# Cohesive Group Nearest Neighbor Queries on Road-Social Networks under Multi-Criteria

Fangda Guo, *Student Member, IEEE,* Ye Yuan, Guoren Wang, *Member, IEEE,* Lei Chen, *Member, IEEE,* Xiang Lian, and Zimeng Wang, *Student Member, IEEE,*

**Abstract**—The group nearest neighbor (GNN) search on a road network $G_r$, i.e., finding the spatial objects as activity assembly points with the smallest sum of distances to query users on $G_r$, has been extensively studied; however, previous works neglected the fact that social relationships among query users, which ensure the maximally favorable atmosphere in the activity, can play an important role in GNN queries. Meanwhile, the ratings of spatial objects can also be used as recommended guidelines. Many real-world applications, such as location-based social networking services, require such queries. In this paper, we study two new problems: (1) a GNN search on a road network that incorporates cohesive social relationships (CGNN) and (2) a CGNN query under multi-criteria (MCGNN). Specifically, both the query users of highest closeness and the corresponding top-$j$ objects are retrieved. To address critical challenges on the effectiveness of results and the efficiency of computation over large road-social networks: (1) for CGNN, we propose a filtering-and-verification framework. During filtering, we prune substantial unpromising users and objects using social and geospatial constraints. During verification, we obtain the object candidates, among which the top $j$ are selected, with respect to the qualified users; (2) for MCGNN, we propose threshold-based selection and expansion strategies, where different strict boundaries are proposed to ensure that correct top-$j$ objects are found early. Moreover, we further optimize search strategies to improve query performance. Finally, experimental results on real social and road networks significantly demonstrate the efficiency and efficacy of our solutions.

**Index Terms**—Query processing, GNN query, $k$-core, graph algorithm, road network, social network.

## 1 Introduction

With the ever-growing popularity of GPS-enabled mobile devices, many location-based service (LBS) systems (e.g., Google Maps) have been deployed and widely accepted by mobile users, who use them to easily capture and upload their own locations during daily activities. Along with the widespread prevalence of LBSs, recent years have witnessed a massive explosion in location-based social networking (LBSN) applications, such as Yelp, Foursquare and Facebook Places. In all these applications, social network users are always associated with location information (e.g., public places, home/office addresses) and their evaluations/ratings for points of interest (POIs), which can be shared with their "friends".

It is envisaged that such location information can bridge the gap between the physical world and the virtual world of social networks, and the ratings of POIs can be used as recommended guidelines. In addition, the nearest neighbor (NN) search and its variants on road networks are fundamental issues in LBSs due to their importance in a wide spectrum of applications [1], [2], [3]. Such search capabilities provide social network users with new opportunities for rapidly organizing impromptu offline activities. Below is a typical example query scenario arising in the new context of road-social networks.

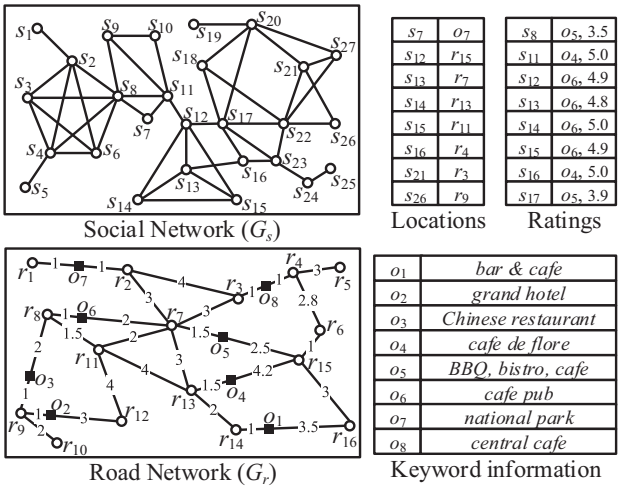



Fig. 1. Example of road-social network.

**Q1** To hold a board game party (e.g., Monopoly), Steve hopes to gather 3 people, each of whom preferably knows others, and to find two highly ranked cafes or bars near everyone (e.g., within 6.5 km) as options while minimizing the total travel distance.

**Q2** Headquarters plans to invite a group (e.g., 30) of branch managers, who better have frequent business contacts or participated in the same projects, and to reserve a hotel ballroom with minimum total travel cost and no more than 10km away from the farthest branch to hold a banquet.

As significant and substantial manual coordination is


- *F. Guo, Y. Yuan, and Z. Wang are with the School of Computer Science and Engineering, Northeastern University, Shenyang, Liaoning, China. E-mail: {fangda@stumail, yuanye@mail, zimeng@stumail}.neu.edu.cn*
- *G. Wang is with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China. E-mail: wanggr@bit.edu.cn*
- *L. Chen is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, China. E-mail: leichen@cse.ust.hk*
- *X. Lian is with the Department of Computer Science, Kent State University, Ohio, USA. E-mail: xlian@kent.edu*

still required, none of the existing methods can solve the above problem. Specifically, the challenges faced in organizing such activity lie in issuing timely invitations that specify both optimal invitees and suitable assembly points, in accordance with the closeness of a limited number of candidate attendees and their proximity to corresponding locations in physical world, and sometimes even the ratings of assembly points as additional references. Intuitively, when attendee size increases, the organization process becomes more complicated and thereby too tedious to coordinate manually. Thus, it is imperative to develop efficient new techniques to alleviate the effort and support impromptu offline activity planning services. The specific motivating example follows.

