The SIGSPATIAL Special

Newsletter of the Association for Computing Machinery
Special Interest Group on Spatial Information

Volume 11 Number 1 March 2019

The SIGSPATIAL Special

The SIGSPATIAL Special is the newsletter of the Association for Computing Machinery (ACM) Special Interest Group on Spatial Information (SIGSPATIAL).

ACM SIGSPATIAL addresses issues related to the acquisition, management, and processing of spatially-related information with a focus on algorithmic, geometric, and visual considerations. The scope includes, but is not limited to, geographic information systems.

For more details and membership information for ACM SIGSPATIAL as well as for accessing the newsletters please visit http://www.sigspatial.org.

The SIGSPATIAL Special serves the community by publishing short contributions such as SIGSPATIAL conferences' highlights, calls and announcements for conferences and journals that are of interest to the community, as well as short technical notes on current topics. The newsletter has three issues every year, i.e., March, July, and November. For more detailed information regarding the newsletter or suggestions please contact the editor via email at azufle@gmu.edu.

The SIGSPATIAL Special (ISSN 1946-7729) Volume 11, Number 1, March 2019.

Table of Contents

Message from the Editor

Andreas Züfle
Department of Geography and GeoInformation Science, George Mason University, USA
Email: azufle@gmu.edu

Dear SIGSPATIAL Community,

The newsletter serves the community by publishing short contributions such as SIGSPATIAL conferences' highlights, calls and announcements for conferences and journals that are of interest to the community, as well as short technical notes on current topics.

The first section of this March 2019 issue features three technical reports highlighting potential new future research directions. These directions include Spatial Pattern Matching, new directions for Spatal Joins, and applications for using Traffic Simulation for research.

The second section consists of SIGSPATIAL 2018 event reports, including the ACM SIGSPATIAL 2018 main conference report and a report of the 2018 SIGSPATIAL CUP. Last but not least, a new type of workshop called "Spatial Gems" for archiving and disseminating fundamental techniques for solving spatial problems is presented herein.

You can download all Special issues from:

`http://www.sigspatial.org/sigspatial-special`

I want to sincerely thank all authors of for their generous contributions of time and effort that made this issue possible. I hope that you will find the newsletters interesting and informative and that you will enjoy this issue.

Yours sincerely,
Andreas Züfle
SIGSPATIAL Newsletter Editor

The SIGSPATIAL Special

Section 1: Special Issue on Spatial Query Processing and Traffic Simulation

ACM SIGSPATIAL

http://www.sigspatial.org

Introduction to this Special Issue: Spatial Query Processing and Traffic Simulation

Andreas Züfle
Department of Geography and GeoInformation Science, George Mason University
Email: azufle@gmu.edu

The goal of this special issue is to disseminate new research directions, challenges, and visions of broad interest to the SIGSPATIAL community. Two topics are covered in this special issue. The first topic of Spatial Query Processing features two articles describing novel and useful spatial query types and surveying new directions for existing queries. The second topic of of Traffic Simulation spotlights one article that puts recent advances in traffic simulation in the context of data generation for the broad community. More specifically,

1. in the first article, Li et al. propose a new research direction of matching spatial patterns. The proposed query allows return spatial objects that satisfy a patterns such as "a house within 10-minute walk from a school, which is at least 2km away from a hospital". In addition to defining the Spatial Pattern Matching query, this article describes applications and a plethora of future research directions inspired from this work;

2. in the second article, Bouros and Mamoulis describe future directions for Spatial Joins. This article surveys the state-of-the-art of spatial join evaluation and identifies new research directions enabled by modern hardware and parallel processing. I expect that this article will revitalize research and applications on spatial join processing;

3. in the third article, we hit the road: Xie et al. describe applications of their recently proposed Scalable Microscopic Adaptive Road Traffic Simulator (SMARTS) used for traffic data generation. In addition to an overview of SMARTS, this article describe applications in routing algorithm evaluation, vehicle prioritization, simuation-based navigation and traffic optimization. Traffic simulation is of particular interest for the ACM SIGSPATIAL CUP 2019, for which the problem is defined in the context of simulating crowdsourced taxicabs searching for customers to pick up;

I hope the readers will enjoy this issue and find it useful in their research work. I'd also like to call upon readers to send me suggestions for news that they would like to appear in the next issues of this newsletter. If you have exciting news that you would benefit the SIGSPATIAL community and that you would like to disseminate, let me know! Finally, I want to cordially thank the authors for their excellent contributions to this issue.

Spatial Pattern Matching: A New Direction for Finding Spatial Objects

Yun Li[1], Yixiang Fang[2], Reynold Cheng[3], Wenjie Zhang[2]
[1]Department of Computer Science and Technology, Nanjing University, China
[2]School of Computer Science and Engineering, The University of New South Wales, Australia
[3]Department of Computer Science, University of Hong Kong, Hong Kong
Email: liycser@gmail.com, yixiang.fang@unsw.edu.au,
ckcheng@cs.hku.hk, wenjie.zhang@unsw.edu.au

Abstract

In this paper, we study the spatial pattern matching *(SPM) query. Given a set D of spatial objects (e.g., houses and shops), each with a textual description, we aim at finding all combinations of objects from D that match a user-defined spatial pattern P. A pattern P is a graph whose vertices represent spatial objects, and edges denote distance relationships between them. The SPM query returns the instances that satisfy P. An example of P can be "a house within 10-minute walk from a school, which is at least $2km$ away from a hospital". The SPM query can benefit users such as house buyers, urban planners, and archaeologists. We first formally formulate the SPM problem, and then propose efficient query algorithms. We also develop an online system, called SpaceKey, which is based on the SPM query, to support some real applications such as property searching. Finally, we point out a list of possible research directions for future work.*

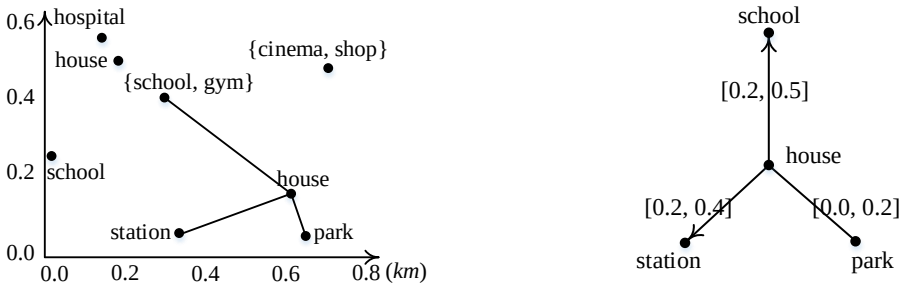
1 Introduction

With the rapid development of location-based services, spatial databases, which contain objects carrying locations and other information [2, 8, 7, 11], are prevalent in emerging platforms (e.g., Google Maps[1] and Foursquare[2]). In this paper, we study *spatial pattern matching*, or SPM in short. In this problem, we wish to find out the instances in a spatial database D that satisfy a given spatial pattern P. Figure 1(a) illustrates D, which contains a variety of spatial objects (e.g., park, station, and house). An example of P, which specifies keywords of objects and their distance relationships (e.g., a house is less than $0.2km$ away from a park; the distance between a bus station and a house is between $0.2km$ and $0.4km$), is shown in Figure 1(b). An instance that satisfies P (called a match), containing objects connected in solid lines, is shown in Figure 1(a).

The SPM can be useful for accommodation-search applications (e.g., AirBNB[3], and trivago[4]), where users can look for apartments or hotels based on their preferences. By using SPM, a user can indicate more complex

Yixiang Fang is the corresponding author.

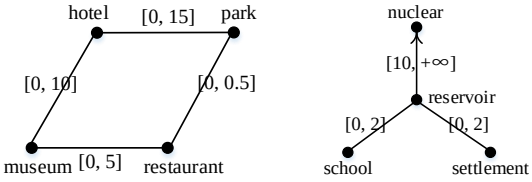
[1]https://maps.google.com.hk
[2]https://foursquare.com
[3]http://www.airbnb.com.hk
[4]http://www.trivago.hk



(a) Spatial object set D (b) Spatial pattern P

Figure 1: Illustrating the SPM query [6].

constraints that are currently not supported in these platforms. For example, a user may want to look for a house which is not too far from a shop, and the shop is close to a canteen. She may also want the house to be at least some distance from a subway station (to avoid noise and crowd). These requirements, which have not been provided by existing applications, can be captured succinctly in terms of a *spatial pattern* (e.g., Figure 1(b)). As another example, a travel agent may wish to enumerate all the possible itineraries, and recommend them to customers [2]. This kind of requests can be formulated as a spatial pattern (e.g., Figure 2(a)), with their spatial relationship indicated (e.g., a museum should be 10 minutes of driving from the restaurant). The SPM can also be used in urban planning. As discussed in [13], urban analysts often need to survey the layout of a city. They need to identify facilities with specific distance relationships (e.g., find out reservoirs that are $10km$ or more from nuclear power stations), in order to gain some insights on the the design of city infrastructures (Figure 2(b)).



(a) Trip planning (b) Urban planning

Figure 2: Example spatial patterns.

To our best knowledge, there does not exist any solution that solves SPM effectively and efficiently. A class of recent works related to SPM query is spatial-keyword query (SKQ) [2, 4, 19, 14, 1, 5, 3]. Typically, these solutions require users to provide a number of query keywords, and return spatial objects that are geographically near to each other; the objects should also be relevant to the keywords. However, this may fail to capture the user's intention, because they do not allow users to specify explicit distance constraints between objects. Another topic related to SPM is graph pattern matching (or GPM) [12], which aims to find subgraphs matching a given pattern in a graph. However, using GPM solutions to solve the SPM problem is not straightforward. Particularly, we have to transform the set of spatial objects involved (e.g., Figure 1(a)) into a graph, and then run a GPM solution on it.

Our contributions. We first present a formal definition of spatial pattern [6, 10]. We propose several distance constraints for a spatial pattern, which specify (1) minimum and maximum distances between two object types; and (2) *exclusion* and *inclusion*. Figure 1(b) illustrates the exclusion relationship ($\rightarrow$), which expresses that (1) a bus station should be at least $0.2km$ from a house but not more than $0.4km$; and (2) no station should be in the vicinity of $0.2km$ of a house. Based on this spatial pattern, we define the SPM problem,

and we prove that it is NP-hard. To tackle this issue, we propose two efficient and effective algorithms based on the IR-tree index [4, 16]. The first solution, which we called the multi-pair-join (or MPJ), is based on performing multi-way join operations [17, 20, 18] on each edge of the spatial pattern. We also develop a sampling-based estimation method to guide the execution order of the joins. Our second solution, namely the multi-star-join (or MSJ), also follows the multi-way join framework, but includes more sophisticated techniques, such as bounded pattern and optimized join order.

In addition, we have implemented our solution in a system, called SpaceKey [9]. SpaceKey is a system for retrieving and visualizing spatial objects returned by SPM queries and some other well-known SKQs. Thus, SpaceKey allows users to perform comparison analysis among queries.

We formulate the SPM problem in Section 2. Section 3 presents our SPM solutions. We introduce SpaceKey in Section 4. We discuss future research directions and conclude in Section 5.

2 Problem Definition

Let D be a database of spatial objects (or *objects* for brevity). Each object $o_i \in D$ ($1 \leq i \leq |D|$) has 2D coordinates (x_i, y_i), and is associated with a set of keywords, denoted by $doc(o_i)$. We say that o_i *matches* with a keyword w, if $w \in doc(o_i)$. Given two objects o_i and o_j, we use $|o_i, o_j|$ to denote their Euclidean distance. We denote a spatial circle with center o and radius r by $O(o, r)$. Next, we formally define spatial patterns.

Definition 1 (spatial pattern): A spatial pattern P is a graph $P(V, E)$ of n vertices $\{v_1, v_2, \cdots, v_n\}$ and m edges, such that the following constraints hold:

- Each vertex $v_i \in V$ has a keyword w_i;
- Each edge $(v_i, v_j) \in E$ has a distance interval $[l_{i,j}, u_{i,j}]$, where $l_{i,j}$ ($u_{i,j}$) is the lower (respectively upper) bound of distances between two matching objects in D;
- Each edge $(v_i, v_j) \in E$ is associated with one of the signs: (1) $v_i \rightarrow v_j$; (2) $v_i \leftarrow v_j$; (3) $v_i \leftrightarrow v_j$; and (4) v_i–v_j.

Let (v_i, v_j) be an edge in E, with distance interval $[l_{i,j}, u_{i,j}]$. Also, let o_k and o_l be the two objects returned in a match of E, where $w_i \in doc(o_k)$ and $w_j \in doc(o_l)$. We now discuss the four possible *signs* of an edge in Definition 1:

- $v_i \rightarrow v_j$ [v_i excludes v_j]: No object with keyword w_j in D should have a distance less than $l_{i,j}$ from o_k.
- $v_i \leftarrow v_j$ [v_j excludes v_i]: No object with keyword w_i in D should have a distance less than $l_{i,j}$ from o_l.
- $v_i \leftrightarrow v_j$ [mutual exclusion]: No object with keyword w_j in D should have a distance less than $l_{i,j}$ from o_k, and the distance of any object with keyword w_i in D should be at least $l_{i,j}$ away from o_l.
- v_i–v_j [mutual inclusion]: The occurrence of any object (other than o_k and o_l) with keywords w_i and w_j in D with distance shorter than $l_{i,j}$ is allowed.

For example, consider the edge *house*$\rightarrow$*school* with distance interval $[0.2, 0.5]$ (*km*) in the pattern of Fig. 1(b). Intuitively, the user wishes to retrieve two objects (say, o_s and o_t) such that: (1) o_s and o_t have keywords *house* and *school* respectively; (2) the distance of o_s from o_t is between $0.2km$ and $0.5km$; and (3) there does not exist any object with keyword *school*, which is less than $0.2km$ from o_s. The arrow in *house*$\rightarrow$*school* is expressed as *house excludes school*, and captures the user's intention of not getting any match for which the *house* has a *school* object less than $0.2km$ from it. We define an *e-match* of an edge (v_i, v_j), as follows:

Definition 2 (e-match): Two objects o_k and o_l constitute an e-match of (v_i, v_j), if $doc(o_k)$ and $doc(o_l)$ include w_i and w_j, respectively, and the objects satisfy the distance constraints of (v_i, v_j).

