line 9) and populates the set with the results of the queries by forming the union with *cq*, which is the clusters for the current query (lines 11 and 12). Then the clusters are sorted in ascending order with respect to their cost, and the top-*k* disjoint clusters with the least cost are returned as the result (lines 14 and 15).

In the following, we describe the three subroutines called from the main algorithm.

The function **RangeQuery** (Algorithm F.2) is used to find the relevant objects in the ϵ-neighborhood of a given location using an IR-tree on the objects. It is a standard range query algorithm with an additional check for text relevance (lines 8 and 14).

The DBSCAN-based algorithm uses the approach employed by the VDB-SCAN algorithm [14] to determine the ϵ values. VDBSCAN first computes the *k*-dist values for each object in the dataset for a given *k* value and determines the cut-off points from the *k*-dist plot. The *k*-dist value for an object is the distance between the object and the kth nearest neighbor. We utilize the function **GetEpsValues** (Algorithm F.3) to determine possible ϵ values using the IR-tree given a query, a *minpts* value, an upper bound(ϵ_{ub}) for ϵ, a relevant object set, and an increase percentage threshold (inc_{th}). The algo-

---

Algorithm F.2 RangeQuery(λ, ψ, ϵ)

Input: λ - Query location, ψ - Query Keywords, ϵ - Query Range
Output: *neighbors* - The list of neighbors

1: Queue *queue* $\leftarrow \emptyset$;
2: *queue*.**Enqueue**(*root*);
3: **while** *queue* is not empty **do**
4:     $e \leftarrow queue$.**Dequeue**();
5:     $N \leftarrow$ **ReadNode**(e);
6:     **if** N is a leaf node **then**
7:         **for all** object o in N **do**
8:             **if** o is relevant to ψ and $\|\lambda\, o.\lambda\|_{min} \leq \epsilon$ **then**
9:                 *neighbors*.**Add**(o);
10:             **end if**
11:         **end for**
12:     **else**
13:         **for all** object e' in N **do**
14:             **if** e' is relevant to ψ and $\|\lambda\, e'\|_{min} \leq \epsilon$ **then**
15:                 *queue*.**Enqueue**(e');
16:             **end if**
17:         **end for**
18:     **end if**
19: **end while**
20: **return** *neighbors*;

---

rithm initializes the list of *k*-dist values as an empty list. Then it determines the *k*-dist value for $k = minpts$ for each relevant object and adds it to the list if the *k*-dist value is not UNDEFINED (lines 3–8). The function **GetKDist** returns the *k*-dist value if it is less than ϵ_{ub}. Otherwise, it is UNDEFINED. Next, the algorithm checks if the list is populated. If not, it returns the empty list (lines 9–11), which means that the algorithm is unable to find ϵ values for the given parameters.

The list is then sorted. After sorting, the algorithm iterates over the list of *k*-dist values and determines the cut-off points for different density levels in the relevant object set (lines 13–27). We define a density level as a sorted list of *k*-dist values that contains at least *minpts* values, do not have a percentage of increase between the consecutive *k*-dist values exceeding the given inc_{th}, and has a percentage of increase exceeding inc_{th} after the last *k*-dist value in the list. The algorithm checks whether the percentage of increase is more than inc_{th} between previous and current values (line 17). If there is an increase and the number of consecutive *k*-dist values without the required percentage of increase (*nic*) exceeds *minpts*, the algorithm adds a pair of the current *k*-

---

Algorithm F.3 GetEpsValues(*irtree*, *q*, D_ψ, *minpts*, ϵ_{ub}, inc_{th})

Input: *irtree* - IR-tree, *q* - *k*-TMSTC query, D_ψ - Relevant object set, *minpts* - The *minpts* value, ϵ_{ub} - The upper bound value for ϵ, inc_{th} - Increase percentage threshold

Output: *params* - the list of DBSCAN parameter tuples (ϵ, *minpts*)

1: *params* $\leftarrow \emptyset$;
2: *kdistValues* $\leftarrow \emptyset$;
3: **for all** object *o* in D_ψ **do**
4:     *kdist* $\leftarrow$ *irtree*.**GetKDist**($o.\lambda$, $q.\psi$, ϵ_{ub}, *minpts*) ;
5:     **if** *kdist* is not UNDEFINED **then**
6:         *kdistValues*.**Add**(*kdist*);
7:     **end if**
8: **end for**
9: **if** *kdistValues*.**Size** $= 0$ **then**
10:     **return** *params*;
11: **end if**
12: **Sort**(*kdistValues*);
13: *prev* $\leftarrow 0$;
14: *nic* $\leftarrow 0$;   ▷ The number of *k*-dist values that do not have the required increase percentage (inc_{th}).
15: **for** $i = 1$ **to** *kdistValues*.**size do**
16:     *curr* $\leftarrow$ *kdistValues*[*i*];
17:     **if** *prev* $\neq 0 \wedge$ the increase percentage between *curr* and *prev* exceeds inc_{th} **then**
18:         **if** *nic* $\geq$ *minpts* **then**
19:             *params*.**Add**((*curr*, *minpts*));
20:         **end if**
21:         *nic* $\leftarrow 0$;
22:     **else**
23:         *nic* $\leftarrow$ *nic* $+ 1$;
24:     **end if**
25:     *prev* $\leftarrow$ *curr*;
26: **end for**
27: **return** *params*;

---

dist value corresponding to a density level and *minpts* input to the list of parameters (lines 18 and 19). If there is an increase, the algorithm sets *nic* to 0 (line 21). If the increase is insufficient, *nic* is incremented (line 23). The algorithm terminates when the list of *k*-dist values is exhausted.

Example. Let us assume that the PoIs provided in the example dataset given in Figure F.1 are the relevant PoIs for a *k*-TMSTC query. We set *minpts*

to 3, ϵ_{ub} to 5 units, and inc_{th} to 15%.

1	1.41	1.41	1.41	1.41	2	2	2	2.24	3.61	3.61	3.61	4	4.24
0	0	1	2	3	0	1	2	3	0	1	2	3	4
0	41	0	0	0	41.84	0	0	12	61.16	0	0	9.75	6

Table F.1: List of k-dist values and corresponding *nic* and increase percentage values

The first row of Table F.1 shows the sorted list of 3-dist values for the relevant objects. The algorithm iterates over the list and computes the *nic* and increase percentage values. The second and third rows of the table show the *nic* and increase percentage values, respectively. The output ϵ values are 2 and 3.61 since the increase percentage values exceed the given inc_{th} and *nic* is equal to 3 for both 3-dist values. However, 1.41 is not included in the output set since the corresponding *nic* value is below the *minpts* parameter.