**Example 1.** Fig. 1 illustrates Q1 over a road-social network, which is split into a social layer ($G_s$) and a road layer ($G_r$) for clarity. The circles in $G_s$ and $G_r$ represent users and road intersections/end-points, respectively. In $G_r$, the rectangles denote spatial-textual objects lying on edges, for which keyword information is listed; the number on each edge represents travel distance on corresponding road segment. In $G_s$, the (partial) published user location and evaluation information provides mappings specifying the location $r \in G_r$ of any user $s \in G_s$ and the ratings of POIs visited. Suppose that $s_{13}$ is Steve, whose current location is $r_7$ in $G_r$. Generally, high rankings can be measured in two ways. (1) Subjectivity: ratings are not referenced. From Steve's perspective, he may be advised that $s_{12}$, $s_{15}$, and $s_{16}$ (at $r_{15}$, $r_{11}$, and $r_4$, respectively) could be invited to $o_5$ or $o_8$, both of which meet his requirements of proximity 6.5 km and keyword "cafe/bar", with a minimum total travel distance of 13.8 or 15.8 km, respectively. However, as observed in $G_s$, $s_{12}$ and $s_{15}$ are both unacquainted with $s_{16}$; in contrast, $s_{12}$, $s_{14}$, and $s_{15}$ (at $r_{15}$, $r_{13}$, and $r_{11}$, respectively) know each other best and thus are the optimal invitees. Accordingly, $o_5$ and $o_6$, with minimum total travel distances of 12 and 15.5 km, respectively, are the best selections that should be ultimately recommended to Steve. (2) Objectivity: ratings are referenced. At this point, some POIs that are relatively far away but have higher ratings can also be considered and topped the list. In view of the ratings available, we can measure the average rating of $o_5$ at $\frac{3.5+3.9}{2} = 3.7$ and its overall score (e.g., ratio of rating to total travel distance) at $\frac{3.7}{12} = 0.31$. However, both $o_4$ and $o_6$ have the overall scores of $0.32$ such that they are the best selections with respect to $s_{12}$, $s_{14}$, $s_{15}$ and Steve.

This example motivates us to consider two novel types of queries on road-social networks, namely, (1) cohesive group nearest neighbor queries (CGNN) and (2) cohesive group nearest neighbor queries under multi-criteria (MCGNN), then to propose efficient processing algorithms respectively. Specifically, for CGNN queries, given $G_s$, $G_r$, the number $c$ of activity attendees (including a query user $u_q$), and an upper limit $\epsilon$ on distance/time, an activity initiator[1] issues a CGNN query that should return $c$ attendees and the top-$j$ objects (i.e., assembly points) from $G_r$ containing keywords $\omega$ such that the travel cost of each attendee is within $\epsilon$, and the total travel cost of all attendees is minimized; for MCGNN queries, $c$ attendees and the top-$j$ objects containing $\omega$ should be returned such that the travel cost of each attendee is within $\epsilon$, and the overall scores of objects are maximized under multi-criteria (e.g., ratings of POIs and total travel cost of all attendees). It is worth mentioning that both queries incorporate a social constraint on the closeness of the $c$ attendees: each invitee of $u_q$ in $G_s$ should know the others as well as possible.

*Challenges.* In this paper, for both CGNN and MCGNN queries, we address the following three challenges. (1) Since the number of attendees is limited to $c$ and their closeness is to be maximized, the social constraint among attendees is different from the traditional $k$-core [4], a well-known concept in graph theory, which is the maximal induced subgraph in which every vertex has at least $k$ neighbors. Thus, selecting of the most cohesive set of attendees is nontrivial because of the massive number of possible combinations to be evaluated. (2) Retrieving the total travel cost to a target object in road networks, as opposed to geometric distances in Euclidean space, is more complex since location and accessibility of objects are restricted by the computation of network distance, particularly with increasing $c$. (3) Both the social and road networks are typically massive. For example, Facebook has 1 billion users, and the USA road network alone has more than 20 million vertices. Thus, it is challenging to efficiently process the queries over typically large road-social networks. Moreover, for MCGNN queries, we have to address an additional challenge as follow: (4) The effectiveness of results and efficiency of computation are the keys of top-$j$ query processing [5], [6], [7] under multi-criteria, where we need a meaningful combining scoring function that supports generality and comparability, and a strict boundary to limit the huge search space.

*Our Solution and Contributions.* A simple strategy for solving a CGNN/MCGNN query is to enumerate all combinations of $c-1$ invitees of $u_q$, select the most cohesive one(s), compute the total cost/overall score for each object by traversing $G_r$, and return the top-$j$ objects within $\epsilon$. However, this strategy is obviously infeasible due to the massive costs of enumeration and traversal.