Definition 3 (match): Given a spatial pattern $P(V, E)$ and a set S of objects, S is a match of P if there exists a surjection $\phi : V \rightarrow S$, such that for all $v, v' \in V$, if $(v, v') \in E$, then the object pair $(\phi(v), \phi(v'))$ forms an e-match of (v, v').

Problem definition. Given a database D of spatial objects and a spatial pattern P, spatial pattern matching (SPM) aims to find all the matches of P in D.

For example, in Fig. 1(a), the four objects connected in solid lines are a match of the pattern in Fig. 1(b) and they form an answer to this SPM query.

3 Solutions

Although SPM can be used for a wide range of applications, it is computational intractable since it is NP-hard as proved in [10]. To solve the SPM problem, we develop two efficient algorithms (i.e., `MPJ` and `MSJ`) based on the IR-tree index [4, 16]. Due to the space limitation, we focus on introducing the key ideas of the second solution. In the following, we first introduce the three key techniques in `MSJ` and then discuss the overall algorithm.

3.1 The Bounded Pattern

Given a spatial pattern P, we define its ***bounded pattern*** $\widehat{P}$ as a clique graph satisfying properties:

- There are n vertices $\{\widehat{v_1}, \widehat{v_2}, \cdots, \widehat{v_n}\}$;
- Each vertex is linked with each other vertex;
- $\forall(\widehat{v_i}, \widehat{v_j})$ of $\widehat{P}$, its distance interval $[\widehat{l_{i,j}}, \widehat{u_{i,j}}]$ is initialized as $[l_{i,j}, u_{i,j}]$ if $(v_i, v_j) \in P$, or $[0, +\infty]$ if $(v_i, v_j) \notin P$.
- The distance intervals on all the edges are computed by dynamic programming using Eqs (1) and (2).

$$\widehat{u_{i,j}} = \min_{1 \le k \le n} \{\widehat{u_{i,j}}, \widehat{u_{i,k}} + \widehat{u_{k,j}}\}. \tag{1}$$

$$\widehat{l_{i,j}} = \max_{1 \le k \le n} \begin{cases} 0 & [\widehat{l_{i,k}}, \widehat{u_{i,k}}] \cap [\widehat{l_{k,j}}, \widehat{u_{k,j}}] \neq \emptyset \\ \widehat{l_{k,j}} - \widehat{u_{i,k}} & \widehat{u_{i,k}} < \widehat{l_{k,j}} \\ \widehat{l_{i,k}} - \widehat{u_{k,j}} & \widehat{l_{i,k}} > \widehat{u_{k,j}} \end{cases}. \tag{2}$$

We can observe that, when computing the lower and upper bound distances between any two vertices using Eqs (1) and (2), we have considered all the paths between them, and so they are *globally* tight. This implies, we can use them to refine P, which may reduce the query computational cost.

Let e=v_i–v_j be an edge with mutual inclusion. We have the following refining criteria:

❶ If $[l_{i,j}, u_{i,j}] \cap [\widehat{l_{i,j}}, \widehat{u_{i,j}}] = \emptyset$, then P is a wrong pattern, since no pair of objects can satisfy the distance constraint.

❷ If $[\widehat{l_{i,j}}, \widehat{u_{i,j}}] \subset [l_{i,j}, u_{i,j}]$, we delete (v_i, v_j), as any set of objects matched with $P \backslash e$ is also a match of P.

❸ If neither criterion ❶ nor criterion ❷ can be applied, then we refine $[l_{i,j}, u_{i,j}]$ as $[l_{i,j}, u_{i,j}] \cap [\widehat{l_{i,j}}, \widehat{u_{i,j}}]$, since any set of objects matched with P is also a match of $\widehat{P}$.

We illustrate above refining criteria by Example 1 as follows.

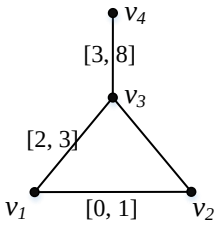


Figure 3: Illustrating pattern refining.

Example 1: Consider a pattern in Figure 3, and the edge e= (v_2, v_3) has three different cases. Note that $[\widehat{l_{2,3}},\widehat{u_{2,3}}]$ is always a subinterval of [1, 4]. If [$l_{2,3}$, $u_{2,3}$]=[5, 6], then it is a wrong pattern by criterion ❶; if [$l_{2,3}$, $u_{2,3}$]=[0, 5], then we delete e by criterion ❷; and if [$l_{2,3}$, $u_{2,3}$] =[2, 5], we update it as [2, 4] by criterion ❸.

If the relationship between v_i and v_j is not mutual inclusion, we replace criteria ❷ and ❸ by criterion ❹:
❹ If $\widehat{u_{i,j}}<u_{i,j}$, we simply refine $[l_{i,j}, u_{i,j}]$ as $[l_{i,j}, \widehat{u_{i,j}}]$.

3.2 The PJ Algorithm

To compute the e-matches of an edge, we assume that there is an IR-tree built for D. Given an IR-tree and an edge with keywords w_i and w_j, PJ finds all the matched pairs of IR-tree nodes level by level in a top-down manner. Specifically, for the root level, the root node and itself form a matched pair (assume that the IR-tree has both keywords w_i and w_j). Then, we find the child node pairs that match and follow them, repeating the same process until all the matched objects at the leaf level are found.

We now explain when a pair of nodes match. Let p and q be two non-leaf nodes, whose inverted files contain w_i and w_j respectively, at the same level of the IR-tree. Intuitively, if p and q's MBRs are far from or too close to each other, we cannot find any pair of objects under them that form an e-match.Then q does not match with p if the minimum distance of their MBRs is not larger than $u_{i,j}$. Similarly, the maximum distance of their MBRs must be less than $l_{i,j}$.

To prune unmatched node pairs, we exploit a key property of the IR-tree. That is, for each node in the IR-tree, its MBR must be contained by the MBR of its parent node. As a result, after finding all matched node pairs at a specific tree level, for the next (lower) level, we can directly find the matched node pairs from their child node pairs, and ignore all the other node pairs. By repeating this process level by level, we can safely prune a large number of unmatched pairs of nodes and obtain all the e-matches.

3.3 The Join Order for MSJ

We propose a simple yet effective and efficient method to determine the join order, denoted by MSJOrder, which relies on a key observation that, in an IR-tree (or other tree-based indexes), with a typical node capacity in the hundreds and a fill-factor of approximately 0.7, the leaf level makes up well beyond 99% of the index [16]. This implies that, the number of non-leaf nodes is much smaller than that of leaf nodes. Meanwhile, the non-leaf nodes, especially those at the lowest level, generally well summarize the objects' locations, which inspires the design of PJ. Thus, we propose to use the number of matched non-leaf node pairs to approximate the join order.

Specifically, we perform three steps in MSJOrder. First, for each edge, we apply the PJ algorithm until the handling of leaf nodes, to find all the matched pairs of non-leaf nodes at the lowest level. Second, we count the number of matched pairs of non-leaf nodes for each edge. Third, we perform the same greedy algorithm as that of MPJOrder, where the estimated numbers of e-matches of edges are replaced by the corresponding numbers of matched non-leaf node pairs, and obtain a join order Γ. Note that all the sets of matched pairs of non-leaf nodes are kept after running MSJOrder, as they will be reused later in the join process.

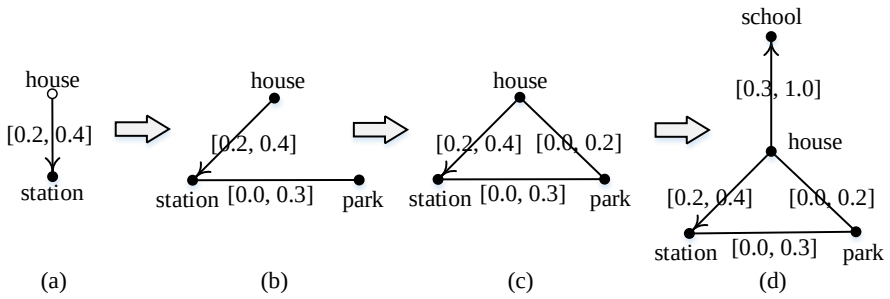


Figure 4: Illustrating the process of MSJ.

3.4 The Overall MSJ Algorithm

Based on the bounded pattern computation, PJ algorithm, and join order above, we develop the MSJ algorithm. Specifically, we first compute the bounded pattern $\widehat{P}$ of P using dynamic programming and refine P. Then, we find the matched non-leaf node pairs for all the edges of P in a collective manner, through which the join order Γ is computed. After that, for each edge of Γ, we compute its e-matches by using PJ. Finally, we follow the order Γ and compute all the matches by linking these e-matches. In Figure 4, we illustrate the process of joining e-matches in MSJ, which follows a particular join order. For detailed pseudocodes, please refer to [6, 10].

4 Application: SpaceKey

In this section, we present *SpaceKey*, a system for retrieving and visualizing spatial objects returned by various SKQs, such as SPM query, mCK query, etc. Moreover, it allows users to perform comparison analysis between different SKQs. We further customize it for a real application of property searching, by resolving practical issues, as well as realizing some additional functionalities related to property searching. We first introduce SpaceKey in Section 4.1 and then present the customized system for property searching in Section 4.2.

4.1 SpaceKey: Exploring Patterns in Spatial Databases

In SpaceKey, to issue an SPM query, the user can easily draw a spatial pattern and view the query results. Figure 5 shows the user interface of SpaceKey configured to run on a dataset of UK Points of Interest (PoIs). To draw a pattern, a user can drag icons (representing keywords) from the panel (bottom-left) to create vertices (top-left), and then create edges by linking pairs of icons. Their distance intervals and relationship can be edited using the pop-up panel, which overlaps with the map in Figure 5. Once the user clicks the "Query" button, all the matches will be returned and the user can view them one by one through the "Previous" and "Next" buttons. Additionally, SpaceKey allows a query user to edit a previous spatial pattern, which would help her to interactively compose a new query pattern and explore the matched objects.

Besides, SpaceKey can seamlessly supports other SKQ algorithms. Currently, we have incorporated three additional SKQ query types: mCK [14], CoSKQ [1, 15], and minSK [3]. Thus, a user can choose which query model to use to fit her needs. In Figure 5, a query user can issue a specific query after clicking its type on top of the left panel. Moreover, SpaceKey provides an Application Programmer Interface (API), which consists of a list of functions. To plug a new SKQ query type or algorithm into SpaceKey, the user only needs to follow the API and slightly modifies the HTML codes in the panel under the logo, and then she can easily view the query results and compare them with other SKQ query algorithms.

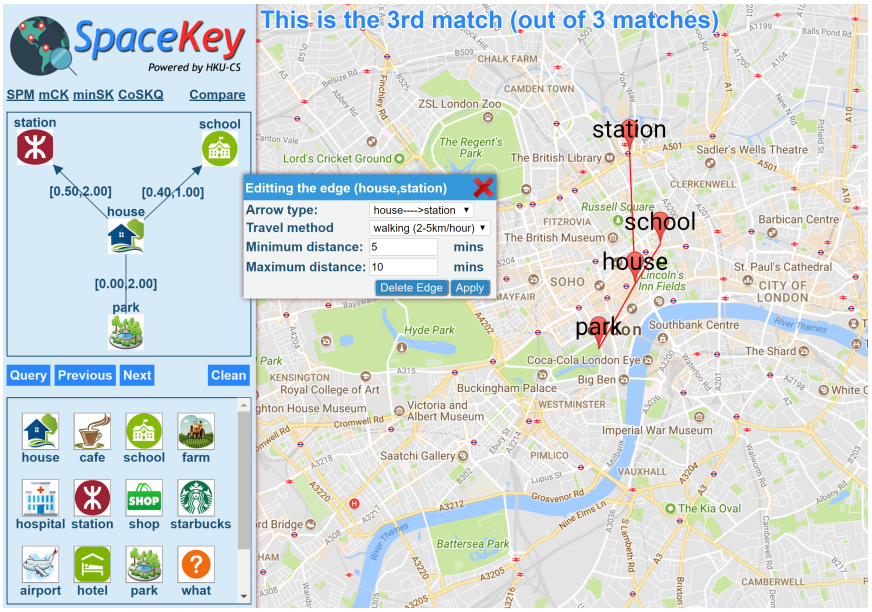


Figure 5: User interface of SpaceKey [9].

Furthermore, SpaceKey can report statistics about the results returned by different algorithms, such as the number of object sets returned, the average pair-wise distance between objects in each set, and the diameter of objects. These features allow users to perform a detailed comparison between the results output from different algorithms. Additionally, users can plug in their own analysis functions through the API provided. For more details of SpaceKey, please refer to [9].

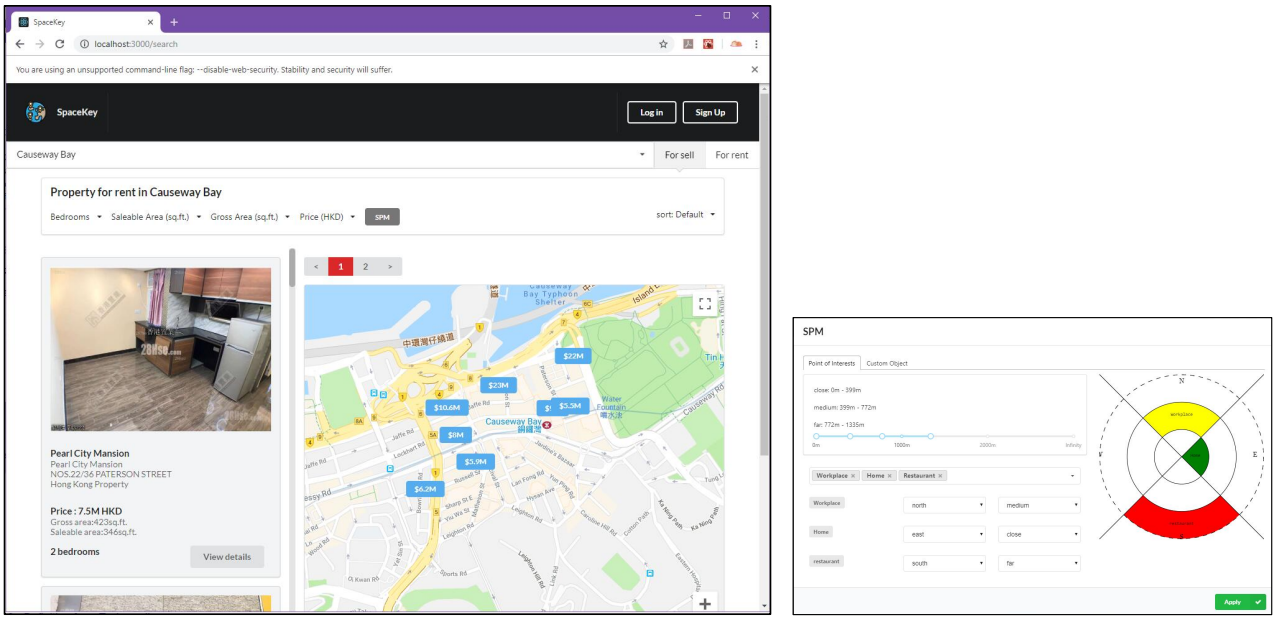
4.2 SpaceKey for Property Searching

The demand for property is steadily rising with the significant market size of real-estate across the world. Existing property searching applications only provide simple filtering functionalities about the properties, such as price, size, etc. However, in reality, users might have some complex requirements about the surroundings of their desired properties. To deal with this unsatisfied demand, we customize SpaceKey so that it enables users to perform property searching. Next, we first introduce the interfaces and then discuss additional features.