The algorithm to determine ϵ values employs the IR-tree on the objects to determine the distance of the kth nearest neighbor to the object. Function **GetKDist** is quite similar to **RangeQuery** given in Algorithm F.2. The query location parameter is the object's location, and the query range is set to ϵ_{ub}. This function has an additional parameter ($k = minpts$), which is the order of the neighbor whose distance is of interest. The algorithm employs a priority queue instead of a regular queue, and the nodes are added to the queue with the priority value being their distance to the query location. Except from using a priority queue, the only part that is different in the algorithm is line 9. Instead of adding to the neighbors list, the algorithm counts the number of neighbors it has processed. If the current object is the kth neighbor, it just returns the distance between the input location and the current object.

Algorithm F.4 ProcessKstcQuery$(irtree, q, D_\psi)$

Input: *irtree* - IR-tree, q - k-STC query, D_ψ - Relevant object set
Output: *clusters* - the list of clusters
  1: $slist \leftarrow$ sort objects in D_ψ in ascending order of $d_{q.\lambda}(o)$;
  2: $clusters \leftarrow \emptyset$;
  3: **while** $slist \neq \emptyset$ **do**
  4:     $o \leftarrow$ first element in *slist*;
  5:     $c \leftarrow$ **GetCluster**$(o, q, irtree, slist)$;
  6:     **if** $c \neq \emptyset$ **then**
  7:         Compute cost of c;
  8:         $clusters.$**Add**(c);
  9:     **end if**
10: **end while**
11: **return** *clusters*;

The function **ProcessKstcQuery** (Algorithm F.4) takes an IR-tree, a k-STC query, and a relevant object set as input and returns the density based clusters. It iterates over a sorted list of elements with respect to the distance to the query location and gets the cluster with the object as core object (lines 4 and 5). If the cluster is not empty, its cost is computed, and it is added to the result list (lines 6–9).

Algorithm F.5 GetCluster($irtree, q, o, slist$)

Input: $irtree$ - IR-tree, q - k-STC query, o - The core object, $slist$ - Sorted list of relevant objects
Output: C - the cluster
1: $C \leftarrow \varnothing$;
2: $neighbors \leftarrow irtree.\textbf{RangeQuery}(o.\lambda, q.\psi, q.\epsilon)$;
3: **if** $neighbors.\textbf{size} < q.minpts$ **then**
4:      Remove o from $slist$;
5:      Mark o as noise;
6:      **return** C;
7: **else**
8:      Add $neighbors$ to C;
9:      Remove $neighbors$ from $slist$;
10:     Remove o from $neighbors$;
11:     **while** $neighbors \neq \varnothing$ **do**
12:        Object $o_i \leftarrow$ remove an element from $neighbors$;
13:        $neighbors_i \leftarrow irtree.\textbf{RangeQuery}(o_i.\lambda, q.\psi, q.\epsilon)$;
14:        **if** $neighbors_i.\textbf{size} \geq q.minpts$ **then**
15:           **for all** Object $o_j \in neighbors_i$ **do**
16:              **if** o_j is noise **then**
17:                Add o_j to C;
18:              **else if** $o_j \notin C$ **then**
19:                Add o_j to C;
20:                Remove o_j from $slist$;
21:                Add o_j to $neighbors$;
22:              **end if**
23:           **end for**
24:        **end if**
25:     **end while**
26: **end if**
27: **return** C;

To get the cluster of a given core object, the function **GetCluster** (Algorithm F.5) is utilized. It issues a range query centered at o with a range of $q.\epsilon$ on the IR-tree (line 2). The goal is to check whether o is a core object. If the re-

sult set *neighbors* has fewer than *q.minpts* objects, object *o* is marked as noise, and an empty set is returned (lines 3–6). Otherwise, the algorithm initiates a cluster *C* containing the objects in *neighbors*. Next, this cluster is expanded by checking the ϵ-neighborhood of each object o_i in *neighbors* except *o* (lines 12 and 13). If the ϵ-neighborhood of o_i is dense (line 14), the objects inside the neighborhood that are previously marked as noise are added to the cluster (lines 16 and 17). The objects that are not processed yet are also added to the cluster, removed from the sorted list, and added to *neighbors* (lines 18–22). If no more objects can be added, cluster *C* is returned as the result (line 27). It is important to note that this **GetCluster** is the same as in the regular DBSCAN algorithm. We provide the pseudocode here to facilitate understanding of the more advanced algorithms described later.

SGPL-based Improvements

The DBSCAN-based algorithm explained above uses the IR-tree for range queries and does not utilize the SGPL-based improvements. We proceed to explain how we can make use of the optimizations, namely selectivity estimation and the FastRange algorithm [11], in the DBSCAN-based algorithm. We also propose a FastKDist algorithm based on SGPL to compute the kth nearest neighbor distances of all objects.

Selectivity Estimation

Wu and Jensen [11] propose a selectivity estimation method based on SGPL to decrease the number of range queries issued to process queries. Given a set $q.\psi$ containing m query keywords, the corresponding m SGPLs are merged to estimate the selectivity of a range query. Wu and Jensen define a merging operator $\oplus$ on several SGPLs that produces a count for each non-empty grid cell as follows:

$$\bigoplus_{w_i \in q.\psi} (q_s) = \{ \langle c_j, | \bigcup_{w_i \in q.\psi} S_{w_i, c_j} | \rangle \mid C_{c_j} \cap q_s \neq \varnothing \} \tag{F.5}$$

The merge operator merges the SGPLs of query keywords and returns a set of pairs, each of which contains a cell id and the number of relevant objects in the cell. The definition approximates the circular query region defined by λ and ϵ by its circumscribed square (q_s) to check the intersection effectively, and C_{c_j} corresponds to the spatial region of the cell with id c_j. The idea is that if the number of relevant objects within the circumscribed square is less than *q.minpts*, there is no need to issue a range query. To incorporate this optimization, we just need to add a selectivity estimation check before issuing range queries in lines 2 and 13 of **GetCluster** as given

in Algorithm F.5. In other words, the algorithm should only issue a range query if the selectivity estimate exceeds *q.minpts*.