To address the above challenges, (1) for both queries, the most cohesive $k$-core model is developed to quantify closeness and ensure the maximally favorable atmosphere, avoiding enumeration while rapidly locating comprehensive solutions that consider both social network topology and activity scale; (2) different incremental and accumulative strategies are adopted to gradually expand the search space in road networks: while for CGNN queries, an effective verification strategy is designed to expedite the extraction of top-$j$ objects among candidates (i.e., filtering-and-verification framework); for MCGNN queries, different strict boundaries are proposed to ensure that correct top-$j$ objects are found early, and an overall score upper bound for any possible object on edge is studied to speed up query processing; (3) for both queries, a state-of-the-art hierarchical tree structure [8] is adopted to index road networks, allowing vertices/objects to be obtained more quickly. The principal contributions are summarized as follows.

- We formulate two pragmatic query types on road-social networks, CGNN/MCGNN queries, to identify suitable spatial-textual objects as assembly points

1. To support general offline activity planning, attendees include the query user and other invitees but not necessarily the activity initiator.

for optimal sets of attendees. Both can accommodate various types of offline activities of specified scales.
- We develop the first efficient algorithm for finding the most cohesive $k$-core model to ensure the social closeness of a limited number of attendees.
- Effective pruning and verification strategies, and threshold-based selection and expansion strategies for finding the top-$j$ CGNNs and MCGNNs are proposed, respectively. Optimization techniques are designed to further improve query processing.
- We conduct extensive experiments on real-world datasets to demonstrate the effectiveness and efficiency of our proposed strategies and algorithms.

The rest of the paper is organized as follows. Section 2 formulates both CGNN and MCGNN problems. Section 3 presents algorithms for CGNN queries. Section 4 proposes approaches for MCGNN queries. Section 5 discusses optimizations. Section 6 reports experimental results. Section 7 reviews related work, and Section 8 concludes the paper.

## 2 Problem Definition

In this section, we formally define our CGNN and MCGNN queries over road-social networks. Table 1 summarizes the mathematical notations used throughout this paper.

### 2.1 Preliminaries

*Road network.* In this paper, a road network is modeled as an undirected weighted graph $G_r = (V_r, E_r)$, where $V_r$ is a set of vertices and $E_r = \{(u, v) | u, v \in V_r \wedge u \neq v\}$ is a set of edges. A vertex $v_r \in V_r$ represents a road intersection or an end of a road, and an edge $e_r = (u, v) \in E_r$ represents a road segment that enables travel between vertices $u$ and $v$. Each edge $(u, v)$ is associated with a nonnegative weight $w(u, v)$ that represents the cost (e.g., distance or travel time) of a corresponding road segment.

Let $p$ be a spatial point lying on the edge $(u, v)$. The travel cost from vertex $u$ to $p$, denoted by $w(u, p)$, is assumed to be proportional to the distance (length) between them. For two given points $u$ and $v$ in $G_r$, we use $dist(u, v)$ to represent the *network distance* (cost) between $u$ and $v$, which is the sum of the edge weights along the least costly path from $u$ to $v$. Note that the least costly path corresponds to the shortest path if the edge weight represents the distance.

A spatial-textual object[2] $o \in O$ is described by a spatial point and a set of keywords from a vocabulary, denoted by $o.loc$ and $o.T$, respectively. For simplicity, we assume that objects always lie along the edges (i.e., road segments) of $G_r$. Suppose that an object $o$ lies on $(u, v)$ with a given cost to each end vertex $u$ and $v$. A new vertex can be created for $o$, and $(u, v)$ is then replaced with edges $(u, o)$ and $(o, v)$.

Thus, we can define the aggregate network distance between an object $o$ and a set of locations $Q$ as follows:

$$dist_f(o, Q) = f_{\forall q_i \in Q} dist(o, q_i), \quad (1)$$

where $f$ is an aggregate function that applies to sets of numbers. In this paper, we simultaneously consider two types of aggregate functions: $dist_{sum}(o, Q) = \sum_{\forall q_i \in Q} dist(o, q_i)$ and $dist_{max}(o, Q) = \max_{\forall q_i \in Q} dist(o, q_i)$.

2. Hereafter, when there is no ambiguity, "spatial-textual object" is abbreviated to "object".

TABLE 1
Summary of Notations

| Notation | Description |
|---|---|
| $c, u_q$ | attendee size and query user for an activity |
| $\omega, \epsilon$ | set of keywords and a distance/time threshold |
| $j$ | number of assembly points to select among |
| $G_s(J), G_s(V_{s^*})$ | subgraphs of $G_s$ induced by $J$ and $V_{s^*}$ |
| $G_s^k, G_s^k(u_q, c)$ | $k$-core of $G_s$ and cohesive $k$-core(s) of $u_q$ in $G_s$ |
| $G_s^{k_{max}}(u_q, c)$ | $G_s^k(u_q, c)$ with maximum coreness |
| $O$ | set of spatial-textual objects |
| $L_s(u_q)$ | published location of $u_q$ in the road network |
| $R_s(v_s, o)$ | rating of object $o$ given by user $v_s$ |
| $\overline{R_s}(U, o)$ | average rating of $o$ against a set of users $U$ |
| $Q$ | query points, i.e., $L_s(G_s^{k_{max}}(u_q, c))$ |
| $N_{G_s}(v), N_{G_s^k}(v)$ | sets of neighbors of $v$ in $G_s$ and $G_s^k$ |
| $dg_{G_s}(v)$ | degree of vertex $v$ in $G_s$ |
| $w(u, v)$ | cost of the road segment between $u$ and $v$ |
| $dist(u, v)$ | network distance between $u$ and $v$ |
| $dist_{sum}(o, Q)$ | total cost from object $o$ to $Q$ |
| $dist_{max}(o, Q)$ | maximum cost from $o$ to any $q_i \in Q$ |
| $\theta(o, U)$ | overall score of object $o$ against a set of users $U$ |
| $Lc/Li$ | largest increment in coreness/largest incidence |
| $TS/TRS$ | threshold selection/threshold ripple selection |
| TE/HE | threshold expansion/holistic expansion |