Figure 6(a) shows the main interface of the customized system. After specifying the query requirements (e.g., room types, areas, and price), the query results will be shown on the map, including the points of interests (POIs) and the properties that satisfies the requirements. On the left, the detailed information of the properties is displayed and the "View details" button will redirect the user to the original website. The user can also get the routing info between locations through simple maneuvers.

Figure 6(b) is the panel for specifying user's desired surroundings. SpaceKey allow users to specify three distance intervals, which stands for close, medium and far. Users could modify these three intervals by dragging points along the axis. In the input bar the user can search and add existing keywords to specify the pattern. For directions, the user can choose north, west, east, west or any; For distance, the user can choose close, medium, far, any or unwanted. On the right is a comprehensive pattern display: As the user add more keywords to the pattern, their respective area will be highlighted. In this panel, the user can also specify custom objects.

Besides SPM patterns, it includes additional features related to the housing location, which might be helpful for the user to find the desired property. The additional features include:



(a) Search page overview

(b) SPM filter panel

Figure 6: SpaceKey overview.

- **Unwanted objects**: the users can set a distance constraint to "unwanted" when they are specifying their desired surroundings.
- **Custom objects**: for users who want to find a property to have strong ties to some specific locations, for example, his workplace or friends' house, it allows them to specify custom objects in the spatial pattern. In other words, the user can add temporary objects to the POI database with its unique keyword, which enables the user with the ability to specify the distance/directional constraints between his/her desired property and the custom object.
- **Directional relations**: users can specify the direction of their desired PoIs. For example, a Hong Kong user may want a temple in the north of her property for the sake of fortune.

5 Conclusions and Future Work

In this paper, we propose the SPM problem and devise efficient solutions to address the SPM problem. Based on the SPM query, we develop a system called SpaceKey. We also develop two applications, namely spatial pattern visualisation and advanced hotel search. There are interesting avenues for future work, for example:

• **Diversified top-k SPM:** As shown by Spacekey, sometimes the number of matches returned by an SPM query is enormous, if the query pattern is not well specified. As a result, the user may have difficulty to rank the numerous results and choose the best ones. To address this issue, it is desirable to enable a new kind of top-k query that maximized the "diversity" of the query results. For example, we can define the diversity as the number of distinct objects in the returned matches, and then study how to find the top-k matches that maximize the diversity.

• **Approximate SPM**: An SPM query may not always return one or more matches. This can be because the query pattern is rare, and consequently there are no instances that precisely match it. In such cases, it is better to find sets of objects that are approximately matched with the query pattern. For instance, if we cannot find an exact match for the query pattern, the query algorithm can automatically relax the distance constraints such that

there is at least one match for the pattern in the database. Note that a key challenge in such approximate SPM is how to minimize relaxation on the constraints. Another choice is to ask the user to specify a priority value of each edge in the pattern, and then edges with higher priority should be satisfied when finding the matched objects.

• **Distributed solutions of SPM:** Real-world spatial datasets, such as Google Maps, are often kept in a cluster of machines distributedly. In this case, it is necessary to develop distributed solutions to support SPM query. Besides, to accelerate the SPM queries, it is natural to develop distributed solutions that support SPM queries to improve the throughput of query processing.

• **Spatial pattern discovery:** In this paper, we assume that the query users can specify proper patterns as input. This assumption, however, may be too strong for users who are not familiar with the spatial database. To remedy this issue, it is better to suggest some meaningful patterns. To discover these patterns, a natural idea is to adapt the existing solutions of frequent subgraph mining such that the frequent spatial patterns can be discovered from the spatial database.

• **More features in spatial patterns:** There are many possible ways to enrich the spatial pattern. For example, we can allow each vertex of the pattern to carry multiple keywords with logical operations (e.g., "AND" and "OR"), supporting for instance the case where users want to find a house that has nearby a hospital or a doctor. Also, the distance constraint can be changed, such that multiple distance intervals can be specified between two vertices in the spatial pattern.

Acknowledgments

Reynold Cheng was supported by the Research Grants Council of Hong Kong (RGC Projects HKU 17229116 and 17205115) and the University of Hong Kong (Projects 102009508 and 104004129). We would like to thank Jikun Wang, Budiman, and Yin Yue for their implementation of SpaceKey.

References

[1] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In *SIGMOD*, pages 373–384. ACM, 2011.

[2] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: an experimental evaluation. *PVLDB*, pages 217–228, 2013.

[3] D. Choi, J. Pei, and X. Lin. Finding the minimum spatial keyword cover. In *ICDE*, pages 685–696. IEEE, 2016.

[4] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *VLDB*, 2(1):337–348, 2009.

[5] K. Deng, X. Li, J. Lu, and X. Zhou. Best keyword cover search. *TKDE*, 27(1):61–73, 2015.

[6] Y. Fang, R. Cheng, G. Cong, N. Mamoulis, and Y. Li. On spatial pattern matching. In *ICDE*, pages 293–304. IEEE, 2018.

[7] Y. Fang, R. Cheng, X. Li, S. Luo, and J. Hu. Effective community search over large spatial graphs. *PVLDB*, 10(6):709–720, 2017.

[8] Y. Fang, R. Cheng, W. Tang, S. Maniu, and X. Yang. Scalable algorithms for nearest-neighbor joins on big trajectory data. *IEEE Transactions on Knowledge and Data Engineering*, 28(3):785–800, 2015.

[9] Y. Fang, R. Cheng, J. Wang, Budiman, G. Cong, and N. Mamoulis. SpaceKey: exploring patterns in spatial databases. In *ICDE*, pages 1577–1580. IEEE, 2018.

[10] Y. Fang, Y. Li, R. Cheng, G. Cong, and N. Mamoulis. Evaluating pattern matching queries for spatial databases. In *VLDB Journal*. Springer-Verlag New York, Inc., 2019.

[11] Y. Fang, Z. Wang, R. Cheng, X. Li, S. Luo, J. Hu, and X. Chen. On spatial-aware community search. *IEEE Transactions on Knowledge and Data Engineering*, 31(4):783–798, 2018.

[12] B. Gallagher. Matching structure and semantics: A survey on graph-based pattern matching. *AAAI FS*, 6:45–53, 2006.

[13] A. B. Gallion and S. Eisner. The urban pattern; city planning and design,. 1950.

[14] T. Guo, X. Cao, and G. Cong. Efficient algorithms for answering the m-closest keywords query. In *SIGMOD*, pages 405–418. ACM, 2015.

[15] C. Long, R. C. W. Wong, K. Wang, and A. W. C. Fu. Collective spatial keyword queries: A distance owner-driven approach. In *SIGMOD*, 2013.

[16] D. Wu, M. L. Yiu, G. Cong, and C. S. Jensen. Joint top-k spatial keyword query processing. *TKDE*, 2012.

[17] Y. Wu, J. M. Patel, and H. Jagadish. Structural join order selection for xml query optimization. In *ICDE*, pages 443–454. IEEE, 2003.

[18] W. Zhang, R. Cheng, and B. Kao. Evaluating multi-way joins over discounted hitting time. In *ICDE*, pages 724–735. IEEE, 2014.

[19] D. Zhang et al. Keyword search in spatial databases: towards searching by document. In *ICDE*, pages 688–699. IEEE, 2009.

[20] L. Zou, L. Chen, and M. T. Özsu. Distance-join: pattern match query in a large graph database. *PVLDB*, 2(1):886–897, 2009.

Spatial Joins: What's next?

Panagiotis Bouros
Institute of Computer Science
Johannes Gutenberg University Mainz, Germany
bouros@uni-mainz.de

Nikos Mamoulis
Dept. of Computer Science and Engineering
University of Ioannina, Greece
nikos@cs.uoi.gr

Abstract

The spatial join is a popular operation in spatial database systems and its evaluation is a well-studied problem. This paper reviews research and recent trends on spatial join evaluation. The complexity of different data types, the consideration of different join predicates, the use of modern commodity hardware, and support for parallel processing open the road to a number of interesting directions for future research, some of which we outline in the paper.

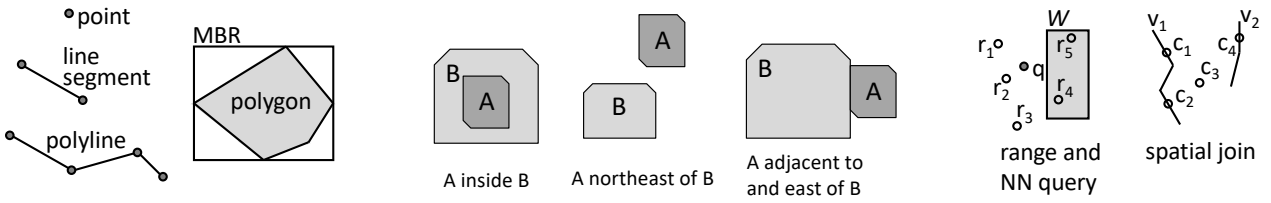
1 Introduction

Spatial data are nowadays more ubiquitous than ever before; examples of such data include meteorological maps, biological and scientific data, socio-economic data, agricultural data and geo-tagged social media. Thanks to the proliferation of mobile location-aware devices and services (e.g., smartphones, tablets, wearables, GPS devices, mobile applications, satellites), the volume of spatial data generated every single day increases at a staggering and unprecedented rate, and so does the number of applications and domains where such data are collected and analyzed. The challenges and the importance of big spatial data management have been extensively sung, e.g., in [14, 19, 48].

The *spatial join* is a fundamental data operation. Traditionally, such a join finds application in spatial data management systems [16] and Geographic Information Systems (GIS) [24]. GIS, for example, typically store multiple thematic layers (e.g., road network, hydrography), which are spatially joined in order to find object pairs (e.g., roads and rivers) that intersect. In addition, spatial joins are used to support data mining operations such as clustering [15] and pattern detection [11]. Today, spatial joins play a key role also in scientific applications. For instance, they can determine neuron synapses in brain models, i.e., pairs of neuron branches that are within very small distance to each other [27], while in medical imaging, spatial joins are used to determine proximity of cells and to facilitate the analysis of high resolution images of human tissue specimens for more effective diagnosis, prediction and treatment of diseases [3].

In this article, we focus on the most common definition of spatial joins termed the *spatial intersection* join. The goal is to determine pairs of spatial objects, e.g., (road, river) pairs, that have at least one common point in space. Besides its popularity and wide adoption, note that evaluation methods for the spatial intersection join can be also used for other types of spatial joins such the *spatial distance* and the *nearest neighbor* joins.

A wide range of spatial join algorithms have been proposed in the literature [18, 25]. Most of them assume that the input data are disk-based and their objective is to minimize I/O accesses during the join. Given the fact that main memories are constantly becoming bigger, cheaper and faster, in-memory join processing has recently received significant attention [29]. In addition, given that commodity hardware supports parallel processing, multi-core join evaluation has also been the focus of recent research. In line with this trend, this article focuses on parallel in-memory evaluation of spatial joins on modern hardware. Our goal is to describe the landscape of efficient spatial join computation under this prism and to discuss interesting directions for future research.



(a) data types (b) spatial relationships (c) spatial queries

Figure 1: Examples of spatial data, relationships and queries

2 Background

In this section, we provide a brief background on spatial data management [26], discussing basic data and query types, indexing structures and principles for query evaluation.

Common types of spatial objects include the *point* (defined by one value per dimension), the *rectangle* (defined by one interval per dimension), the *line segment* (defined by a pair of points), the *polygon* (defined by a sequence of points), the *polyline* (defined by a sequence of points), etc. Figure 1(a) shows examples of these datatypes on the plane. To characterize the relative position between two spatial objects, three classes of *spatial relationships* can be used. *Topological* relationships (e.g., overlap, inside, contains, disjoint, etc.) model relationships between the geometric extents of objects. *Directional* relationships (e.g., north/south, east/west, above/below, left/right, etc.) compare the relative locations of the objects with respect to a coordinate (or cardinal) system. Last, *distance* relationships capture distance information between two objects. Figure 1(b) illustrates examples of spatial relationships.

The most frequently applied query operation on spatial data is the *spatial selection* (or *range query*) which asks for the objects that intersect (or are inside) a spatial range, or are within some distance from a reference spatial object. Another popular operation is the *nearest neighbor search* which asks for the objects that are the nearest to a reference location. Finally, the *spatial join* finds pairs of objects from two collections that qualify a spatial predicate. The most common join operation is the *spatial intersection join* which we primarily discuss in this article. Formally, given collections R and S, the objective is to find pairs (r, s), such that $r \in R$, $s \in S$ and their geometries intersect, i.e., r and s have at least one common point. Other popular definitions of spatial joins include the ϵ-*distance join* which determines (r, s) pairs of objects within at most distance ϵ from each other, and the *nearest-neighbor* join which returns for every object $r \in R$, the nearest objects from input collection S. As an example of a spatial selection assume that r_1 to r_5 in Figure 1(c) are locations of restaurants and that a user is interested in finding which restaurants are located in region W. The result includes objects r_4 and r_5. If the objective is to find the nearest restaurant to location q, the result is r_2. Assuming that v_1 and v_2 are rivers and c_1 to c_4 are cities, the result of the spatial intersection joins between these two sets are pairs $\{(v_1, c_1), (v_1, c_2), (v_2, c_4)\}$.