FastRange Algorithm

Wu and Jensen [11] propose an algorithm to process range queries using SGPL. To be able to process queries with several keywords, they override the merging operator ($\oplus$) with $\overline{\oplus}$ as follows:

$$\overline{\bigoplus_{w_i \in q.\psi}} (q_s) = \{\langle c_j, \bigcup_{w_i \in q.\psi} S_{w_i, c_j} \rangle \mid C_{c_j} \cap q_s \neq \varnothing\} \tag{F.6}$$

The overrided merge operator given in Equation F.6 produces a set of objects instead of a count for each grid cell to be able to process range queries effectively.

Algorithm F.6 FastRange($\lambda, \psi, \epsilon, sgplList$)

Input: λ - Query location, ψ - Query Keywords, ϵ - Query Range, *sgplList* - The list of SGPLs
Output: *neighbors* - The list of neighbors
1: *neighbors* $\leftarrow \varnothing$;
2: $q_s \leftarrow$ The circumscribed square around the circular query region defined by λ and ϵ;
3: *mergedSgpl* $\leftarrow \overline{\bigoplus_{w_i \in \psi}} (q_s)$;
4: **for all** Cell $c \in mergedSgpl$ **do**
5:     **if** c is completely inside the query region **then**
6:         Add all the objects inside c to the *neighbors*;
7:     **else**
8:         **for all** Object o inside c **do**
9:             **if** $\|\lambda o\| \leq q_c.\epsilon$ **then**
10:                Add o to *neighbors*;
11:             **end if**
12:         **end for**
13:     **end if**
14: **end for**
15: **return** *neighbors*;

The FastRange algorithm (Algorithm F.6) takes a location (λ), a query range (ϵ), a set of query keywords (ψ), and the list of SGPLs (*sgplList*) as arguments. It first applies the overrided merge operator to the given list of SGPLs for the circumscribed square around the query region and assigns the result to *mergedSgpl* (lines 2 and 3). If a cell c from *mergedSgpl* is completely inside the query region, all objects in c are added to the result (lines 3 and

4). If a cell intersects the query region, only objects in c that have distance to object location no greater than ϵ are added to the result (lines 9 and 10).

The FastRange algorithm can be utilized in the DBSCAN-based algorithm by just changing its calls to *irtree*.**RangeQuery** to calls to **FastRange** in lines 2 and 13 of the **GetCluster** function (Algorithm F.5).

FastKDist Algorithm

The DBSCAN-based algorithm needs to process the relevant object set to determine the appropriate ϵ values for the set. To do this, the **GetEpsValues** function given in Algorithm F.3 is employed. This requires a large amount of compute time because that the algorithm traverses the IR-tree for each relevant object. For this reason, a more efficient way to find the distance of kth nearest neighbor (k-dist) for all relevant objects is desirable. We utilize the approach proposed by Wu and Tan [30] together with SGPL index.

(a) Levels   (b) Groups
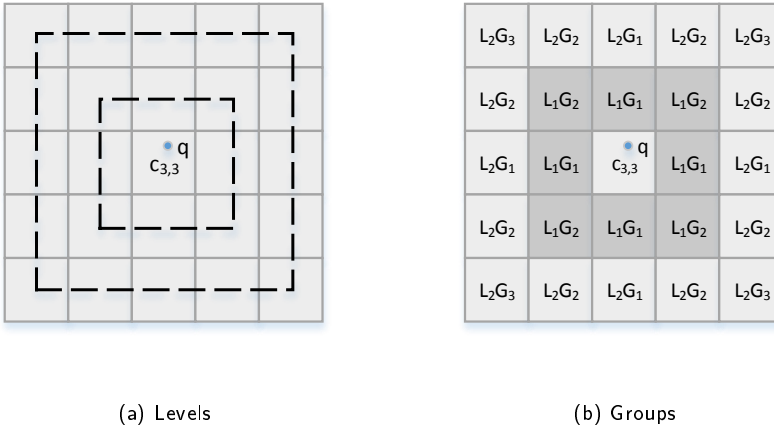
Fig. F.4: Levels and Groups for an Example kNN Query (redrawn from [30])

Wu and Tan [30] propose an algorithm on top of a grid-based index to process k-nearest neighbor (kNN) queries. They propose a method to build a visit order consisting of cells, levels, and groups in order to reduce the number of cells that should be visited to answer the kNN query. Assuming that $c_{a,b}$ is the cell that contains the query object, level l is the set of cells such that each cell $c_{i,j}$ satisfies either $i = a \pm l \wedge b - l \leq j \leq b + l$ or $j = b \pm l \wedge a - l \leq i \leq a + l$. The cells in group g ($1 \leq g \leq l + 1$) of level l is the set of cells $c_{i,j}$ that satisfy either $i = a \pm (g - 1) \wedge j = b \pm l$ or $i = a \pm l \wedge j = b \pm (g - 1)$. $L_l G_g$ denotes group g of level l. Figures F.4a and

F.4b show the corresponding levels and groups for an example kNN query (q) that is located in cell $c_{3,3}$.

FastKDist employs the same idea on top of SGPLs since we need a grid index that takes textual relevance to the given query into account. It also makes use of caching in order to reuse the distance computations between the objects.

The FastKDist algorithm (Algorithm F.7) takes a query q, the relevant object set (D_ψ) for q, k, an upper bound for ϵ (ϵ_{ub}), and the list of SGPLs as input and returns the k-dist values for all objects contained in the relevant object set. It first merges the SGPLs to form the SGPL corresponding to the query keywords (line 2). Then it computes the k-dist value for each object o in D_ψ (lines 3–28). The algorithm initializes the priority queue to build a visit order for the object and enqueues the cell the object is located in and the first group of the first level. Then it gets the next cell from the priority queue (function call **NextCell**) and checks the distance between the objects located in the cell and the current object. Function **GetDistanceBetween** employs a cache to reuse the distance computations. If the distance is already computed, it returns the value from the cache. Otherwise, the distance between the objects is computed and added to the cache. If the distance between the objects are less than the upper bound for k-dist value it is added to the list, and the upper bound is updated if necessary (lines 13–21). This loop continues until the priority queue is exhausted or the visited cell has a minimum distance to the object that exceeds the upper bound.