For example, Fig. 1 displays a road network $G_r$ and a textual description of each object. Edge $(r_7, r_{15})$ has a distance weight of $w(r_7, r_{15}) = 4$. Object $o_5$ lies on this edge, with $w(r_7, o_5) = 1.5$ and $w(r_{15}, o_5) = 2.5$. The path $r_{11}r_7r_{15}$ is the shortest path from $r_{11}$ to $r_{15}$, with $dist(r_{11}, r_{15}) = 6$.

*Social network.* We model a social network as an unweighted and undirected graph $G_s = (V_s, E_s, L_s, R_s)$, where $V_s$ is the set of vertices (representing users), $E_s \subseteq V_s \times V_s$ is the set of edges (i.e., social relations), $L_s$ and $R_s$ are the sets of mappings defined on $V_s$ such that for each vertex $v_s$ in $V_s$, $L_s(v_s)$ specifies the attributes of $v_s$ (e.g., name, gender, and location), and $R_s(v_s, o^*)$ records the rating given by $v_s$ to the visited object $o^*$. In our case, $L_s(v_s)$ provides a mapping of each user's location in the road network. Given a vertex $v_s$, we denote the set of its neighbors, $\{u_s | (u_s, v_s) \in E_s\}$, by $N_{G_s}(v_s)$. The degree of $v_s$, $|N_{G_s}(v_s)|$, is denoted by $dg_{G_s(v_s)}$. Then, we can formally define induced subgraph as follows.

**Definition 1 (Induced Subgraph).** *A graph $G_s(V_{s^*}) = (V_{s^*}, E_{s^*}, L_{s^*})$ is called the subgraph of $G_s$ induced by $V_{s^*}$, where (1) $V_{s^*} \subseteq V_s$, (2) edge $(u, v) \in E_{s^*}$, iff $u, v \in V_{s^*}$, $(u, v) \in E_s$, and (3) for each $v \in V_{s^*}$, $L_{s^*}(v) = L_s(v)$.*

The $k$-core concept [4], which has been widely used to describe cohesive subgraphs, is formally defined as follows.

**Definition 2 ($k$-Core).** *Given a graph $G_s$, an induced subgraph $G_s(J)$ is the $k$-core of $G_s$, denoted by $G_s^k$, iff the following two conditions are true. (1) $k$-*degree*: $dg_{G_s(J)}(v) \geq k$ for every $v \in J$. (2)* maximality*: For any $J'$ such that $J \subset J' \subseteq V_s$, there exists a $u \in J' \setminus J$ such that $dg_{G_s(J')}(u) < k$.*

Note that we have $G_s^{k+1} \subseteq G_s^k$ [9]. A vertex $v \in G_s$ has *coreness* $k$ if it belongs to $G_s^k$ but not to $G_s^{k+1}$. For any $V \subseteq V_s$, the largest coreness in a graph $G_s(V)$ is called the *coreness* of $G_s(V)$, which is denoted by $cn(G_s(V))$.

### 2.2 Definition of the CGNN Problem

Now we can formally define the cohesive group nearest neighbor queries (CGNN) over road-social networks.

**Definition 3 (Cohesive $k$-Core).** *Given a constant $c$ and a query user $u_q \in J \subseteq V_s$ in the social network $G_s$, the cohesive*

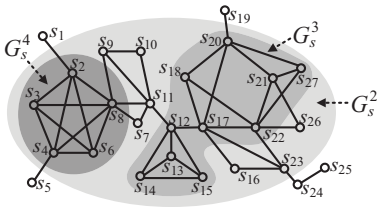



Fig. 2. $k$-cores of $G_s$ in Fig. 1.

*$k$-core, denoted by $G_s^k(u_q, c)$, is a connected induced subgraph $G_s(J)$ such that $G_s(J)$ is a $k$-core and $|J| = c$.*

We may conclude that the user group $J$ of $G_s^k(u_q, c)$ consists of $\{u_q, u_1, \ldots, u_{c-1}\}$, whose coreness can be expressed as $cn(G_s(J))$. Among all $k$-cores $G_s^k(u_q, c)$, any $k$-core with the maximum coreness is referred to as the most cohesive $k$-core, denoted by $G_s^{k_{max}}(u_q, c)$; i.e., $\forall k' > k_{max}$, there exists no $G_s^{k'}(u_q, c)$. Note that $k_{max}$ can be up to $c-1$.

**Example 2.** In Fig. 2, $G_s^2, G_s^3$ and $G_s^4$ are annotated in different colors. Suppose that $u_q = s_8$ and $c = 5$; then, the subgraph induced by $\{s_7, s_8, s_9, s_{10}, s_{11}\}$ is a cohesive 2-core. Similarly, another six subgraphs can also be identified as $G_s^2(s_8, 5)$. However, there exists one additional subgraph induced by $\{s_2, s_3, s_4, s_6, s_8\}$ with a coreness of 4 that is the most cohesive $k$-core, i.e., $G_s^4(s_8, 5)$.