Due to the potentially complex geometry of the objects, spatial query evaluation is typically performed in two steps. In the *filter* step, the query is applied on the *minimun bounding rectangles* (MBRs) of the objects, which are simple lightweight approximations; see for example the MBR of a polygon in Figure 1(a). Object MBRs (or pairs of object MBRs) that do not qualify the query can be pruned in the filter step, while for the objects (or object pairs) that pass the filter an expensive *refinement step* is applied using their exact geometries. For example, if the MBR of a river does not intersect the MBR of a city, there is no chance that the exact geometries of the two objects intersect.

In order to process spatial queries efficiently, a number of spatial access methods have been proposed to index the spatial object collections. The most popular spatial index is the R-tree and its variants [17, 6], a balanced tree, similar to the B$^+$-tree, which groups nearby object MBRs into the leaf nodes of the tree. The non-leaf nodes are formed by hierarchically grouping nearby MBRs of lower-level nodes. When evaluating a

spatial query, a node (and the corresponding sub-tree) whose MBR does not qualify the predicate of the query can be pruned, drastically reducing the search space. A simpler index for memory-resident data is a spatial grid, which divides the space into cells and its more sophisticated quadtree version [41].

3 Join Computation Landscape

Given two large collections of spatial objects R and S, a spatial join is in principle processed by first dividing the collections into small partitions and then joining these partitions. For each collection, we may use an existing partitioning or index; in either case, the join is broken down into a number of small joins that can be processed fast in memory. In what follows, we overview the landscape in computing spatial join queries and organize literature in four key categories. Note that the vast majority of the proposed evaluation methods focus on the filter step.

3.1 Small Join Computation

For in-memory processing of small spatial joins, a typical approach is to use adaptations of a plane sweep algorithm that compute rectangle intersections [37]. The most commonly used adaptation was proposed by Brinkhoff et al. in [10]. In brief, the collections are first sorted based on their lowest value in one dimension; then, the sorted inputs are scanned concurrently and merged in a merge-join fashion. This process resembles a perpendicular line that sweeps along the sorting dimension. Every time the sweep line stops, e.g., at object r from R, the algorithm *forwardly* scans input S to produce the join results. Arge et al. studied in [5] a variation of the plane sweep algorithm which maintains an active list of previously encountered objects at every position of the sweep line, performing essentially a *backward scan*. In practice, the performance gain over [10] is insignificant unless a special data structure is used to organize active lists, similar to the gapless hash map of [35].

3.2 Data Partitioning

Data partitioning has been considered as a *divide-and-conquer* approach to split the input collections into smaller subsets that can then be spatially joined fast in memory. A partition from input R is then joined with a partition from S only if their MBRs intersect. A large number of spatial join algorithms that follow this paradigm have been proposed; the methods can be classified into *single-assignment*, multi-join (SAMJ) methods and *multi-assignment*, single-join (MASJ) approaches [23]. SAMJ methods assign each object to exactly one partition; the partitions are determined by spatial clustering heuristics. A partition from one input may have to be joined with multiple partitions of the other. The *R-tree Join* algorithm (RJ) in [10] is a classic SAMJ approach when both inputs are indexed by an R-tree. RJ starts by finding all pairs of entries (e_R, e_S) one from each root node of the trees that intersect. For each such pair, the algorithm recursively applies the same procedure for the nodes pointed by e_R and e_S, until pairs of leaf node entries (which correspond to intersecting object MBRs) are found. A SAMJ approach that does not rely on pre-defined indices is *Size Separation Spatial Join* from [20].

On the other hand, the borders of the partitions in MASJ are pre-determined, and so an object is assigned to every partition it spatially intersects. Each partition from one collection is joined with exactly one partition from the other (which has exactly the same MBR). The most popular MASJ approach is *Partition-based Spatial Merge Join* (PBSM) [32] which divides the space by a regular grid and assigns objects from both input collections to all tiles that spatially overlap them. For each partition, PBSM accesses the objects from both inputs and performs the small join in memory (e.g., using plane sweep). Since an object can be replicated to multiple tiles, this method can generate duplicate results; duplicates can be eliminated by reporting a join result in a tile only if a specific corner of the intersection falls in the tile [12]. Other MASJ approaches include *Spatial Hash Join* from [23] and *Scalable Sweeping-Based Spatial Join* from [5].

More recent work investigates the role of the distribution and density of the input collections. Motivated by a neuroscience application, which requires joining datasets of contrasting density, Pavlovic et al. design a spatial join algorithm in [34] that partitions the dense dataset. Then, the algorithm 'crawls' through the partitions guided by the object locations in the sparse dataset, skipping partitions that do produce any results. Based on the same motivation, a more sophisticated approach in [33] called *TRANSFORMERS*, adapts the type of partitioning (MASJ or SAMJ) and the join technique used locally, depending on the differences in the densities of the two inputs.

3.3 In-Memory Processing

Despite the recent advances that allow us to store the entire input collections in main memory, directly applying plane sweep can be too expensive. This is due to the large number of candidates produced by forward scans, which do not materialize to actual results. Hence, in-memory join approaches also consider data partitioning or indexing to accelerate the join computation. For example, a grid (similar to PBSM) can be used to break the problem into numerous small instances that can be solved fast. The *TOUCH* algorithm from [30] is an in-memory algorithm, designed for scientific applications that join huge datasets that have different density and skew. TOUCH first bulk-loads an R-tree for one of the inputs using the STR technique [21]. Then, all objects from the second input are assigned to buckets corresponding to the non-leaf nodes of the tree. Each object is hashed to the lowest tree node, whose MBR overlaps it, but no other nodes at the same tree level do. Finally, each bucket is joined with the subtree rooted at the corresponding node with the help of a dynamically created grid data structure for the subtree. A recent comparison of spatial join algorithms for in-memory data [29] shows that PBSM and TOUCH perform best and that the join cost depends on the data density and distribution. Tauheed et al. [45] suggest an analytical model for configuring the grid of PBSM-like join processing in main memory.

3.4 Distributed and Parallel Processing

Early efforts on parallelizing spatial joins include extensions of the R-tree join and PBSM algorithms in a distributed environment of single-core processing nodes with local storage. For R-tree join, overlapping pairs of root entries essentially define independent join processes on the corresponding sub-trees. Hence, these tasks can be assigned to different computer nodes [9]. To reduce I/Os, a virtual global buffer is shared among the processing nodes to avoid accessing the same data multiple times from the disk. An early approach in parallelizing PBSM was presented in [54]; the method asks the processing nodes to perform the partitioning of data into tiles independently and in parallel. Then, each computer node is assigned one partition and then nodes exchange data, so that each one gets all objects that fall in its partition. The join phase is finally performed in parallel.

Recently, the research interest shifted to spatial join processing for distributed cloud systems and multi-core processors. Pandey et al. conducted an experimental evaluation in [31] between parallel and distributed spatial data management systems where, among other queries, spatial joins were tested. Essentially, approaches in this context fall into two categories. The first category capitalizes on the popularity and commercial success of the MapReduce framework. For instance, the *Spatial Join with MapReduce* (SJMP) algorithm from [53] is an adaptation of the PBSM algorithm where grid tiles are examined in a space-filling curve order and grouped to partitions in a round-robin fashion. *Map* tasks assign input objects to one or more partitions based on the tiles they overlap, while every partition is then processed by a separate *reduce* task. The reducers perform their joins by dividing the space that corresponds to them into stripes and then applying plane sweep. Duplicate results are avoided by reporting a join pair only at the tile with the smallest id where the two objects commonly appear. *Handoop-GIS* [4] processes spatial joins in a similar manner, i.e., by applying a regular grid. A key difference is that the data objects are both globally and locally indexed. A *global index* shared between nodes, is used to find the HDFS files where the contents of each tile are stored. A *local index* is defined on every processing node to independently perform a spatial join. Finally, the results are merged. *Spatial-Hadoop* [13] also employs a global

index, stored on the Master node. The system investigates the best join strategy when partitioning schemes pre-exist the queries for the input collections. If the two join inputs are partitioned differently essentially, we could either directly use the existing partitions and join every pair of overlapping partitions or re-partition the smaller input according to the partitioning of the larger and then use PBSM. The cost of each the two strategies is estimated and the cheapest one is selected accordingly. A query optimizer for MapReduce-based spatial join algorithms is presented in [40]. The second category of systems build on top of Apache Spark [51]. *Simba* [46], *SpatialSpark* [47], *GeoSpark* [49], *LocationSpark* [44] and *Magellan* [1] store both the indexing structures and the intermediate results in memory shared by all nodes in the cluster. Computing spatial joins is focused on effective spatial indexing of the *resilient distributed datasets* (RDDs).

4 Directions for Future Research

While there has been a vast amount of work on spatial join evaluation, the continuous evolution of hardware and the increase of volume and variety of big spatial data opens new, promising research directions. In this section, we briefly discuss research challenges and open issues related to spatial join evaluation.

4.1 Specialized join algorithms for different datatypes and accelerating the refinement step

Most spatial join algorithms treat all types of spatial data in the same way. Specifically, they solve the join problem on the MBRs of the objects in a *filter step* and then, for each pair of intersecting rectangles, they apply a *refinement step*. However, for certain (simple) data types, we can design specialized join algorithms that apply directly on the exact geometric representations of the objects, while achieving a cost similar to that of the MBR intersection join. In a recent work in this direction [28], a multi-core spatial join algorithm based on plane sweep was designed for line segment collections. A promising direction is to explore the development of algorithms that operate in the same spirit and handle other spatial datatypes such as polylines.

The refinement step of a spatial join can be much more expensive than the filter step if the objects have complex spatial extent. A number of ideas have been proposed in the direction of reducing the cost of the refinement step. In this direction, besides the MBR, alternative object approximations [8, 55] and *true hit filtering* can be applied. For example, objects can be approximated by the maximum rectangles or circles that are enclosed in them. If two such approximations intersect, then we can guarantee that the exact geometries of the objects intersect [8]. Another idea is to define a raster representation for each object, where a set of coarse pixels (i.e., cells of a fine grid) is used. For the cells that each object intersects, we can measure the percentage the cell that the object covers. If two objects intersect a cell by at least 50%, this guarantees that they are a spatial join result. An interesting research direction is to parallelize this technique (i.e., handle each cell in parallel) and the challenge would be to (i) handle duplicates, (ii) minimize the required space due to object replication.

Two recent pieces of related work study the point-to-polygon joins. Zacharatou et al. [50] use raster representations of the objects and employ GPUs to parallelize the spatial join, by handling each pixel in parallel. There is no issue of duplicate elimination because a point is guaranteed to intersect a polygon in at most one cell. Kipf et al. [19] evaluate point to polygon joins again using object decompositions (by a quadtree representation) and paralellization, aiming at minimizing the refinement cost. There is still room for improving the mechanisms for avoiding refinements wherever possible and for optimizing how refinements are implemented and scheduled in a distributed environment [39]. An interesting direction is to study polygon-to-polygon joins in parallel using raster or quadtree representations for each object. The main challenge is to avoid the production of duplicate results and the computational overhead for their computation.

4.2 Distance and nearest neighbor joins

Distance and nearest neighbor joins [52] are evaluated by extending or adapting algorithms for intersection joins. Special algorithms that take into consideration the fact that such joins are typically conducted between point-sets could be designed. For example, Bouros et al. study the computation of ϵ-distance joins in [7], where the objective is to find pairs of points within distance at most ϵ to each other. A regular grid, where the projections of the cells at each axis have length at most ϵ is used to online partition the data. Then, each cell needs to be joined with at most five cells in order to produce the result. Interesting research directions would be to parallelize this approach and to also consider objects which are not points. A first attempt towards the former was presented in [42] for distributed spatial data on top of MapReduce. For the point-to-polygon distance join, a combination of approaches from [50, 19] and [7] could be applied. Nearest neighbor joins are more challenging because the nearest neighbor of a point might not be located in the same or neighboring cells. Using statistics about the cell occupancies or a non-uniform (quadtree-style) partitioning could be a way for approaching this problem.

4.3 Scaling up vs. scaling out

In most of the applications that manage spatial data, the inputs can be preprocessed and then easily fit in the main memory of a single commodity machine. For example, in most public spatial data collections (see, for example, http://spatialhadoop.cs.umn.edu/datasets.html) the number of objects is in the order of 100M or less. Such data collections can fit in the memory of a commodity machine. Hence, scaling up (i.e., solving a medium size problem more efficiently) might be a more relevant problem compared to scaling out to larger data sizes that need large clusters or cloud computing algorithms. Multi-core parallelism is gaining ground, so a promising research direction is to design good shared-memory parallel algorithms for spatial joins, especially in applications where the data are produced and need to be processed at a high rate, i.e., streaming spatio-temporal data [19, 43]. Spatial join computation can also benefit from the high degree of parallelization achieved by using a single or multiple GPU chips [36], besides accelerating the refinement step. Aghajarian et al. present an attempt towards this direction in [2].

4.4 Optimizing partition-to-partition joins

As discussed already, spatial join algorithms for large-scale data typically operate in two phases. In the first phase, the data are partitioned and in the second phase the partitioned data are joined. If an spatial index exists for a join input, it can be used to define the partitions of the input, hence the partitioning phase for that input can be skipped. Although a large number of join algorithms have been proposed, the problems of (i) selecting the most appropriate partitioning and (ii) selecting the most appropriate technique for each partition-to-partition join has not been studied adequately. For (i), we need accurate cost models, based on spatial statistics, which can be used to determine the best way to partition the data. For (ii), a typical approach is to use plane sweep, however, this might not be efficient for joining partitions of contrasting skew and densities and for 3D data. In such cases, a finer partitioning can be used for the smaller partition to spatially hash its data and the objects in the larger partition can be probed against it [45, 33]. So far, there is no comprehensive evaluation on the conditions under which this spatial hashing partition-to-partition join technique is faster than plane sweep.