Function **NextCell** takes an object and a priority queue as input and returns the next cell to be visited. It dequeues an element from the queue and checks whether it is a cell or a group. If it is a cell, it is returned immediately. Otherwise, if the dequeued element is a group of the form $L_l G_g$, it first enqueues the cells of $L_l G_g$ to the queue with the distance between the cell and the object. Next, it enqueues $L_{l+1} G_g$ group with its minimum distance to the given object. Next, the function checks whether $l = g$ and if so, it enqueues $L_l G_{g+1}$. Finally, the function calls itself recursively.

The FastKDist algorithm can be utilized in the DBSCAN-based algorithm quite easily. We just need to remove lines 2–12 in function **GetEpsValues** (given in Algorithm F.3) and add a function call to **FastKDist** to get the k-dist values.

4.3 OPTICS-Based Algorithm

The main drawback of the DBSCAN-based algorithm presented in Section 4.2 is that given n different *minpts* and an average of m ϵ values for a k-TMSTC query, it needs to process $n \times m$ k-STC queries in parallel. By using the OPTICS algorithm [15], we can reduce the number of parallel queries to n since it is possible to extract clusters from the output cluster ordering for

---

Algorithm F.7 FastKDist$(D_\psi, k, \epsilon_{ub}, sgplList)$

Input: D_ψ - Relevant object set, k - The k value, ϵ_{ub} - The upper bound value
   for ϵ, *sgplList* - The list of SGPLs
Output: *kdistValues* - The list of k-dist values
 1: *kdistValues* ← new list;
 2: *mergedSgpl* ← $\displaystyle\overline{\bigoplus_{w_i \in q.\psi}}$ ();   ▷ The merge operator is applied to the whole

   grid.
 3: **for all** Object $o \in D_\psi$ **do**
 4:    *distList* ← new list;
 5:    $kdist_{ub} \leftarrow \epsilon_{ub}$;
 6:    *visitPQ* ← new priority queue;   ▷ Initialization of the priority queue
   to determine the cells to be visited for *o*.
 7:    *visitPQ*.**Enqueue**(**Cell**(o));
 8:    *visitPQ*.**Enqueue**(L_1G_1); ▷ Cells, groups and levels of *mergedSgpl* are
   added to the priority queue with their minimum distance to the $o.\lambda$.
 9:    $c \leftarrow$ **NextCell**($o, visitPQ$);
10:    **while** $c \neq$ **null** $\wedge kdist_{ub} >$ **Dist$_{min}$**(o, c) **do**
11:       **for all** Object $o_c \in$ cell c **do**
12:          $d \leftarrow$ **GetDistanceBetween**(o, o_c);
13:          **if** $d < kdist_{ub}$ **then**
14:             *distList*.**Add**(d);
15:             **if** *distList*.**size** $> k$ **then**
16:                Remove the largest value from *distList*;
17:                $kdist_{ub} \leftarrow k$-th largest value from *distList*;
18:             **else if** *distList*.**size** $= k$ **then**
19:                $kdist_{ub} \leftarrow k$-th largest value from *distList*;
20:             **end if**
21:          **end if**
22:       **end for**
23:       $c \leftarrow$ **NextCell**($o, visitPQ$);
24:    **end while**
25:    **if** *distList*.**size** $= k$ **then**
26:       *kdistValues*.**Add**(k-th largest value from *distList*)
27:    **end if**
28: **end for**
29: Sort *kdistValues*;
30: **return** *kdistValues*;

---

the ϵ values lower than the upper bound ϵ_{ub}. The second drawback of the
DBSCAN-based algorithm is that the DBSCAN parameters must be deter-
mined before processing k-STC queries. By using OPTICS, we can determine

the set of parameters while constructing the cluster order since the parameters are not needed until we extract the clusters from the cluster order. So, we do not need to have a separate parameter determination phase.

In the following, we first briefly introduce preliminaries of the OPTICS algorithm and then present the OPTICS-based algorithm for the processing of *k*-TMSTC queries.

Preliminaries

The OPTICS algorithm [15] is proposed by Ankerst et al. for density based clustering as an extension to the DBSCAN algorithm. The algorithm requires *minpts* and generating ϵ (ϵ_g) parameters just like DBSCAN. However, instead of a single list of density-based clusters, it constructs an ordering of data points, the so-called density-based cluster order, which then can be used to extract clusters for different values of ϵ_i that are smaller than the generating distance ϵ_g (i.e., $\epsilon_i < \epsilon_g$). To define the cluster order, we first need to define the relevant concepts.

Definition F.10. *The **core distance** of an object o is defined as the minimum ϵ value such that o is a core object. It is defined only if o is a core object with respect to a generating distance parameter (ϵ_g), and it is equal to the distance between o and its kth nearest neighbor with k = minpts.*

Definition F.11. *The **reachability distance** of an object p with respect to object o is defined as the maximum ϵ value that makes p reachable from o. It is undefined if o is not a core object with respect to ϵ_g. If o is a core object then it is the maximum of the core-distance of o and the distance between o and p. Reachability here means direct reachability as defined in Definition F.4.*

Definition F.12. *A **cluster order** is an ordered list of objects where each object has two additional attributes: the core distance and the reachability distance from one of the core objects. It contains all the objects from the database in the order of their processing by the algorithm.*

The OPTICS algorithm proposed by Ankerst et al. [15] is given in Algorithm F.8. It creates a cluster order from the set of relevant objects according to a generating ϵ value (ϵ_g) and a *minpts* parameter. The OPTICS algorithm first initializes the cluster order as an empty list (line 1). It then iterates over the set of relevant objects. If an object is not already processed, the algorithm expands the cluster order starting from that object (lines 3–24).