**Definition 4 (Maximum Connected Component).** *The maximum connected component of graph $G_s$ with regard to a vertex $u_q$ and a constant $c$, denoted by $C_{max}(u_q, c)$, is the component that contains $u_q$ in the $k$-core with maximum coreness among all $k$-cores of $G_s$ such that $|C_{max}(u_q, c)| \geq c$.*

*Road-social network.* A road-social network is a pair of graphs $(G_r, G_s)$, where $G_r$ is a road network and $G_s$ is a social network. Each vertex $u_s \in G_s$ is associated with a vertex $v_r$ or a spatial point $p$ of $G_r$, indicating that user $u_s$ is currently in location $v_r$ or $p$, i.e., $L_s(u_s) = v_r$ or $p$ in our case.

For example, Fig. 1 shows a road-social network. The mapping between $s_{26}$ and $r_9$ in the published user location information indicates that $s_{26}$ is currently at $r_9$.

*Problem statement (CGNN).* Given graphs $G_s$ and $G_r$, an activity initiator issues a query $q = \langle u_q, c, \omega, \epsilon, j \rangle$, where $u_q$ is the query user in $G_s$, $c$ is the attendee size, $\omega$ is a set of keywords, $\epsilon$ is the network distance threshold and $j$ is an integer. A CGNN query retrieves $c$ users and a corresponding set $A_q$ of $j$ objects from set $O$ with the smallest total cost to all locations in set $Q$ over the road-social network such that (1) $Q$ consists of locations in $G_r$ associated with the $c$ users from $G_s^{k_{max}}(u_q, c)$, denoted by $L_s(G_s^{k_{max}}(u_q, c))$, i.e., $\{L_s(u_q), L_s(u_1), \ldots, L_s(u_{c-1})\}$; (2) $A_q \subseteq O^* \subseteq O$ in $G_r$, $\forall o \in O^*, w \subseteq o.T$, and $|A_q| = j$; and (3) $\forall o \in A_q, \forall o' \in O^* \setminus A_q$, both $dist_{sum}(o, Q) \leq dist_{sum}(o', Q)$ and $dist_{max}(o, Q) \leq \epsilon$.

We regard the locations that comprise the most cohesive $k$-core for a group of users as query points $Q$; e.g., CGNN query $q = \langle Steve, 4, \text{“}cafe\text{”}, 6.5, 2 \rangle$ maps to Q1. Note that there may be more than one group of $c$ users in $G_s^{k_{max}}(u_q, c)$. As illustrated in Section 3.2, the local-search-based solution returns each group in $G_s^{k_{max}}(u_q, c)$ but the heuristics only select the most promising one rapidly leading to a solution.

### 2.3 Definition of the MCGNN Problem

In this section, we formally define the cohesive group nearest neighbor queries under multi-criteria (MCGNN) over road-social networks. As shown in Example 1, there usually exist objects that are relatively far away from the query points $Q$, i.e., $L_s(G_s^{k_{max}}(u_q, c))$, but objectively rated higher. In fact, more criteria can be considered as metrics as well. Given such multidimensional metrics that need to be referenced simultaneously, more objects which can substitute the set $A_q$ of $j$ objects ultimately recommended to $u_q$ typically have to be considered, compared to CGNN queries. In this way, a meaningful combining scoring function is urgently needed that supports generality and comparability.

Thus, we define the overall score $\theta^*(o, U)$ of an object $o$ against a set of users $U$ under $m$ criteria in the road-social network $(G_r, G_s)$ as follows:

$$\theta^*(o, U) = \sum_{i=1}^{m} \alpha_i \cdot d_i, d_i = \begin{cases} s_i(o, U) & \text{if beneficial;} \\ 1 - s_i(o, U) & \text{otherwise;} \end{cases} \quad (2)$$

where $d_i$ is the $i$-th dimensional metric and $s_i(o, U)$ is the normalized score of the $i$-th dimensional criterion w.r.t. $o$ and $U$ (e.g., $U$'s rating of $o$), $\alpha_i \in (0, 1)$ for each $i \in [1, m]$ is the weight of $d_i$ used to balance the $m$ criteria and that $\sum_{i=1}^{m} \alpha_i = 1$. Generally, for criterion that is not expected by or not beneficial to users, the smaller its $s_i(o, U)$, the better (e.g., $U$'s total travel cost to $o$, corresponding to $d_i = 1 - s_i(o, U)$); thus the greater the overall score, the better.

*Problem statement (MCGNN).* Given graphs $(G_r, G_s)$, a MCGNN query $q = \langle u_q, c, w, \epsilon, j \rangle$ retrieves $c$ users, but unlike CGNN queries, the corresponding set $A_q$ consists of $j$ objects with the greatest overall scores over the road-social network, where it retains the conditions (1) and (2) but replaces the condition (3) in CGNN problem statement: $\forall o \in A_q, \forall o' \in O^* \setminus A_q$, both $\theta^*(o, G_s^{k_{max}}(u_q, c)) \geq \theta^*(o', G_s^{k_{max}}(u_q, c))$ and $dist_{max}(o, Q) \leq \epsilon$.