4.5 Extended join Operations

Spatial joins are traditionally defined and studied with respect to geometries or locations while ignoring other types of information attached to the input objects; an exception arises in the case of spatio-temporal joins where the temporal aspect is additionally taken into account. However, modern spatial data are not only more ubiquitous than ever but also more complex; they can be routinely assigned or associated with other types of information, e.g., text or social information. In this context, it is interesting to examine whether new join operations

can be defined building on top of spatial joins and incorporating other types of information available. Besides defining these new join operations, we also need to investigate their efficient computation; a key challenged is how to combine the different algorithms and indexing structures employed for each type of information. Efforts towards this direction include the *spatio-textual similarity* join proposed in [7] which comes as hybrid of a spatial ϵ-distance join and a set similarity join, the techniques proposed in [22] for joins in GeoRDF data and the top-k distance join in [38] which uses the numerical information present in the objects to formulate the ranking component of the join.

5 Conclusions

In this article, we briefly reviewed evaluation techniques for spatial joins, with a focus on in-memory join processing and on parallel and distributed evaluation. Although spatial joins have been studied for decades, recently the research interest has been renewed, due to the opportunities brought by modern commodity hardware, which comes together with large memories and CPUs or GPUs with high parallelization capacity. This brings in chances and challenges for re-designing classic approaches for partitioning and join evaluation and accelerating the refinement step of the join, which we outline in the paper.

References

[1] Magellan: Geospatial analytics using spark. https://github.com/harsha2010/magellan.

[2] D. Aghajarian, S. Puri, and S. K. Prasad. GCMF: an efficient end-to-end spatial join system over large polygonal datasets on GPGPU platform. In *SIGSPATIAL*, pages 18:1–18:10, 2016.

[3] A. Aji, F. Wang, and J. H. Saltz. Towards building a high performance spatial query system for large scale medical imaging data. In *SIGSPATIAL/GIS*, pages 309–318, 2012.

[4] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. H. Saltz. Hadoop-GIS: A high performance spatial data warehousing system over mapreduce. *PVLDB*, 6(11):1009–1020, 2013.

[5] L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, and J. S. Vitter. Scalable sweeping-based spatial join. In *VLDB*, pages 570–581, 1998.

[6] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, 1990.

[7] P. Bouros, S. Ge, and N. Mamoulis. Spatio-textual similarity joins. *PVLDB*, 6(1):1–12, 2012.

[8] T. Brinkhoff, H. Kriegel, R. Schneider, and B. Seeger. Multi-step processing of spatial joins. In *SIGMOD*, pages 197–208, 1994.

[9] T. Brinkhoff, H. Kriegel, and B. Seeger. Parallel processing of spatial joins using R-trees. In *ICDE*, pages 258–265, 1996.

[10] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Efficient processing of spatial joins using r-trees. In *SIGMOD*, pages 237–246, 1993.

[11] H. Cao, N. Mamoulis, and D. W. Cheung. Discovery of periodic patterns in spatiotemporal sequences. *IEEE Trans. Knowl. Data Eng.*, 19(4):453–467, 2007.

[12] J. Dittrich and B. Seeger. Data redundancy and duplicate detection in spatial join processing. In *ICDE*, pages 535–546, 2000.

[13] A. Eldawy and M. F. Mokbel. SpatialHadoop: A mapreduce framework for spatial data. In *ICDE*, pages 1352–1363, 2015.

[14] A. Eldawy and M. F. Mokbel. The era of big spatial data. *PVLDB*, 10(12):1992–1995, 2017.

[15] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, pages 226–231, 1996.

[16] R. H. Güting. An introduction to spatial database systems. *VLDB Journal*, 3(4):357–399, 1994.

[17] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[18] E. H. Jacox and H. Samet. Spatial join techniques. *ACM Transactions on Database Systems*, 32(1):7, 2007.

[19] A. Kipf, H. Lang, V. Pandey, R. A. Persa, P. A. Boncz, T. Neumann, and A. Kemper. Adaptive geospatial joins for modern hardware. *CoRR*, abs/1802.09488, 2018.

[20] N. Koudas and K. C. Sevcik. Size separation spatial join. In *SIGMOD*, pages 324–335, 1997.

[21] S. T. Leutenegger, J. M. Edgington, and M. A. López. STR: A simple and efficient algorithm for r-tree packing. In *ICDE*, pages 497–506, 1997.

[22] J. Liagouris, N. Mamoulis, P. Bouros, and M. Terrovitis. An effective encoding scheme for spatial RDF data. *PVLDB*, 7(12):1271–1282, 2014.

[23] M.-L. Lo and C. V. Ravishankar. Spatial hash-joins. In *SIGMOD*, pages 247–258, 1996.

[24] P. A. Longley, M. Goodchild, D. J. Maguire, and D. W. Rhind. *Geographic Information Systems and Science*. Wiley Publishing, 3rd edition, 2010.

[25] N. Mamoulis. Spatial join. In L. Liu and M. T. Özsu, editors, *Encyclopedia of Database Systems*, pages 2707–2714. Springer US, 2009.

[26] N. Mamoulis. *Spatial Data Management*. Morgan & Claypool Publishers, 2011.

[27] H. Markram, K. Meier, T. Lippert, S. Grillner, R. S. Frackowiak, S. Dehaene, A. Knoll, H. Sompolinsky, K. Verstreken, J. DeFelipe, S. Grant, J. Changeux, and A. Saria. Introducing the human brain project. In *FET*, pages 39–42, 2011.

[28] M. McKenney, R. Frye, M. Dellamano, K. Anderson, and J. Harris. Multi-core parallelism for plane sweep algorithms as a foundation for GIS operations. *GeoInformatica*, 21(1):151–174, 2017.

[29] S. Nobari, Q. Qu, and C. S. Jensen. In-memory spatial join: The data matters! In *EDBT*, pages 462–465, 2017.

[30] S. Nobari, F. Tauheed, T. Heinis, P. Karras, S. Bressan, and A. Ailamaki. TOUCH: in-memory spatial join by hierarchical data-oriented partitioning. In *SIGMOD*, pages 701–712, 2013.

[31] V. Pandey, A. Kipf, T. Neumann, and A. Kemper. How good are modern spatial analytics systems? *PVLDB*, 11(11):1661–1673, 2018.

[32] J. M. Patel and D. J. DeWitt. Partition based spatial-merge join. In *SIGMOD*, pages 259–270, 1996.

[33] M. Pavlovic, T. Heinis, F. Tauheed, P. Karras, and A. Ailamaki. TRANSFORMERS: robust spatial joins on non-uniform data distributions. In *ICDE*, pages 673–684, 2016.

[34] M. Pavlovic, F. Tauheed, T. Heinis, and A. Ailamaki. GIPSY: joining spatial datasets with contrasting density. In *SSDBM*, 2013.

[35] D. Piatov, S. Helmer, and A. Dignös. An interval join optimized for modern hardware. In *ICDE*, 2016.

[36] S. K. Prasad, M. McDermott, S. Puri, D. Shah, D. Aghajarian, S. Shekhar, and X. Zhou. A vision for gpu-accelerated parallel computation on geo-spatial datasets. *SIGSPATIAL Special*, 6(3):19–26, 2014.

[37] F. P. Preparata and M. I. Shamos. *Computational Geometry - An Introduction*. Springer, 1985.

[38] S. Qi, P. Bouros, and N. Mamoulis. Efficient top-k spatial distance joins. In *Advances in Spatial and Temporal Databases - 13th International Symposium, SSTD 2013, Munich, Germany, August 21-23, 2013. Proceedings*, pages 1–18, 2013.

[39] S. Ray, B. Simion, A. D. Brown, and R. Johnson. Skew-resistant parallel in-memory spatial join. In *SSDBM*, pages 6:1–6:12, 2014.

[40] I. Sabek and M. F. Mokbel. On spatial joins in mapreduce. In *SIGSPATIAL/GIS*, 2017.

[41] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.

[42] T. Seidl, S. Fries, and B. Boden. MR-DSJ: distance-based self-join for large-scale vector data analysis with mapreduce. In *BTW*, pages 37–56, 2013.

[43] B. Sowell, M. A. V. Salles, T. Cao, A. J. Demers, and J. Gehrke. An experimental analysis of iterated spatial joins in main memory. *PVLDB*, 6(14):1882–1893, 2013.

[44] M. Tang, Y. Yu, Q. M. Malluhi, M. Ouzzani, and W. G. Aref. Locationspark: A distributed in-memory data management system for big spatial data. *PVLDB*, 9(13):1565–1568, 2016.

[45] F. Tauheed, T. Heinis, and A. Ailamaki. Configuring spatial grids for efficient main memory joins. In *BICOD*, 2015.

[46] D. Xie, F. Li, B. Yao, G. Li, Z. Chen, L. Zhou, and M. Guo. Simba: spatial in-memory big data analysis. In *SIGSPATIAL/GIS*, pages 86:1–86:4, 2016.

[47] S. You, J. Zhang, and L. Gruenwald. Large-scale spatial join query processing in cloud. In *CloudDB, ICDE Workshops*, pages 34–41, 2015.

[48] J. Yu and M. Sarwat. Geospatial data management in apache spark: A tutorial. In *ICDE*, pages 2060–2063, 2019.

[49] J. Yu, Z. Zhang, and M. Sarwat. Spatial data management in apache spark: the geospark perspective and beyond. *GeoInformatica*, 23(1):37–78, 2019.

[50] E. T. Zacharatou, H. Doraiswamy, A. Ailamaki, C. T. Silva, and J. Freire. GPU rasterization for real-time spatial aggregation over arbitrary polygons. *PVLDB*, 11(3):352–365, 2017.

[51] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *HotCloud*, 2010.

[52] J. Zhang, N. Mamoulis, D. Papadias, and Y. Tao. All-nearest-neighbors queries in spatial databases. In *SSDBM*, pages 297–306, 2004.

[53] S. Zhang, J. Han, Z. Liu, K. Wang, and Z. Xu. SJMR: parallelizing spatial join with mapreduce on clusters. In *CLUSTER*, pages 1–8, 2009.

[54] X. Zhou, D. J. Abel, and D. Truffet. Data partitioning for parallel spatial join processing. In *SSD*, pages 178–196, 1997.

[55] G. Zimbrao and J. M. de Souza. A raster approximation for processing of spatial joins. In *VLDB*, pages 558–569, 1998.

Generating Traffic Data for Any City using SMARTS Simulator

Hairuo Xie, Egemen Tanin, Kotagiri Ramamohanarao, Shanika Karunasekera,
Lars Kulik, Rui Zhang and Jianzhong Qi
School of Computing and Information Systems, University of Melbourne, Australia
{xieh, etanin, kotagiri, karus, lkulik, rui.zhang, jianzhong.qi}@unimelb.edu.au

Abstract

We have recently developed a flexible traffic simulator called Scalable Microscopic Adaptive Road Traffic Simulator (SMARTS) [13]. Among many important features of SMARTS in this article, we focus on traffic generation, which is key in analyzing and understanding traffic. SMARTS is a fully distributed simulator that can run on a computer cluster, enabling it to perform large-scale simulations faster than real time. SMARTS has a number of features that empower users to build realistic simulations. For example, users can download road map data for an arbitrarily chosen area for traffic simulation.

1 Introduction

Road traffic data is useful for research on transportation problems. For example, vehicle trajectory data can be used for optimizing public transportation routes [2]. Traffic-induced air pollution can be estimated based on traffic flow models calibrated with real traffic flow data [15]. Vehicle counts and speed data can be used for training a neural network in traffic prediction [11]. Due to the limited availability of real traffic data, many researchers are interested in artificial traffic data created with software generators. Traffic data generators are built upon various road traffic models. For example, Brinkhoff's generator [3] implements the models for distributing trip origins and destinations, adjusting travel speed at road links and re-routing vehicles in certain circumstances. BerlinMOD [4] creates acceleration events in vehicle trips based on a probability model. Highly realistic traffic data can be generated from microscopic traffic simulators, such as SUMO [6], VISSIM [10] and MATSIM [14], which simulate the detailed behaviour of vehicles based on the comprehensive modelling of traffic.

Scalable Microscopic Adaptive Road Traffic Simulator (SMARTS) is a distributed microscopic simulator that is capable of performing realistic large-scale traffic simulations faster than real time [13]. We developed the simulator to support research on transportation problems. SMARTS models car-following behaviour, lane-changing behaviour and customized road rules. It also models various driving styles based on the aggressiveness in driving. For example, a vehicle with a highly aggressive model follows its front vehicle more closely and changes lanes more frequently. Our simulator can be used to generate several types of traffic data, including route plans, vehicle trajectories and travel times.

The simulator is finely tuned for large-scale simulations as it can use distributed computing resources to accelerate simulations. Users can easily deploy SMARTS on a cluster of computers with commodity hardware. The simulator builds road networks from the freely-available OpenStreetMap data [9], which can be important to researchers who have limited access to commercial road map data. The Graphical User Interface (GUI) of SMARTS provides a feature for downloading OpenStreetMap data of an arbitrary area in the world. The simulator has a number of other useful features. For example, it can simulate random traffic and pre-defined traffic

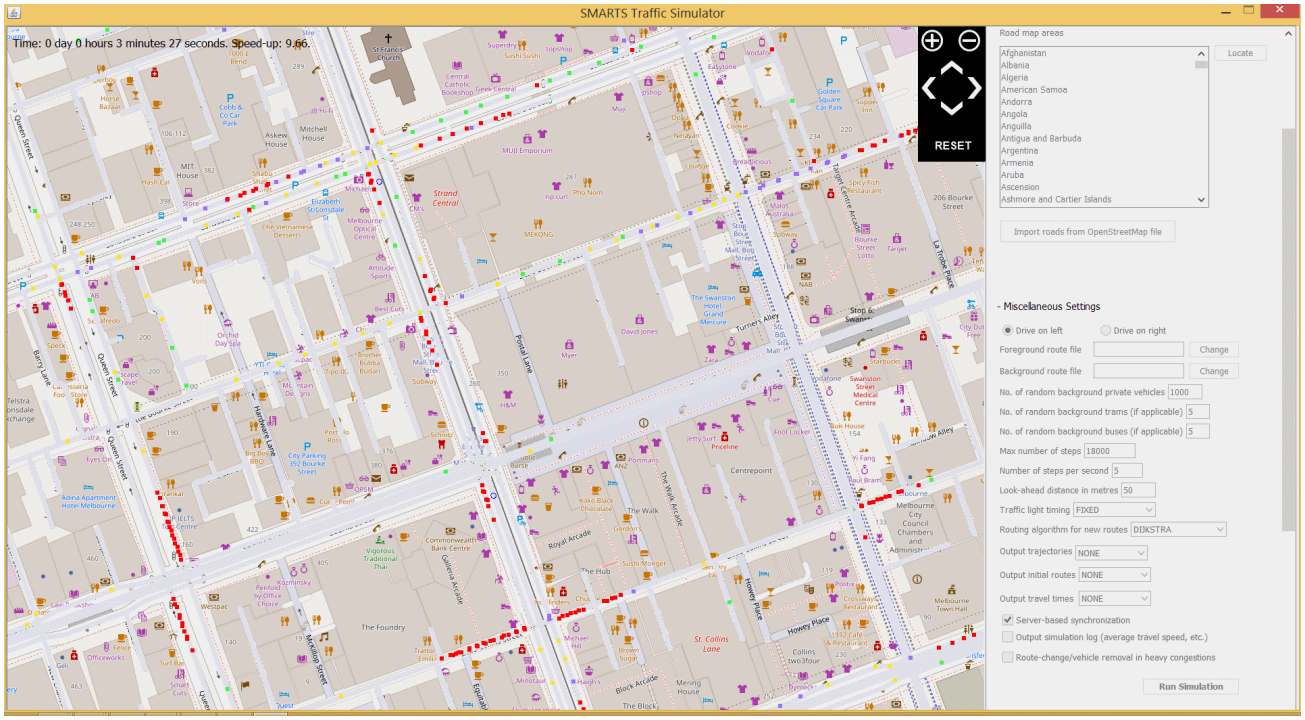


Figure 1: Simulating road traffic in Melbourne with SMARTS.

at the same time. It also supports scripted simulations. Figure 1 shows a screenshot of the simulator's GUI. The simulated vehicles are drawn as little squares on roads. Red squares represent vehicles in congestions or waiting at traffic lights. Green squares represent vehicles moving at free-flow speed. Squares of other colours represent vehicles moving at intermediate speeds. The map image from OpenStreetMap is shown in the background.