To expand the cluster order from a given object *o*, the algorithm first creates a priority queue of seeds to store candidates for the cluster order (line 4). The priority queue uses the reachability distance with respect to the closest core object as the priority. The algorithm then issues a range query to get

---

Algorithm F.8 OPTICS$(irtree, q, D_\psi, \epsilon_g, minpts)$

Input: *irtree* - IR-tree, *q* - *k*-TMSTC query, D_ψ - Relevant object set, ϵ_g - The generating ϵ parameter, *minpts* - The *minpts* value

Output: *clusterOrder* - the output cluster order

 1: *clusterOrder* ← new list
 2: **for all** $o \in D_\psi$ **do**
 3:    **if** !o.**processed then**
 4:       *seeds* ← new priority queue;
 5:       *neighbors* ← *irtree*.**RangeQuery**$(o.\lambda, q.\psi, \epsilon_g)$;
 6:       o.**processed** ← TRUE;
 7:       o.**reachability_distance** ← UNDEFINED;
 8:       *clusterOrder*.**Add**(o);
 9:       **if** *neighbors*.**size** ≥ *minpts* **then**
10:          o.**core_distance** ← *minpts*-dist to o from *neighbors*;
11:          **UpdateOrderSeeds**$(seeds, o, neighbors)$;
12:          **while** *seeds* is not empty **do**
13:             o_i ← dequeue an element from *seeds*;
14:             *neighbors$_i$* ← *irtree*.**RangeQuery**$(o_i.\lambda, q.\psi, \epsilon_g)$;
15:             o_i.**processed** ← TRUE;
16:             *clusterOrder*.**Add**(o_i);
17:             **if** *neighbors$_i$*.**size** ≥ *minpts* **then**
18:                o_i.**core_distance** ← *minpts*-dist to o_i from *neighbors$_i$*;
19:                **UpdateOrderSeeds**$(seeds, o_i, neighbors_i)$;
20:             **end if**
21:          **end while**
22:       **end if**
23:    **end if**
24: **end for**
25: **return** *clusterOrder*;

---

the ϵ_g-neighborhood of o (line 5). Then, o is added to the cluster order (line 8). If the ϵ_g-neigborhood is dense, the object's core distance is determined (line 10). The algorithm then updates the seeds queue with the objects in the ϵ_g neighborhood of o (line 11). The **UpdateOrderSeeds** function call adds the neighbor objects to the seeds queue with their new reachability distances from o. If a neighbor object is already in the queue, its reachability distance is updated if the reachability distance from o is less than its current reachability distance. The same procedure is applied for each object o_i dequeued from the seeds queue (lines 12–21). When the order seeds queue is empty, the algorithm returns back to the main loop and starts the same expansion from one of the remaining unprocessed objects, if there are any.

Algorithm

We proceed to propose the OPTICS-based algorithm that utilizes the IR-tree.

Algorithm F.9 OPTICS-based Algorithm

Input: *irtree* - IR-tree, *q* - *k*-TMSTC query, *minmp* - Minimum *minpts* value,
   maxmp - Maximum *minpts* value
Output: *clusters* - The list of clusters
 1: $\Delta_{ub}, \epsilon_{ub} \leftarrow$ **GetBounds**($q.tm_c, q.tm_i$);
 2: $D_\psi \leftarrow irtree.$**RangeQuery**($q, \Delta_{ub}$);
 3: $cSet \leftarrow \varnothing$;
 4: **for** *minpts* = *minmp* **to** *maxmp* **do**
 5:    $cm \leftarrow$ **ProcessQueryForMinPts**($irtree, q, D_\psi, minpts, \epsilon_{ub}$);   ▷ Gets
   executed in parallel using fork-join model
 6:    $cSet \leftarrow cSet \cup cm$;
 7: **end for**
 8: *clusters* $\leftarrow$ top-*k* disjoint clusters of *cSet* sorted with respect to the cost;
 9: **return** *clusters*;

The OPTICS-based algorithm is given in Algorithm F.9. It first determines the upper bound values Δ_{ub} and ϵ_{ub} and initializes the set of relevant objects (lines 1 and 2). The algorithm then initializes the set of clusters *cSet* to an empty set and processes the query for different *minpts* values in the range of [*minmp–maxmp*] in parallel (lines 3–7). It uses a simple fork-join model, and at the end of each parallel execution, expands *cSet* by taking its union with the result from the parallel execution (line 6). Finally, it returns the top-*k* disjoint clusters from *cSet* with respect to the cost.

The function **ProcessQueryForMinPts** (Algorithm F.10) processes the *k*-TMSTC query for a single *minpts* parameter. It first determines the possible values for ϵ and populates the cluster order from the relevant object set using the OPTICS algorithm [15] with parameters ϵ_{ub} and *minpts* (line 3). Normally, the OPTICS algorithm as given in Algorithm F.8 just creates a cluster order. We update it to return the set of ϵ values while populating the cluster order. The algorithm then iterates over the *epsValues* and constructs the clusters for each ϵ value (lines 3–16). To construct the clusters for a single ϵ value, it iterates over the cluster order and checks the reachability distance of the object (line 6). If it is reachable from the current cluster with respect to the ϵ value then it is just added to the current cluster (line 7). Otherwise, the algorithm checks whether the object is a core object (line 8). If so, the algorithm first adds the current cluster to the set of clusters, if it is populated, and initiates a new cluster with core object *o* (lines 9–13). The algorithm terminates when the clusters for all possible ϵ values are extracted.

Function **OPTICS** is updated to return a list of ϵ values together with

---

Algorithm F.10 ProcessQueryForMinPts($irtree, q, D_\psi, minpts, \epsilon_{ub}$)

Input: *irtree* - IR-tree, *q* - *k*-TMSTC query, D_ψ - Relevant object set, *minpts* - The *minpts* value, ϵ_{ub} - The upper bound value for ϵ

Output: *clusters* - the result set of clusters

1: *clusters* $\leftarrow \emptyset$;
2: *clusterOrder, epsValues* $\leftarrow$ **OPTICS**($irtree, q, D_\psi, \epsilon_{ub}, minpts$);
3: **for all** ϵ in *epsValues* **do**
4:     $C \leftarrow \emptyset$ $\triangleright$ Current cluster
5:     **for all** Object *o* in *clusterOrder* **do**
6:         **if** *o.reachability_distance* $\leq \epsilon$ **then** $\triangleright$ The object is reachable from the current cluster.
7:             Add *o* to *C*;
8:         **else if** *o.core_distance* $\leq \epsilon$ **then** $\triangleright$ The object is a core object with respect to ϵ, so we can initiate a new cluster.
9:             **if** *C* is not empty **then**
10:                 Add *C* to *clusters*;
11:                 $C \leftarrow \emptyset$;
12:             **end if**
13:             Add *o* to *C*;
14:         **end if**
15:     **end for**
16: **end for**
17: **return** *clusters*;

---

the cluster order. We decided to integrate determining the ϵ parameters into the OPTICS algorithm since we do not need the ϵ values until we construct the clusters. As shown in Algorithm F.8, the OPTICS algorithm issues range queries to get the ϵ_g neighborhood of each object to expand the cluster order. The updated version issues range queries together with *k*-dist queries for each object and process these two types of queries in an integrated manner. For this reason, we use the function **RangeQueryWithKDist** that returns the *k*-dist value together with the ϵ_g neighborhood instead of using the function **RangeQuery** in Algorithm F.8. These two functions are quite similar, except that the former takes an extra argument $k = minpts$ and uses a priority queue where the priority value is the node's distance to the given object location. It keeps track of the number of objects added to the neighbors and sets *k*-dist to the distance of the kth object added to the neighbors. It then returns the *k*-dist value together with the list of neighbors. When all relevant objects are processed and the list of *k*-dist values is populated, the updated OPTICS algorithm determines the list of ϵ values from the *k*-dist values using the procedure given in Algorithm F.3 (lines 12–27).