For ease of presentation, we consider MCGNN queries in two-dimensional metrics ($m = 2$) in the following, i.e., average rating and total travel cost, such that $\theta^*(o, U) = \alpha_1 \cdot \|\overline{R_s}(U, o)\| + (1 - \alpha_1) \cdot (1 - \|dist_{sum}(o, L_s(U))\|)$, where $\overline{R_s}(U, o) = \sum_{\forall u_i \in U} R_s(u_i, o)/|U|$ represents the average rating of $o$ against $U$, and $\|\cdot\|$ indicates normalization. In order to avoid normalizing $dist_{sum}(o, L_s(U))$[3], which is a requirement for using $\theta^*(o, U)$, we employ an equivalent $\theta(o, U)$ instead of $\theta^*(o, U)$ as follows:

$$\theta(o, U) = \frac{\overline{R_s}(U, o)}{\alpha \cdot dist_{sum}(o, L_s(U))}. \quad (3)$$

Here, $\alpha$ is a positive real number and defines the importance of one measure over the other. For example, $\alpha > 1$ increases the importance of the aggregate network distance over the average rating. In particular, if $u_i$ has not been or rated $o$ before, $R_s(u_i, o)$ will be replaced by an average rating for all users who have rated in $G_s$. Note, however, that our approaches in Section 4 also support $\theta^*(o, U)$ and can easily be applied to the cases of $m > 2$.

## 3 Algorithms for CGNN Queries

This section presents our efficient approach for CGNN queries. In Section 3.1, we briefly introduce the framework of our solution. Section 3.2 shows how to find the most cohesive $k$-core, and Section 3.3 presents efficient algorithms for

3. In the context of road networks, normalizing the network distance requires computing the shortest path between any two points, which is prohibitively costly to process in practice.

**Procedure** CGNN_Framework{
**Input**: $(G_r, G_s)$, $q = \langle u_q, c, w, \varepsilon, j\rangle$
**Output**: $G_s^{k_{max}}(u_q, c)$ and $A_q$
(1) find the most cohesive $k$-core of $u_q$ with size $c$
(2) prune objects with keywords $w$ and network distance $\varepsilon$
(3) verify top-$j$ qualified objects from candidate set $S$

Fig. 3. Framework for a CGNN query.

filtering out objects based on network distance restrictions. Section 3.4 describes new verification techniques for reducing the cardinality of the candidate objects and presents our final CGNN algorithm that integrates these techniques.

### 3.1 Framework of Our Approach

For processing CGNN queries, we propose the filtering-and-verification framework shown in Fig. 3. First, we locate and select the most cohesive $k$-core $G_s^{k_{max}}(u_q, c)$ for $u_q$. Second, we filter out objects beyond a given network distance threshold $\epsilon$ from the locations in $G_r$ (i.e., query points) linked to each user in $G_s^{k_{max}}(u_q, c)$ and keep the remaining objects. Then, we compute the next nearest neighbor (NN) of each query point to obtain the candidate set $S$ of objects until $j$ common objects are found for all query points. Finally, we identify unpromising objects through inference and return the top-$j$ qualified objects. The framework consists of a filtering (i.e., finding the most cohesive $k$-core in the social network), pruning objects based on keywords and a network distance $\epsilon$ on the road network, and verification (i.e., verifying qualified objects in candidate set $S$).

Note that we initially employ the popular inverted indexing technique to organize the objects. Thus, only objects $O^* \subseteq O$ whose textual descriptions correspond to all user-specified keywords $\omega$ are retained in the search, and all others are pruned. Loading objects that do not match all query keywords could result in performance degradation, especially when $\omega$ is not small. In the rest of this paper, we use $O^*$ to denote the objects meeting keywords $\omega$.

### 3.2 Finding the Most Cohesive k-Core

To determine the most cohesive group relationships between $u_q$ and its correlative neighbors, which may also be interconnected, we propose a local-search-based solution called center expansion (CE). The intuition is that the most cohesive group for a given vertex should be in the vicinity of the vertex. Thus, the entire $G_s$ is not necessarily involved in the search. The local-search-based solution works as follows.

CE leverages the social-distance-based pruning (SD) as described in Section 5.1 and the $k$-core decomposition [10], and treats $u_q$ as a center in $C_{max}(u_q, c)$ for outward diffusion, i.e., a breadth search. In each round of CE, multiple vertices are selected without duplication from among the neighbors of the center, and an unmarked vertex is taken as the new center. When the size of an expansion reaches $c$, the subgraph and its coreness are recorded. Once all possible expansions in the current connected component of $G_s^k$ have been completed, CE returns the induced subgraph(s) with coreness $k$, if any, as $G_s^{k_{max}}(u_q, c)$; otherwise, it expands $u_q$ in connected component of $G_s^{k-1}$. CE exploits graph topology to scale out from the center so that induced subgraph of each completed expansion is connected and includes $u_q$.