The project website of SMARTS, `https://projects.eng.unimelb.edu.au/smarts`, provides documentation, examples and the binary versions of the simulator.

2 Traffic Data Generation with SMARTS

SMARTS can generate three types of traffic data. The first data type is route plans. A route plan contains the waypoints that the vehicle needs to pass through during its trip. The waypoints are a subset of the road network nodes, which are extracted from OpenStreetMap data. The route plan also contains other details about the vehicle, such as the starting time of the trip, the aggressiveness level of the vehicle driver, etc. A route plan can be imported into future simulations, which will replicate the traffic as specified in the plan. The second data type is timestamped trajectories. The trajectory of a vehicle contains a list of records, each of which includes a timestamp, a latitude value and a longitude value. The third data type is travel times. A travel time data file contains a number of records, each of which shows a vehicle ID and the total time that the vehicle spends on its trip. When configuring a simulation, users can specify the types of the data that needs to be generated from the simulations. SMARTS saves the data to disk when the simulation is completed.

2.1 Foreground Traffic vs. Background Traffic

For convenience of users, SMARTS divides the vehicles into two groups, *foreground vehicles* and *background vehicles*. The output data can be collected from foreground vehicles, background vehicles or all the vehicles. For example, a user may ask the simulator to output the trajectories of foreground vehicles, the route plans

of background vehicles and the travel times of all the vehicles. The foreground vehicles can be defined by importing a route-plan file at simulation initialization. The background vehicles can be defined in the same way but can also be randomly generated during a simulation. The differentiation between foreground vehicles and background vehicles can be useful in many circumstances. For example, assuming we want to study the impact of morning peak-hour traffic on individual vehicles, we can define several foreground vehicles that travel across a city and ask SMARTS to output the trajectories of these vehicles. During the simulation, the simulator can generate a large number of random background vehicles that travel towards the city centre without outputting data from those vehicles.

2.2 Data Generation Workflow

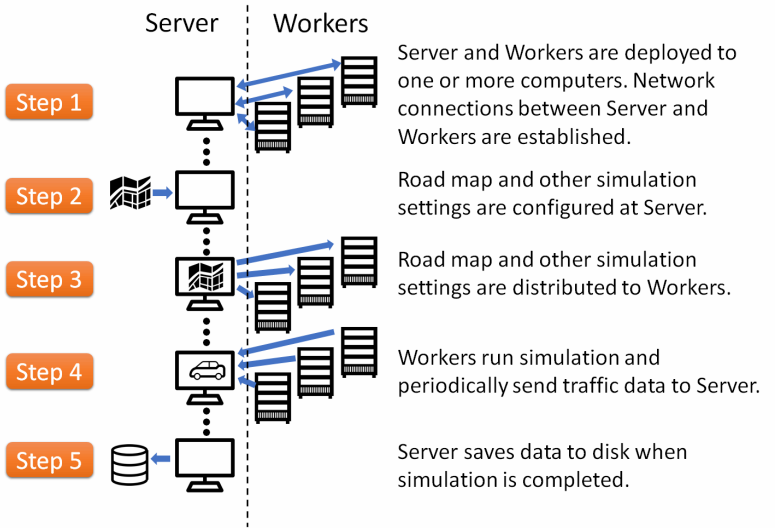


Figure 2: The workflow of data generation.

The workflow for generating traffic data from a simulation is illustrated in Figure 2. SMARTS runs with one server and one or more workers. In Step 1, the server and the workers are deployed to one or more computers. More details about the deployment of SMARTS are provided in Section 3.2. The simulation is configured in Step 2. During this step, users can specify the types of the data that needs to be generated from the simulation. In Step 3, the server partitions the simulation area into one or more sub-areas. The number of the sub-areas equals to the number of the workers. The server distributes the road map to the workers and sends other simulation configurations to the workers. For example, the server tells a specific worker the number of random vehicles that the worker should maintain in its simulation area. After receiving the simulation configuration details, workers will start to run the simulation. As shown in Step 4, workers periodically sends vehicle positions or other traffic information to the server during a simulation. The information can be used for simulation visualization. The server will also aggregate the information if traffic data needs to be generated from the simulation. Once the simulation is completed, the server saves the traffic data to disk, as shown in Step 5.

3 Background

3.1 SMARTS vs. Other Simulators

SMARTS has a more comprehensive feature set than existing traffic simulators shown in Table 1. For example, with its distributed computing capability, SMARTS can perform fast large-scale simulations on a cluster of com-

Table 1: Comparison of traffic simulators: SUMO [6], TRANSIMS [7], VISSIM [10], MATSIM [14], ParamGrid [5], and SMARTS. To the best of our knowledge, ParamGrid has not been maintained in recent years.

	SUMO	TRANSIMS	VISSIM	MATSIM	ParamGrid	SMARTS
Distributed computing	×	√	×	×	√	√
Map feature generation	√	×	×	×	×	√
Continuous spatial automaton	√	×	√	√	√	√
Workload balancing	×	√	×	×	√	√
Decentralized synchronization	×	×	×	×	√	√
Visualized control	√	√	√	√	×	√

puters. SMARTS implements a *decentralized synchronization* technique, which allows distributed processors to compute without waiting for instructions from the central server during a simulation. This can make distributed simulations run even faster as it eliminates a potential communication bottleneck in the system. SMARTS can generate certain map features based on simple map data. For example, it can find the position of a tram stop on the road that is parallel to a tram track.

3.2 Deployment of SMARTS

We provide two binary versions, a *standalone* version and a *distributed* version. All the versions are made as *Java ARchive (JAR)* files. Users can launch the JAR files on any computer that has Java Runtime Environment (JRE) installed. SMARTS has been tested with JRE 8.

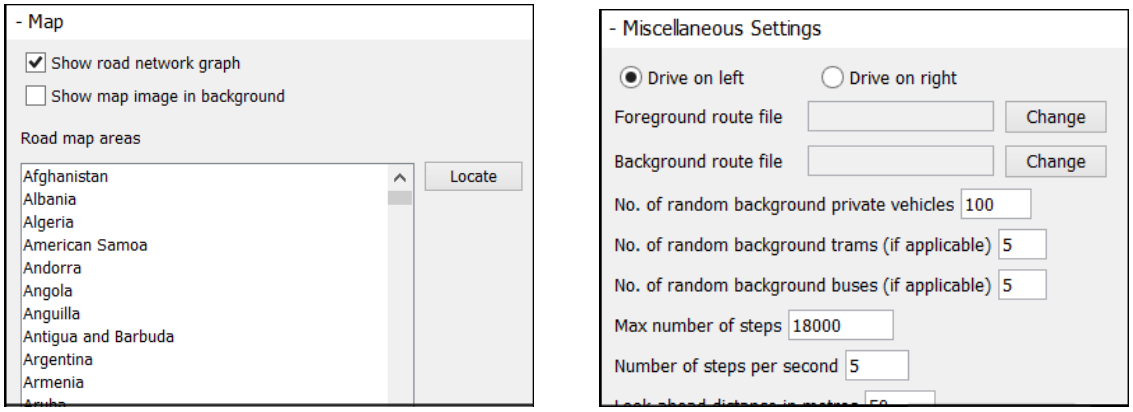
The standalone version bundles a *server* component and a *worker* component into one JAR file. When the standalone version is launched, a server instance is created based on the server component and a worker instance is created based on the worker component. The server receives simulation configuration from users, visualizes simulations, collects simulation data from workers and generates result files. The workers are responsible for simulating traffic and transferring simulation data to the server. Users can create more workers by changing a setting on the GUI. The server and all the workers run within the same Java Virtual Machine. The standalone version is suitable for small-scale simulations, such as simulating thousands of vehicles in a road network of hundreds of road links.

For large-scale simulations, such as simulating hundreds of thousands of vehicles in a large city, one may need to use the distributed version, which includes two JAR files, one for creating server instances and another for creating worker instances. When using the distributed version, the server JAR file and the copies of the worker JAR file can be deployed to different computers. The server should be launched before the workers are launched. The workers need to be explicitly started with a command line argument, which includes the IP address of the server. For example, suppose a distributed simulation is deployed with four computers. One of the computers, whose IP address is *192.1.1.1*, hosts the server. The server JAR file is copied to this computer and launched there. The worker JAR file is then copied to each of the remaining three computers, where we run a command, *java -jar worker.jar 192.1.1.1*, which starts a worker instance and connects it to the server instance that runs on the computer at IP address *192.1.1.1*.

3.3 Configuring Simulations with SMARTS

Simulations can be configured in two ways. If the GUI was enabled, users can configure simulations using the input fields on the GUI. If the GUI was disabled, users need to prepare script files that contain the parameters of the simulations. As the GUI is built into the server component, it is enabled

by default in the standalone version of SMARTS. For the distributed version, the GUI is enabled when the server package is launched without any command line argument. An example of running a distributed simulation with GUI can be found at `https://projects.eng.unimelb.edu.au/smarts/example-4-running-a-distributed-simulation-with-gui/`. The GUI allows users to con-



(a) A part of the map control panel

(b) A part of the miscellaneous settings panel

Figure 3: Graphical user interface for configuring simulation settings.

figure a range of settings. Users can control the amount of computing resources for simulations by specifying the number of workers. If the users were using the standalone version, the system will automatically create worker instances when the users change the number of workers. Using multiple workers in the standalone version may still accelerate simulations to a certain extent, depending on the number of threads supported by the processor and the scale of the simulations. If the users were using the distributed version, the worker instances will need to be explicitly launched by the users. As shown in Figure 3a, the GUI provides shortcuts to many regions in the world. Users can change the map using the shortcuts, manipulate the map view, define a rectangular area in the map view and download OpenStreetMap data for the area. An example of the map-downloading feature can be found at `https://projects.eng.unimelb.edu.au/smarts/example-1-simulating-traffic-in-an-arbitrary-area-in-the-world/`. Users can also change the map by importing an OpenStreetMap data file. Many other parameters, such as the number of vehicles, can also be configured on the GUI as shown in Figure 3b.

The GUI should be disabled if users needed to run distributed simulations based on scripts. To run a scripted distributed simulation, users need to first launch the server JAR file with a command line argument, *-gui false*. An example of running a scripted simulation can be found at `https://projects.eng.unimelb.edu.au/smarts/example-5-running-a-distributed-simulation-based-on-a-script/`. The server program will display command line instructions to guide users through the configuration process. The path to a script file needs to be entered during the configuration process. The script file can specify a number of parameters for the simulations. A simulation can be repeated a specified number of times based on a parameter setting in the script. Users can use one script to configure a sequence of different simulations.

3.4 Online Resources

The binary files of the simulator can be downloaded from `https://projects.eng.unimelb.edu.au/smarts/downloads`. To help users get familiar with the simulator, we provide simple examples on `https://projects.eng.unimelb.edu.au/smarts/examples`. The examples give step-by-step instruction on setting up a simulation environment, downloading maps and generating route plans and trajectories. Technical documentation can be found on `https://projects.eng.unimelb.edu.au/smarts/`

`documentation`. The documents include the descriptions of different versions of SMARTS. They also include the specifications of simulation script and route plan files.

4 Example Applications

SMARTS has been used intensively in many research projects on transportation-related topics. Some of the research projects are described as follows.

4.1 Routing Algorithm Evaluation

We use SMARTS to evaluate and present a new routing algorithm, *Random A** [8]. For this project, we adjust the GUI to highlight the routes generated by the algorithm. For evaluating the performance of the new algorithm, we create various traffic scenarios within SMARTS. For example, one of the scenarios creates traffic that starts from a number of areas outside Melbourne's central area and ends in the city centre. We compare the new algorithm and a classic routing algorithm based on the vehicle travel times collected from the simulator.

4.2 Vehicle Prioritization Evaluation

SMARTS is used to compare different prioritization strategies for emergency vehicles in an intelligent transportation system, where vehicles are highly connected with each other [16]. First, we extend SMARTS by implementing traffic light pre-emption for emergency vehicles. The system adjusts traffic lights for emergency vehicles such that emergency vehicles always get green light until they reach their destinations. Then, we implement three prioritization strategies, which can control the behaviour of non-priority vehicles when they are informed by the system that emergency vehicles are approaching. We collect the travel times of emergency vehicles for a wide range of traffic scenarios by varying prioritization strategies, road maps, traffic volumes, clearance distances of emergency vehicles and travel distances of emergency vehicles.

4.3 Simulation-Based Navigation

We build a prototype navigation service based on traffic simulations [1]. The service optimizes route advices by considering future traffic conditions. For example, a suggested route can avoid an area, which is not congested now but may become congested later as predicted by a simulation. The prototype service uses SMARTS for traffic prediction as the simulator can run faster than real time.