SGPL-based Improvements

The OPTICS-based algorithm explained above utilizes the IR-tree to process range queries and k-dist queries in an integrated manner. In this section, we explain how we can utilize SGPL instead of the IR-tree for processing a range query together with a k-dist query.

   The idea is quite similar to the FastKDist algorithm given in Algorithm F.7. The function **FastRangeWithKDist** is defined by the main loop in the Algorithm F.7 (lines 4–27), and it takes an object as an extra parameter. Since we need to add all objects whose distance to the given object does not exceed ϵ_{ub}, the loop condition in line 10 is updated to $c \neq null \wedge \epsilon_{ub} \geq \textbf{Dist}_{\textbf{min}}(o, c)$. We add an outer if statement before line 13 since we need to check whether the distance between the object and the input object does not exceed ϵ_{ub} for each object in the cell. If that is the case, the object is added to the list of neighbors. This function returns the k-dist value and the list of neighbors for the given object.

   It is straightforward to utilize the function **FastRangeWithKDist** in the OPTICS-based algorithm. We just need to call **FastRangeWithKDist** instead of **RangeQueryWithKDist**.

5 Experimental Evaluation

We proceed to evaluate the proposed algorithms. Section 5.1 presents the experimental setup. The proposals for k-TMSTC queries are evaluated in Section 5.2. We first study the effect of the algorithm parameters on the proposals. Then, we study the effect of the query parameters on the proposals.

5.1 Experimental Setup

We use a real dataset from Yelp that contains $156,639$ PoIs. The textual description of a PoI contains the name of the PoI, the categories that the PoI belong to, and additional information such as parking and ambience. The total number of distinct words in the dataset exceeds $40,000$. The set of query keywords is formed by choosing 10 random keywords from the top-50 categories in the dataset. The set of query locations consists of random locations in the center region of the 10 cities with the highest PoI density. We form a k-TMSTC query with $k = 6$ for each location and for each keyword, so the set of k-TMSTC queries consists of 100 queries. Note that the value of k does not affect the performance of the algorithms since they form all density based clusters and compute their costs before returning the top-k clusters.

   We evaluate the performance of the DBSCAN-based algorithm on the IR-tree (v1), the DBSCAN-based algorithm on the SGPL (v2), the OPTICS-based algorithm on the IR-tree (v3), and the OPTICS-based algorithm on the SGPL

Table F.2: Parameters

Notation	Explanation	Values
inc_{th}	Increase threshold	**0.05**, 0.06, 0.07, 0.08, 0.09, 0.1
$maxmp$	Maximum *minpts* value	8, 9, **10**, 11, 12, 13, 14, 15
h	Order of z-curve in SGPL	14, **15**, 16, 17, 18, 19, 20
$|q.\psi|$	Number of keywords	**1**, 2, 3, 4
ϵ_{ub}	Upper bound for ϵ (km)	0.25, 0.5, 0.75, **1.0**, 1.25, 1.5, 1.75, 2.0
Δ_{ub}	Upper bound for distance (km)	10, 20, 30, 40, **50**, 60

(v4) under different algorithm and query parameter settings. Table F.2 lists the parameter values used in the experiments, where bold values are the default values. We normally assume that we have access to a rule-based function to determine the upper bounds for ϵ and distance (ϵ_{ub}, Δ_{ub}) according to transportation modes. However, we manually set these upper bounds for the experiments to observe their effect on the query processing performance. We change the value of one parameter while fixing the other parameters to their default values in each experiment. Then we process the queries and report the average elapsed time cost. All algorithms are implemented in Java, and an Intel(R) Core(TM) i7-4720HQ CPU @ 2.60GHz with 16GB main memory is used for the experiments. The IR-tree and SGPL index structures are disk-resident.

5.2 Performance Evaluation

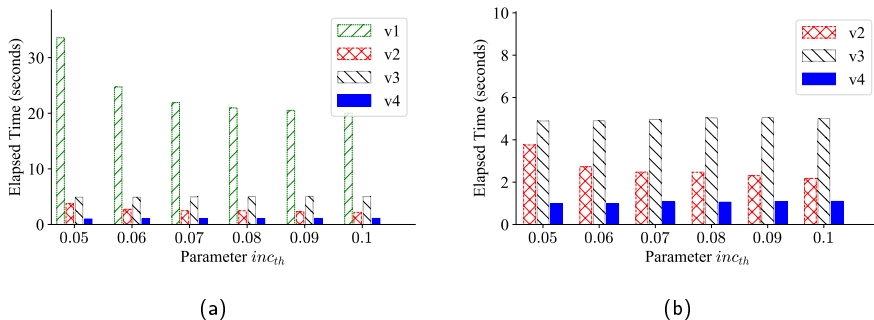We evaluate our proposals while varying the parameters listed in Table F.2.

(a) (b)