Although SD may reduce the search space and the time to compute $k$-core is linear in the number of edges [9], i.e., $\mathcal{O}(|E_s|)$, the cost is high due to the numerous combinations. The most cohesive $k$-core must exist in a subgraph of size $s$ containing $c$ users, and we must verify at least $C_{c-1}^{s}$ permutations; thus, the time complexity is $\mathcal{O}(s^c)$. Since the subgraph's coreness cannot be greater than that of $C_{max}(u_q, c)$, which is typically taken to be constant, we can think of $s$ as being closely related to the scale of $|V_s|$, especially when $G_s$ has a high density (e.g., a high average degree). Thus, the lower bound on the complexity of any exact algorithm is $\mathcal{O}(|V_s|^c)$. As shown in Section 6, our exact algorithm takes a relatively long time even on a graph with only thousands of vertices. Thus, a time complexity of $\mathcal{O}(|V_s|^c)$ is already beyond reach, and the solution is intractable on big data.

**Algorithm 1:** HeuristicsFramework$(u_q, c)$

**Input**: $u_q$: query user; $c$: size constraint
**Output**: $G_s^{k_{max}}(u_q, c)$
1 $G_s' \leftarrow$ Social Distance based Pruning$(u_q, c)$;
2 $C_{max}(u_q, c) \leftarrow k$-core of $G_s'$;
3 $queue$.enqueue$(u_q)$; $subset\ H \leftarrow \emptyset$;
4 **while** $queue \neq \emptyset$ **do**
5  $v \leftarrow queue$.dequeue(); $H \leftarrow H \cup \{v\}$;
6  **if** $|H| = c$ **then**
7  **return** $H$ as $G_s^{k_{max}}(u_q, c)$;
8  **foreach** $(v, w) \in$ *edges in* $C_{max}(u_q, c)$ **do**
9  **if** $w$ is not visited **then**
10  $queue$.enqueue$(w)$;

Next, we propose two intelligent lightweight heuristics of constant cost. The basic idea for refining candidate generation is to use a priority queue to select the most promising vertex that will rapidly lead to a solution. Theorem 1 can be used as a prerequisite to support our heuristics.

**Theorem 1.** *For graph $G_s$ and query vertex $u_q$, given an attendee size $c$, there must exist $G_s^{k_{max}}(u_q, c) \subseteq C_{max}(u_q, c)$.*

Intuitively, $G_s^{k_{max}}(u_q, c)$ is a subgraph of the connected component of $G_s$ containing $u_q$, whose size is no less than $c$ and coreness is the greatest. Note all the skipped proofs of theorems can be found in the supplemental material.

*Largest increment in coreness (Lc).* Selecting the vertex that leads to the largest increment in the coreness measure is a straightforward heuristic since the final goal for $G_s^{k_{max}}(u_q, c)$ is to find a subset $H$ satisfying that $|H| = c$ and $cn(G_s(H))$ is as close as possible to or even equal to $c$–1. In this strategy, the priority $f(v)$ of a vertex $v$ is defined as

$$f(v) = cn(G_s(H \cup \{v\})) - cn(G_s(H)). \quad (4)$$

This approach is a greedy one since only the improvement in $cn(H)$ in the next step is considered. Note that whenever a vertex is added to $H$, the coreness of the current $H$ is incremented by at most 1. Hence, this strategy is equivalent to random selection from the vertices adjacent to one of the vertices with the minimal degree in $H$.

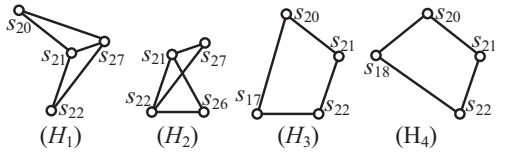



Fig. 4. Local search from $s_{21}$ with $c = 4$.

*Largest incidence (Li).* This selection approach is more intelligent. The priority of a vertex $v$ is defined as

$$f(v) = dg_{G_s(H \cup \{v\})}(v). \quad (5)$$

In this strategy, we select the vertex with the largest number of connections to the current $H$. This technique yields the fastest increase in the mean degree of $G_s(H)$. In general, the lowest degree of a graph increases as its density grows; consequently, a valid solution $H$ with $cn(G_s(H))$ is expected to be found within finite steps if such a solution exists.

**Example 3.** Suppose that $s_{21}$ wishes to find the most cohesive group of 3 invitees in Fig. 2 ($u_q = s_{21}, c = 4$). CE returns $G_s^2(s_{21}, 4)$, consisting of 4 different user groups, as shown in Fig. 4. On the other hand, if we always choose the vertex with the largest number of connections to $H$, i.e., use the *Lc* or *Li* selection strategy, only $G_s(H_1)$ or $G_s(H_2)$ is found. Because $u_q$ has 4 neighbors, one of which is selected after $u_q$, e.g., $s_{20}$, only $s_{27}$ and $s_{22}$ can be successively selected, yielding $G_s(H_1)$, due to the vertex priorities defined in Equations 4 and 5.

*Complexity analysis*. Let $n'$ and $m'$ denote the numbers of vertices and edges, respectively, in $G_s(H)$. Generally, *Lc* and *Li* can be implemented in $\mathcal{O}(n' + m'logn')$ time. When a new vertex $v$ is added to the queue, at most $dg_{C_{max}(u_q,c)}(v)$ queue update operations must be executed, that is, at most $m'$ update operations in total. Each queue operation (insert, delete, priority update) generally has a time complexity of $\mathcal{O}(logn')$. Through careful design, *Li* can be implemented in $\mathcal{O}(n'+m')$ time with an expansion cost of $\mathcal{O}(1)$. We maintain a set of lists, each containing vertices with the same $f(v)$. Each time $v$ is added to $H$, the $f(\cdot)$ value of $v$'s neighbors (except those already in $H$) increases by 1. We move each influenced neighbor from its original $f(\cdot)$ list to the list for $f(\cdot)+1$. In this way, we can always find one vertex with the maximal $f(\cdot)$ in $\mathcal{O}(1)$ time.