4.4 Traffic Optimization Evaluation

SMARTS is used for evaluating traffic optimization with autonomous vehicle platooning [12]. We are interested in the situations, where a group of adjacent autonomous vehicles that move in the same direction can form a platoon. All the vehicles in the same platoon can accelerate and decelerate at the same time. We use SMARTS to compare two scenarios, one is with autonomous vehicle platooning, another is without the platooning.

5 Conclusion

We introduced SMARTS for its features and its applications. It is straightforward to run simulations based on real road networks and generate artificial traffic data within SMARTS. The simulator can generate traffic data based on a subset of the simulated vehicles. This capability can be useful for studying specific vehicles in large-scale simulations as well as creating large city scale traffic data. The simulator has the flexibility to suit various

computing environments and simulation needs. We hope SMARTS can become a useful tool for the research community.

References

[1] A. AlDwyish, H. Xie, E. Tanin, S. Karunasekera, and K. Ramamohanarao. Using a traffic simulator for navigation service. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 78. ACM, 2017.

[2] F. Bastani, Y. Huang, X. Xie, and J. W. Powell. A greener transportation mode: flexible routes discovery from GPS trajectory data. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 405–408. ACM, 2011.

[3] T. Brinkhoff. A framework for generating network-based moving objects. *Geoinformatica*, 6(2):153–180, 2002.

[4] C. Düntgen, T. Behr, and R. H. Güting. BerlinMOD: a benchmark for moving object databases. *The VLDB Journal*, 18(6):1335–1368, 2009.

[5] R. Klefstad, Y. Zhang, M. Lai, R. Jayakrishnan, and R. Lavanya. A distributed, scalable, and synchronized framework for large-scale microscopic traffic simulation. In *Intelligent Transportation Systems*, pages 813–818. IEEE, 2005.

[6] D. Krajzewicz, G. Hertkorn, C. Rössel, and P. Wagner. SUMO (Simulation of Urban MObility) – an open–source traffic simulation. In *Middle East Symposium on Simulation and Modelling*, pages 183–187. DLR, 2002.

[7] K. Nagel and M. Rickert. Parallel implementation of the TRANSIMS micro-simulation. *Parallel Computing*, 27(12):1611–1639, 2001.

[8] U. T. Nguyen, S. Karunasekera, L. Kulik, E. Tanin, R. Zhang, H. Zhang, H. Xie, and K. Ramamohanarao. A randomized path routing algorithm for decentralized route allocation in transportation networks. In *Proceedings of the 8th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, pages 15–20. ACM, 2015.

[9] OpenStreetMap Foundation. OpenStreetMap. `http://www.openstreetmap.org`, 2018. Accessed: 2018-11-21.

[10] PTV Group. PTV VISSIM. `http://vision-traffic.ptvgroup.com/en-uk/products/ptv-vissim`, 2018. Accessed: 2018-11-21.

[11] C. Quek, M. Pasquier, and B. B. S. Lim. POP-TRAFFIC: A novel fuzzy neural approach to road traffic analysis and prediction. *IEEE transactions on intelligent transportation systems*, 7(2):133–146, 2006.

[12] K. Ramamohanarao, J. Qi, E. Tanin, and S. Motallebi. From how to where: Traffic optimization in the era of automated vehicles. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 10. ACM, 2017.

[13] K. Ramamohanarao, H. Xie, L. Kulik, S. Karunasekera, E. Tanin, R. Zhang, and E. B. Khunayn. SMARTS: Scalable Microscopic Adaptive Road Traffic Simulator. *ACM Transactions on Intelligent Systems and Technology*, 8(2):26:1–26:22, 2017.

[14] R. A. Waraich, D. Charypar, M. Balmer, and K. W. Axhausen. Performance improvements for large-scale traffic simulation in MATSim. In *Computational Approaches for Urban Environments*, pages 211–233. Springer, 2015.

[15] L. Xia and Y. Shao. Modelling of traffic flow and air pollution emission with application to Hong Kong Island. *Environmental Modelling & Software*, 20(9):1175–1188, 2005.

[16] H. Xie, S. Karunasekera, L. Kulik, E. Tanin, R. Zhang, and K. Ramamohanarao. A simulation study of emergency vehicle prioritization in Intelligent Transportation Systems. In *85th Vehicular Technology Conference (VTC Spring)*, pages 1–5, June 2017.

The SIGSPATIAL Special

Section 2: Event Reports

ACM SIGSPATIAL
http://www.sigspatial.org

Conference Report: The 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL 2018) Seattle, Washington, USA November 6—9, 2018

Ralf Hartmut Güting
FernUniversität in Hagen, Germany
rhg@fernuni-hagen.de

Roberto Tamassia
Brown University, USA
rt@cs.brown.edu

Li Xiong
Emory University, USA
lxiong@emory.edu

Farnoush Banaei-Kashani
University of Colorado, USA
farnoush.banaei-kashani@ucdenver.edu

Erik Hoel
Esri, USA
ehoel@esri.com

This report describes the development and finalization of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL 2018), held in Seattle, Washington, USA, November 6-9, 2018. The attendance for the 2018 conference was 414 (representing 29 countries), the first such event with more than 400 attendees. The conference was also notable for having 148 of the attendees from industry, exceeding the previous high of 100 attendees in 2015 (also in Seattle).

Historically, what is now the ACM SIGSPATIAL conference started as a series of workshops and symposia in 1993. Its aim was to promote the interdisciplinary discussions among researchers, developers, users, and practitioners and fostering research in all aspects of Geographic Information Systems – hence the original workshop acronym ACM GIS. The focus was on novel systems based on geospatial data and knowledge. It continued its mission of providing a forum for original research results, addressing conceptual, design, and implementation aspects of geospatial data ranging from applications, user interfaces and visualization, to data storage, query processing, indexing and data mining. The conference is now the premier annual event of the ACM Special Interest Group on Spatial Information (ACM SIGSPATIAL).

The technical program of the conference was decided in a two-stage process:

(1) Each submitted paper was first reviewed by at least three members of a carefully chosen program committee (PC) consisting of experts in the relevant fields. Our PC had a total of 112 volunteers from academia and industry, plus an additional 21 members who were designated as the Senior PC. The assignment of papers to reviewers followed a bidding stage, during which PC members were allowed to express ranked preferences regarding their willingness to review a particular submission. In addition to three reviewers from the PC, each paper was also assigned a designated Senior PC member who studied the reviews, discussed the merits of the submission with the reviewers, wrote a metareview, and formulated an accept/reject recommendation.

(2) Similar to 2017, we implemented a rebuttal phase where the authors received preliminary versions of the reviews and the meta-review, and were offered the opportunity to address the concerns raised therein by submitting a response. The reviews, meta-reviews, and accept/reject recommendations were then finalized, taking

into account the author responses. The selection of papers to include in the conference program was unltimately made by the PC Chairs. Certain papers that were not accepted for the conference, with the permission of the authors, were forwarded to the conferences Workshop Chairs to be considered for inclusion in relevant workshops co-located with SIGSPATIAL.

Papers were submitted and accepted in different categories. We received a total of 150 research submissions and 19 industrial experience and systems submissions. We accepted 38 of those as full 10-page research papers for oral presentation, resulting in an acceptance rate of 22.5%. We accepted an additional 40 submissions as poster presentations (38.2% cumulative acceptance rate), to be published as 4-page papers. We also received 24 demonstration submissions, of which we accepted 14 for live demonstrations, to be published as 4-page papers (acceptance rate of 58%). Finally, once again we encouraged the submission of papers describing visionary ideas. Of the 12 vision papers submitted, 4 were accepted for oral presentation (33.3% acceptance rate) and publication as 4-page papers. Our reviewers put in a significant amount of effort in reviewing the papers and our hope is that the reviews were beneficial even to those authors whose papers were not accepted.

Continuing the tradition, ACM SIGSPATIAL 2018 had a Cup programming contest, which focused on analyzing large spatial networks taken from the utility domain in order to find upstream features from a given set of starting points. The competition received 15 submissions and the teams totaled 42 members submitting formal entries. Three entries were selected as winners, and were additionally qualified for an invited paper, an oral presentation and award prizes during the banquet.

For the third time, after its debut in 2016, the conference had a Student Research Competition that aimed at providing a forum for undergraduate and graduate students to share their research results and exchange ideas with other students, judges, and conference attendees. This year, 7 papers (co)authored by graduate students were selected to enter the final round of the competition.

ACM SIGSPATIAL 2018 had two invited talks: Xin Chen (HERE Technologies) with a keynote presentation *HD Live Maps for Automated Driving: An AI Approach*, and Daniel Delling (Apple Maps) whose keynote addressed *Route Planning in Transportation from Research to Practice*.

The conference was expertly chaired by Erik Hoel (Esri, USA) and Farnoush Banaei-Kashani (University of Colorado, Denver, USA). It was preceded by 13 associated workshops managed by the Workshop Co-Chairs were John Krum (Microsoft, USA) and Mohamed Sarwat (Arizona State University, USA), in addition to the many respective individual workshops organizers.

Recreating and organizing such a vibrant forum from year to year takes tremendous efforts and collaboration among many dedicated individual. We are especially grateful to our PC, Senior PC and external reviewers, who generously and carefully reviewed the submissions and produced valuable feedback for both us and the authors. To produce the proceedings, we had the pleasure to work closely with two individuals who took a lot of the burden and did a great job Proceedings Co-Chairs Gabriel Ghinita (University of Massachusetts, USA) and Raymond Wong (Hong Kong University of Science and Technology, Hong Kong). We thank Chrysovalantis Anastasiou (University of Southern California, USA) and Xu Teng (Iowa State University, USA) who were extremely responsive as our Webmasters. We are also very thankful to the Publicity Co-Chairs: Muhammad Cheema (Monash Universiry, Australia), Sangho Kim (Esri, USA), and Yanhua Li (Worcester Polytechnic Institute, USA). Furthermore we thank Martin Werner (Leibniz-University Hanover, Germany) and Xun Zhou (University of Iowa, USA) who served as Poster Co-Chairs, and extend our special thanks to Dev Oliver (Esri, USA), Bo Xu (HERE, USA), and Yuanyuan Pao (Lyft, USA) who organized the SIGSPATIAL Cup programming contest this year.

Many other fine individuals were involved and did a great job for the technical organization of the event and were in charge of many related activities. We thank Wei-Shinn Ku (Auburn University, USA) and Amr Magdy (University of California, Riverside, USA) who served as Treasurer Co-Chairs, along with our special thanks to Kyriakos Mouratidis)Singapore Management University) and Fusheng Wang (Stony Brook University, USA) who were in charge of organizing the SRC (Student Research Competition). We are indebted to Jing (David) Dai (Google, USA) and Zhenhui Li (Penn State University, USA) who served as Registrations Co-

Chairs. Local Arrangements Co-Chairs James Biagioni (CARMERA, USA), Ahmed Eldawy (University of California, Riverside, USA), and Hien To (Amazon, USA) are the ones to whom we are all indebted for the diligence and hard work that they put in to ensure that everything ran smoothly at the venue.

We are also thankful to the ACM SIGSPATIAL Executive Committee for their expert, sustaining guidance of the conference: Cyrus Shahabi (Chair, University of Southern California, USA), Goce Trajcevski (Vice-Chair, Iowa State University, USA), Egemen Tanin (Secretary, University of Melbourne, Australia), and John Krum (Treasurer, Microsoft Research, USA).

Very special thanks and recognitions are in order for our very generous corporate sponsors who this year contributed more than $100,000 in sponsorship Amazon, Apple, and HERE (Platinum Sponsors), Esri, Lyft, and Uber (Silver Sponsors), Oracle (Bronze Sponsor), and Microsoft and IBM many of whom have supported this conference for multiple years. We also recognize and appreciation the work of Mark McKenney (Southern Illinois University Edwardsville, USA) and Chengyang Zhang (Amazon, USA) who were instrumental in getting these companies as our sponsors. We are also grateful for the publishing sponsorship by Springer Publishers (Bronze Publisher Sponsor).

A very distinct token of gratitude goes to the US National Science Foundation (NSF) for its institutional sponsorship enabling travel-grants for students. Additionally, we appreciate IBM and Esri who provided financial sponsorship for the SIGSPATIAL Cup and the awards for the winners. Last, but not the least, the top three papers out of the X accepted vision papers received an award from the Computing Research Associations Computing Community Consortium (CCC).

Every year, the conference highlights the most important advances in GIS and provides a forum for lively exchange of ideas among leading researchers and practitioners in the field. We are confident that you will find a similar value in this record of the conference. In conclusion, we would like to express once again our gratitude to all the authors who submitted papers, the members of the PC and the senior PC, the conference officers, and all the other individuals who contributed their expertise and time to make the conference possible.

ACM SIGSPATIAL Cup 2018 - Identifying Upstream Features in Large Spatial Networks

Dev Oliver[1], Bo Xu[2], Yuanyuan Pao[3]
[1]Environmental Systems Research Institute (Esri)
[2]HERE
[3]Lyft
doliver@esri.com, bo.5.xu@here.com, ypao@lyft.com

Abstract

ACM SIGSPATIAL Cup 2018 was the 7th GIS-focused algorithm contest hosted by the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL 2018). The contest explored the problem of analyzing large spatial networks (e.g., utility networks) in order to find upstream *features from a given set of starting points (a feature is considered to be upstream if it is on a simple path between a starting point and a controller).*

1 Introduction

For over 26 years the ACM SIGSPATIAL conference has addressed a variety of topics in the field of geographic information systems. In addition to presenting scientific discoveries in the form of papers, short papers, posters, and demos, the conference began acknowledging the art of algorithm design and implementation from a practical perspective through the ACM SIGSPATIAL Cup. Each year, a challenging yet well-researched computational topic is chosen and participants are required to solve the given problem with high accuracy, quality, and performance.

This year's contest focused on analyzing large spatial networks in order to find upstream features from a given set of starting points. The identification of upstream features is a key problem in critical infrastructure analysis for locating assets that are crucial to a nation's economy, security, and health. An example is identifying transformers that supply electricity to a water pump that is needed by the sole hospital in a region. Given a spatial network, a set of starting points (which may be junctions or edges in the spatial network), and a set of controllers (which are also junctions in the spatial network), the goal is to find all features originating at starting points and terminating at controllers where each "upstream" feature returned is on a simple path between a starting point and a controller.