Fig. F.5: Effect of Parameter inc_{th}

Increase Threshold (*inc$_{th}$***).** Figure F.5a shows the performance of the algorithms when varying the increase threshold parameter. Figure F.5a shows that the DBSCAN-based algorithm on the IR-tree (v1) has substantially lower performance than the competitors. For this reason and to show more detail in the figures, we only report the average elapsed time for the other algorithms as shown in Figure F.5b. The DBSCAN-based algorithms (v1 and v2) improve as *inc$_{th}$* increases. This is expected since the number of *k*-STC queries processed decreases as *inc$_{th}$* increases. As expected, the OPTICS-based algorithms (v3 and v4) are insensitive to variations in the increase threshold. The OPTICS-based algorithms first create the cluster order for each *minpts* value and determine the ϵ values for density-based clustering according to the *inc$_{th}$* parameter. Then the clusters are extracted from the cluster order with respect to the ϵ values. Although the number of ϵ values decreases as *inc$_{th}$* increases, this does not affect the elapsed time for the OPTICS-based algorithms. This suggests that we can utilize smaller *inc$_{th}$* values for the OPTICS-based algorithms without a performance loss.
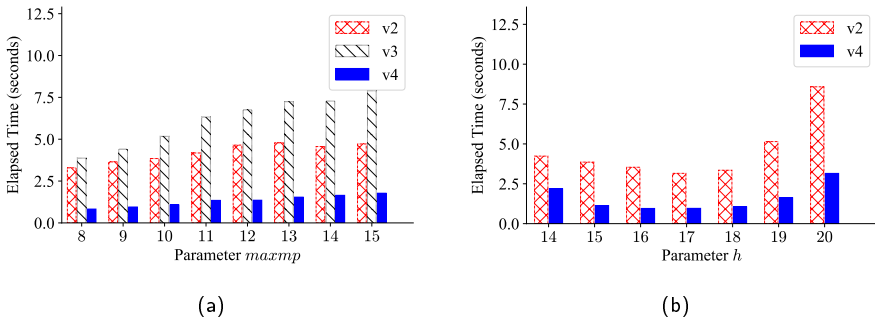

(a)   (b)

Fig. F.6: Effects of Parameters *maxmp* and *h*

Maximum *minpts* **Value (***maxmp***).** In order to observe the effect of the range of *minpts* values on the proposed algorithms, we set the minimum *minpts* value to 3 and vary the maximum *minpts* value. Figure F.6a shows the elapsed time of the proposed algorithms when varying *maxmp*. The performance of all algorithms become worse as *maxmp* increases, as expected. A larger *maxmp* means a larger number of *k*-STC queries to be processed for the DBSCAN-based algorithms. For the OPTICS-based algorithms, the number of parallel queries is directly related to the range of *minpts* values as can be seen in line 4 of Algorithm F.9.

Order of Z-Curve in SGPL (*h***).** SGPL is constructed based on the grid over the data set. The grid is indexed by a space filling curve. We adopt the Z-curve in our evaluation. Other types of space filling curves can also be utilized. The order *h* of the Z-curve defines the granularity of the grid

that is expected to affect the performance of SGPL. A large h provides a finer granularity for the grid. Since the parameter h is related to SGPL, it only affects the algorithms that use the SGPL (v2 and v4). The average elapsed time for these algorithms is shown in Figure F.6b. The time decreases as h increases up to a certain point. Then the elapsed time increases again. This shows that a finer granularity for the grid might result in having too many distance computations between the objects in FastRange, FastKDist, and FastRangeWithKDist algorithms.
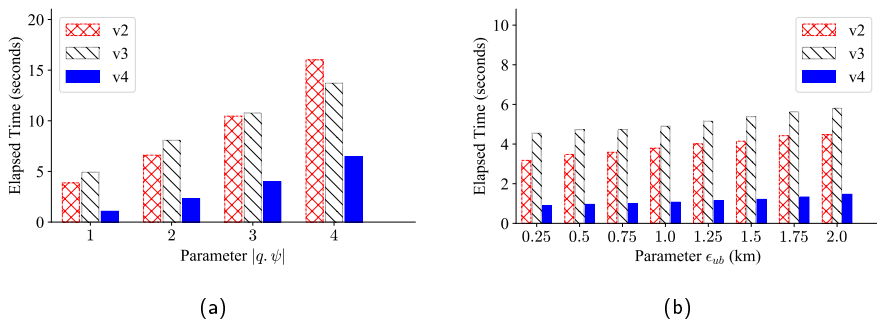

(a)   (b)

Fig. F.7: Effects of Parameters $|q.\psi|$ and ϵ_{ub}

Number of Keywords ($|q.\psi|$). Figure F.7a shows the performance of the proposed algorithms when varying the number of query keywords. The performance decreases as the number of query keywords increases. This is expected since the number of relevant objects increases when more keywords are part of the query. This results in a larger search space; thus, the number of range queries issued increases as well.

Upper Bound for ϵ (ϵ_{ub}). Figure F.7b shows the effect of ϵ_{ub} on the performance of the algorithms. The performance of the DBSCAN-based algorithms becomes worse as ϵ_{ub} increases since a larger ϵ_{ub} value results in more k-STC queries. The performance of the OPTICS-based algorithms also becomes worse as ϵ_{ub} increases since these algorithms issue range queries with the radius of ϵ_{ub}; thus, a larger ϵ_{ub} value means a larger search space for the range queries.

Upper Bound for Distance (Δ_{ub}). Figure F.8 shows the effect of Δ_{ub} on the performance of the algorithms. This parameter directly affects the number of relevant objects, which increases as Δ_{ub} increases. However, this increase only affects the DBSCAN-based algorithm on the IR-tree (v1). For this reason, the elapsed time for v1 is shown in this figure. The performance of the other approaches is not affected substantially since the increase in the number of relevant objects is not high. Since all the query locations are in city centers, most of the PoIs are already within 10 kms of the query locations.
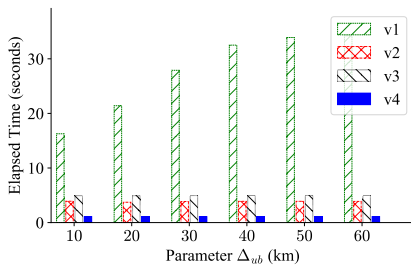
Fig. F.8: Effect of Parameter Δ_{ub}

So, increasing this upper bound to 60 kms does not add a lot of extra PoIs to the relevant PoI set.

Summary. Overall, the experimental evaluation shows that the OPTICS-based algorithm on SGPL (v4) outperforms the other algorithms. It provides a speedup up to 30x compared to v1, up to $4x$ compared to v2, and up to 5x compared to v3. The experiments also suggest that we are able to process k-TMSTC queries with a response time that supports interactive search. The effects of the FastKDist and FastRange algorithms on SGPL can be seen from the difference between the elapsed times of the DBSCAN-based algorithm on the IR-tree (v1) and the DBSCAN-based algorithm on the SGPL (v2).

6 Conclusion

This paper proposes and studies the top-k transportation-mode aware spatial textual cluster (k-TMSTC) query that returns k density-based clusters of points of interest that are relevant to query keywords. We propose a spatial and a spatio-textual cost function to compute the costs of clusters. The top-k clusters with the lowest costs form the output. We propose an algorithm based on DBSCAN and an algorithm based on OPTICS to process k-TMSTC queries. We also propose an algorithm called FastKDist to compute the kth nearest neighbor distances for all objects in a given set. The DBSCAN-based algorithm utilizes FastKDist to determine the density-based clustering parameters. An experimental evaluation using real point of interest data suggests that the proposed algorithms are capable of processing queries in interactive time. The evaluation also indicates that FastKDist improves the performance of the DBSCAN-based algorithm.

The proposed algorithms assume that upper bounds for the distance and density-based clustering parameters are available. As future work, it is of interest to propose a method that determines the upper bounds with respect to the relevant PoIs on the fly. Furthermore, it is of interest to build an index

structure that considers the cost function and is capable of providing an upper bound that enables effective search space pruning. Such an index structure might improve the query processing time of the proposed algorithms since the need to form all candidate clusters before forming top-*k* clusters may be removed.

References

[1] (2015, Jun.) Google annual search statistics. [Online]. Available: http://www.statisticbrain.com/google-searches/

[2] I. D. Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *Proceedings of the 24th International Conference on Data Engineering (ICDE 2008)*, 2008, pp. 656–665.

[3] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 337–348, Aug. 2009.

[4] A. Cary, O. Wolfson, and N. Rishe, "Efficient and scalable method for processing top-k spatial Boolean queries," in *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM 2010)*.   Springer, 2010, pp. 87–95.

[5] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg, "Efficient processing of top-k spatial keyword queries," in *Proceedings of the International Symposium on Spatial and Temporal Databases (SSTD 2011)*. Springer, 2011, pp. 205–222.

[6] Z. Li, K. C. K. Lee, B. Zheng, W. C. Lee, D. Lee, and X. Wang, "IR-tree: An efficient index for geographic document search," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 4, pp. 585–599, April 2011.

[7] J. a. B. Rocha-Junior and K. Nørvåg, "Top-k spatial keyword queries on road networks," in *Proceedings of the 15th International Conference on Extending Database Technology (EDBT '12)*.   ACM, 2012, pp. 168–179.

[8] D. Wu, G. Cong, and C. S. Jensen, "A framework for efficient spatial web object retrieval," *The VLDB Journal*, vol. 21, no. 6, pp. 797–822, 2012.

[9] D. Wu, M. L. Yiu, G. Cong, and C. S. Jensen, "Joint top-k spatial keyword query processing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 10, pp. 1889–1903, 2012.

[10] D. Zhang, K.-L. Tan, and A. K. H. Tung, "Scalable top-k spatial keyword search," in *Proceedings of the 16th International Conference on Extending Database Technology (EDBT '13)*. ACM, 2013, pp. 359–370.

[11] D. Wu and C. S. Jensen, "A density-based approach to the retrieval of top-k spatial textual clusters," in *Proceedings of the 25th ACM International Conference on Information and Knowledge Management (CIKM '16)*. ACM, 2016, pp. 2095–2100.

[12] A. Skovsgaard and C. S. Jensen, "Finding top-k relevant groups of spatial web objects," *The VLDB Journal*, vol. 24, no. 4, pp. 537–555, Aug. 2015.

[13] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*. AAAI Press, 1996, pp. 226–231.

[14] P. Liu, D. Zhou, and N. Wu, "Vdbscan: Varied density based spatial clustering of applications with noise," in *Proceedings of the International Conference on Service Systems and Service Management (ICSSSM 2007)*, 2007, pp. 1–4.

[15] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," in *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD '99)*. ACM, 1999, pp. 49–60.

[16] X. Cao, L. Chen, G. Cong, C. S. Jensen, Q. Qu, A. Skovsgaard, D. Wu, and M. L. Yiu, "Spatial keyword querying," in *Proceedings of the 31st International Conference on Conceptual Modeling (ER 2012)*. Springer, 2012, pp. 16–29.

[17] X. Cao, G. Cong, and C. S. Jensen, "Retrieving top-k prestige-based relevant spatial web objects," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 373–384, Sep. 2010.

[18] G. Li, J. Feng, and J. Xu, "Desks: Direction-aware spatial keyword search," in *Proceedings of the 28th International Conference on Data Engineering (ICDE 2012)*, 2012, pp. 474–485.

[19] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong, "Efficient continuously moving top-k spatial keyword query processing," in *Proceedings of the 27th International Conference on Data Engineering (ICDE 2011)*, 2011, pp. 541–552.

References

[20] D. Wu, M. L. Yiu, and C. S. Jensen, "Moving spatial keyword queries: Formulation, methods, and analysis," *ACM Trans. Database Syst.*, vol. 38, no. 1, pp. 7:1–7:47, 2013.

[21] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi, "Collective spatial keyword querying," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD '11).* ACM, 2011, pp. 373–384.

[22] C. Long, R. C.-W. Wong, K. Wang, and A. W.-C. Fu, "Collective spatial keyword queries: A distance owner-driven approach," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13).* ACM, 2013, pp. 689–700.

[23] X. Cao, G. Cong, T. Guo, C. S. Jensen, and B. C. Ooi, "Efficient processing of spatial group keyword queries," *ACM Trans. Database Syst.*, vol. 40, no. 2, pp. 13:1–13:48, 2015.

[24] H. K.-H. Chan, C. Long, and R. C.-W. Wong, "Inherent-cost aware collective spatial keyword queries," in *Proceedings of the International Symposium on Spatial and Temporal Databases (SSTD 2017)*, 2017, pp. 357–375.

[25] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data (SIGMOD '84).* ACM, 1984, pp. 47–57.

[26] K. S. Bøgh, A. Skovsgaard, and C. S. Jensen, "Groupfinder: A new approach to top-k point-of-interest group retrieval," *Proc. VLDB Endow.*, vol. 6, no. 12, pp. 1226–1229, Aug. 2013.

[27] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Commun. ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[28] J. M. Ponte and W. B. Croft, "A language modeling approach to information retrieval," in *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '98).* ACM, 1998, pp. 275–281.

[29] J. Zobel and A. Moffat, "Inverted files for text search engines," *ACM Comput. Surv.*, vol. 38, no. 2, Jul. 2006.

[30] W. Wu and K. L. Tan, "isee: Efficient continuous k-nearest-neighbor monitoring over moving objects," in *Proceedings of the 19th International Conference on Scientific and Statistical Database Management (SSDBM 2007)*, 2007, pp. 36–36.