The type of spatial networks that the contest focused on were utility networks [5, 3, 1, 6], which are used to model utility systems such as electric, gas, water, and telecommunications. Utility networks define and manage a collection of database tables, metadata, business rules, and a network graph, used to store and model the connectivity information. They are used to represent the relations of the real-world objects that utility companies use to deliver resources to their customers. A utility network is subdivided into a set of zones or circuits. Each zone or circuit has a special set of junctions called controllers from which resources flow. The contest operated under the assumption that the zones or circuits are source-based (e.g., electric, water, and gas networks), meaning that resources will flow from controllers or sources to the rest of a zone or circuit (note

that zones can also be sink-based such as in gravity-fed networks, e.g., sewer networks, where resources flow towards the controllers or sinks). The main tasks in this contest were to build the network topology and to infer flow direction in a source-based network such that features that are upstream to starting points can be identified. The term "starting point" defines the location where analysis begins, i.e., the feature for which we are trying to identify upstream features. In looped topologies, flow direction is assumed to be bidirectional.

2 Sponsors and Supporters

ACM SIGSPATIAL Cup 2018 was sponsored by IBM and Esri, which was a great help for the SIGSPATIAL community. The sponsorship resulted in cash awards of $500 for first place, $300 for second place, and $200 for third place. Additionally, the authors of the best three submissions were invited to write a short paper about the key ideas of their approaches and to give an oral presentation of their work in a conference session dedicated to the cup.

3 Problem Description

The problem that was explored in this contest is defined as follows:

Given:

- A collection of point features representing devices and junctions
- A collection of line features
- A set of controllers, which are a subset of the device features
- A set of starting points, S, where S is a subset of the device, junction, and line features

Find:

- All upstream features (a feature is considered to be upstream if it is on a simple path between a starting point and a controller).

Objective:

- Computational efficiency

3.1 Example

Figure 1 shows an example input and output of the problem. The input consists of a collection of point features, represented by junctions $J1$, $J2$, $J3$, $J4$, $J5$, $J6$, $J7$, $J8$, $J9$, $J10$, $J11$, $J12$, and $J13$, a collection of line features represented by edges $E1$, $E2$, $E3$, $E4$, $E5$, $E6$, $E7$, $E8$, $E9$, $E10$, $E11$, $E12$, $E13$, and $E14$, two controllers $J1$ and $J5$, and one starting point $J9$. The output is all upstream features which are highlighted (i.e., junctions $J1$, $J2$, $J3$, $J5$, $J6$, $J7$, $J8$, and $J9$ and edges $E1$, $E2$, $E4$, $E5$, $E6$, $E7$, $E8$, $E9$). The edges and junctions that were not included in the results were not on a simple path between a starting point and a controller.

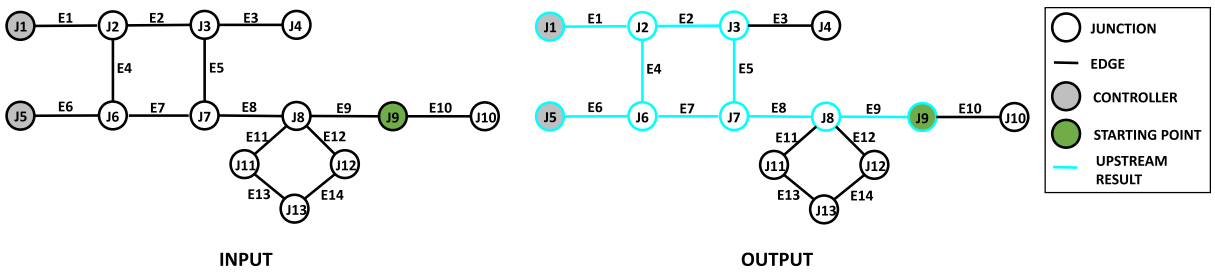


Figure 1: Example Input and Output of Identifying Upstream Features

3.2 Data

Two datasets were provided for this challenge. The first was a sample dataset comprised of 30 features, which was useful for participants to verify the correctness of their approaches. The second was Esri's Naperville Electric Network dataset (comprised of approximately 15,000 features). Figure 2 shows an example of identifying upstream features in Esri's Naperville electric dataset. The starting point is represented by the green circle towards the north western portion of the image and the controller is located towards the south western portion of the image. The features that are upstream are highlighted in blue.

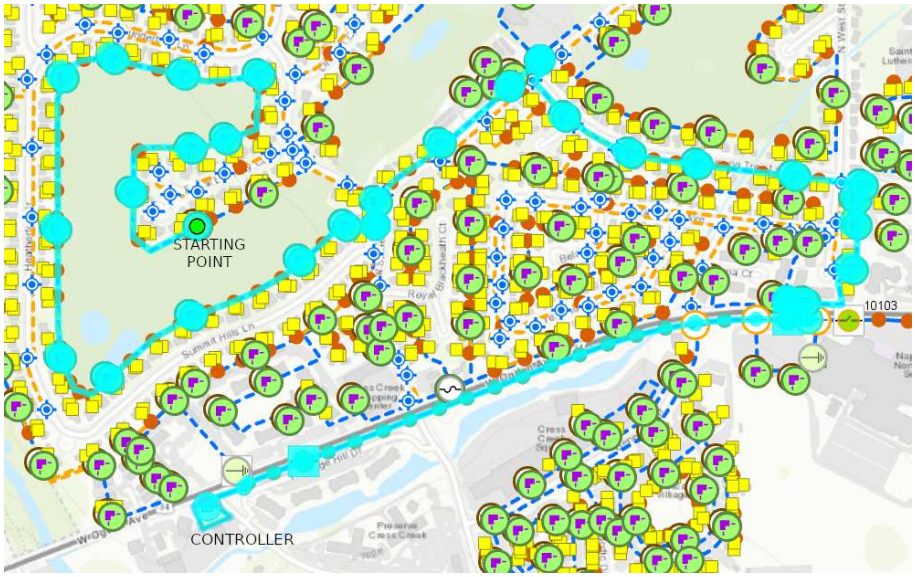


Figure 2: Example of identifying upstream features in Esri's Naperville Electric Dataset

Each dataset contained a json file for constructing the network topology, a set of shapefiles for visualization, a sample starting points file, and a sample output file. Submissions were required to consume the json file to construct the network topology and extract the controller information.

4 Submissions and Winners

The challenge received 15 very good submissions and the teams totaled 42 members from all over the world. Three entries were selected as winners. The best three submissions were written in C++. They all leveraged a

block-cut tree for identifying upstream features from a set of starting points in the network.

The winning submission was by Salles Viana Gomes Magalhães, W. Randolph Franklin, and Ricardo dos Santos Ferreira [4]. Their submission computed articulation points and biconnected components using Tarjan's algorithm [7]. However, instead of reusing existing implementations of Tarjan's algorithm (such as the one provided by the Boost Graph Library), they implemented it from the ground up for performance purposes. Their implementation that was parallelized using OpenMP also used two 64-bit integers to represent the 32-digit (where each digit is in hexadecimal) global identifier for features (to reduce memory footprint and accelerate operations like comparing the ids of two features) and a custom parser to read the JSON files representing the network. The second place submission was by Thomas C. van Dijk, Tobias Greiner, Bas den Heijer, Nadja Henning, Felix Klesen, and Andre Löffler [8]. Their approach used a linear-time algorithm based on an annotated block-cut tree. The third place submission was by Zach Goldthorpe, Jason Cannon, Jesse Farebrother, Zachary Friggstad, and Mario A. Nascimento [2]. They implemented a linear time algorithm based primarily on a two-sweep depth-first search which decomposes a graph into its biconnected components prior to collecting the upstream features.

Acknowledgments

We wish to acknowledge the support from our sponsors IBM and Esri. Furthermore, we want to express our thanks to the organizing committee of the ACM SIGSPATIAL Conference on Advances in Geographic Information Systems for hosting this competition.

References

[1] P. Bakalov, E. G. Hoel, and S. Kim. A network model for the utility domain. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 32. ACM, 2017.

[2] Z. Goldthorpe, J. Cannon, J. Farebrother, Z. Friggstad, and M. A. Nascimento. Using biconnected components for efficient identification of upstream features in large spatial networks. In *Proceedings of the 26th SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2018.

[3] E. Hoel, P. Bakalov, S. Kim, and T. Brown. Moving beyond transportation: utility network management. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 8. ACM, 2015.

[4] S. V. G. Magalhães, R. W. Franklin, and R. d. S. Ferreira. Fast analysis of upstream features on spatial networks. In *Proceedings of the 26th SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2018.

[5] B. Meehan. *Modeling electric distribution with GIS*. Esri Press Redlands, 2013.

[6] D. Oliver and E. G. Hoel. A trace framework for analyzing utility networks: A summary of results. In *Proceedings of the 26th SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2018.

[7] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[8] T. C. van Dijk, T. Greiner, B. den Heijer, N. Henning, F. Klesen, and A. Löffler. Wüpstream: Efficient enumeration of upstream features. In *Proceedings of the 26th SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2018.

Spatial Gems: A New Type of Workshop at the SIGSPATIAL Conference

John Krumm
Microsoft Research
jckrumm@microsoft.com

Cyrus Shahabi
University of Southern California
shahabi@usc.edu

Andreas Züfle
George Mason University
azufle@gmu.edu

As researchers, we sometimes develop new approaches for solving spatial problems that fall between a textbook chapter and a research paper. These are fundamental techniques that would be useful to both researchers and practitioners. Until now, there has not been a good forum to disseminate these "spatial gems".

We are hosting a new workshop at the 2019 ACM SIGSPATIAL conference called "1st ACM SIGSPATIAL International Workshop on Spatial Gems (SpatialGems 2019" where we will collaboratively document and publish these techniques to benefit our field. We expect these contributions will be frequently read and referenced, so the workshop will focus on ensuring the quality and clarity of the papers. Instead of a sequence of presentations, workshop participants will primarily work together to edit each others papers. Were putting the "work" back in workshop.

Spatial Gems is modeled after the successful Graphics Gems book series. Spatial gems are not research papers. Instead, they are fundamental solutions for spatial processing that are likely to become part of a larger approach. While a gem may have already been published as a small part of a paper, extracting it into a gem makes it much more likely to be found and used by others. Examples of spatial gems include:

- Converting latitude/longitude coordinates into a locally Euclidean coordinate system
- Computing the mean and variance of speed from two noisy location measurements
- Tessellating the earth in a convenient, useful way
- Simplifying a latitude/longitude polygon while preserving its perimeter and area
- Matching two trajectories with dynamic time warping
- An R-Tree implementation in Python
- Random spatial point data generators including uniform, normal, and clustered

Each gem will be two to four pages long. Where appropriate, a good gem will include numerical examples so programmers can verify their implementations. The goal of a spatial gem is to convey a fundamental solution technique. A spatial gem is not a research paper with data and experiments. Spatial gems should be self-contained, not a pointer to code nor data, and not a summary of a research article. It is appropriate for a spatial gem to elaborate and focus on a technique that has already been published as part of a research paper as long as the gem contains proper attribution.

The result of the workshop will be a public archive of spatial gems. If the workshop continues in future years, we have the opportunity to produce a volume of accumulated gems. Because a large part of the workshop will be devoted to collaboratively editing the submissions, we will require all papers to be submitted as a shared LaTeX project on Overleaf (www.overleaf.com). At the workshop, we will have a sequence of editing sessions where authors will be paired with other authors to edit each others submissions. The goal of these sessions is to iterate toward a high-quality final paper. The workshops co-organizers will review the submitted papers and participate in the editing sessions.

There are more details about the workshop at `www.spatialgems.net`. Submissions are due on 16 August 2019. We invite you to participate. If you have an idea for a spatial gem, please feel free to contact us to discuss whether or not it would be appropriate for the workshop.

join today!

SIGSPATIAL & ACM

www.sigspatial.org **www.acm.org**

The **ACM Special Interest Group on Spatial Information** (SIGSPATIAL) addresses issues related to the acquisition, management, and processing of spatially-related information with a focus on algorithmic, geometric, and visual considerations. The scope includes, but is not limited to, geographic information systems (GIS).

The **Association for Computing Machinery** (ACM) is an educational and scientific computing society which works to advance computing as a science and a profession. Benefits include subscriptions to *Communications of the ACM*, *MemberNet*, *TechNews* and *CareerNews*, full and unlimited access to online courses and books, discounts on conferences and the option to subscribe to the ACM Digital Library.

- ❑ SIGSPATIAL (ACM Member) ..... $ 15
- ❑ SIGSPATIAL (ACM Student Member & Non-ACM Student Member) ..... $ 6
- ❑ SIGSPATIAL (Non-ACM Member) ..... $ 15
- ❑ ACM Professional Membership ($99) & SIGSPATIAL ($15) $114
- ❑ ACM Professional Membership ($99) & SIGSPATIAL ($15) & ACM Digital Library ($99) $213
- ❑ ACM Student Membership ($19) & SIGSPATIAL ($6) ..... $ 25

payment information

Name _____

ACM Member # _____

Mailing Address _____

_____

City/State/Province _____

ZIP/Postal Code/Country _____

Email _____

Mobile Phone _____

Fax _____

Mailing List Restriction
ACM occasionally makes its mailing list available to computer-related organizations, educational institutions and sister societies. All email addresses remain strictly confidential. Check one of the following if you wish to restrict the use of your name:

- ❑ ACM announcements only
- ❑ ACM and other sister society announcements
- ❑ ACM subscription and renewal notices only

Credit Card Type: ❑ AMEX ❑ VISA ❑ MC

Credit Card # _____

Exp. Date _____

Signature _____

Make check or money order payable to ACM, Inc

ACM accepts U.S. dollars or equivalent in foreign currency. Prices include surface delivery charge. Expedited Air Service, which is a partial air freight delivery service, is available outside North America. Contact ACM for more information.

Questions? Contact:
ACM Headquarters
2 Penn Plaza, Suite 701
New York, NY 10121-0701
voice: 212-626-0500
fax: 212-944-1318
email: acmhelp@acm.org

Remit to:
ACM
General Post Office
P.O. Box 30777
New York, NY 10087-0777

SIGAPP

www.acm.org/joinsigs

Association for Computing Machinery

Advancing Computing as a Science & Profession

The SIGSPATIAL Special

ACM SIGSPATIAL
http://www.sigspatial.org