If there is a tie in the maximal $f(\cdot)$ values for both strategies, we select the vertex with the minimal social distance to $u_q$ based on the SD process described in Section 5.1; if there is still a tie, a vertex is selected arbitrarily.

### 3.3 Range Filter

To support general road networks with various cost models (e.g., distance or travel time), we adapt the incremental network expansion (INE) algorithm of [3] to incrementally access objects since INE does not rely on specific restrictions (e.g., Euclidean distances [3]) or precomputed road networks (e.g., shortcuts [11] or Voronoi diagrams [12]). In [3], network distance is calculated from scratch for each object; to alleviate computational cost, we integrate Dijkstra's algorithm [13] into INE such that the network distances to objects are calculated cumulatively from each query point $q_i \in Q$, i.e., $L_s(G_s^{k_{max}}(u_q, c))$, during network expansion.

For simplicity of presentation, in Algorithm 2, we assume that each query point $q_i$ starts from a vertex[4] $u$ in Line 3. For $q_i \in Q$, a min-priority queue $M_i \in M$ is used to store the vertices accessed during expansion, where $dist(n) = \infty$ if vertex $n$ has not been visited. Here, $dist(n) = dist(q_i, n)$ if a vertex $n$ is marked (Line 7). Similarly, we use $dist(o)$ to compute $dist(q_i, o)$, and $dist(o) = dist(q_i, o)$ if an object $o$ can be visited from both end vertices of its edge (Line 15). In Algorithm 2, vertices are accessed in nondecreasing order of their network distances from $q_i$. Line 6 updates $dist_T$,

4. If $q_i$ is on an edge, we can use the two end vertices of the edge to find nearby objects and merge the answer sets.

**Algorithm 2:** RangeFilter($Q, \epsilon$)

**Input**: $Q$: query points; $\epsilon$: network distance threshold
**Output**: $R$: objects with network distance restriction

```
1  R:=∅; M:=∅; dist_T:=0;
2  foreach query point q_i ∈ Q do
3      u ← the vertex where q_i is; M_i.enqueue(⟨u, 0⟩);
4      while M_i ≠ ∅ do
5          ⟨n, dist⟩ ← M_i.dequeue();
6          dist_T := dist; terminate while if dist_T > ε;
7          Mark the vertex n;
8          foreach unmarked vertex n_i ∈ Adj[n] do
9              Update dist(n_i) if dist(n_i) > dist + w(n, n_i);
10             foreach object o ∈ O* on (n, n_i) do
11                 dist(o):=dist+w(n, o); ⟨o, dist(o)⟩ → R_i;
12             M_i.enqueue(⟨n_i, dist(n_i)⟩);
13         foreach marked n_i ∈ Adj[n] do
14             foreach t ∈ R_i and t.o on (n_i, n) do
15                 t.dist := min{t.dist, dist + w(n, o)};
16     R_i:=R_i\t if ∃t ∈ R_i with t.dist>ε; ∪_{i=1}^{|Q|} R_i → R;
```

which is the lower bound on the network distance for any unmarked vertex. The expansion terminates when $dist_T > \epsilon$, which implies that $dist(q, n_x) > \epsilon$ for any unmarked vertex $n_x$. For each vertex $n_i$ in the list of vertices adjacent to $n$, Line 9 updates $dist(n_i)$ if $n_i$ is not marked, and the objects on edge $(n, n_i)$, that satisfy the keyword constraint are loaded if $n_i$ is visited for the first time (Line 10), followed by a network distance computation based on $dist(n)$ (Line 11). If vertex $n_i$ is already marked, Lines 13-15 may update the network distances of the objects since both end vertices ($n$ and $n_i$) of the edge are marked. Finally, objects with network distances longer than $\epsilon$ are pruned from $R_i \in R$. Note that Algorithm 2 can be adapted for directed road networks due to our integration of the INE and Dijkstra algorithm.

**Example 4.** In Fig. 1, suppose that $q_1$ is at $r_6$, with $\epsilon = 4.5$ and $\omega = \{\text{“}cafe\text{”}\}$. After $r_6$ is marked, its neighbors $r_{15}$ and $r_4$ with updated network distances are placed in $M_1 = \{\langle r_{15}, 1\rangle, \langle r_4, 2.8\rangle\}$ since no object is found on either edge $(r_6, r_{15})$ or $(r_6, r_4)$. Next, $\langle o_5, 3.5\rangle, \langle o_4, 5.2\rangle$ on $(r_{15}, r_7)$ and $(r_{15}, r_{13})$, derived from the first element in $M_1$, are discovered and placed in $R_1$. Then, $M_1 = \{\langle r_4, 2.8\rangle, \langle r_{16}, 4\rangle, \langle r_7, 5\rangle, \langle r_{13}, 6.7\rangle\}$ is updated. Now, $r_4$ reaches $r_3$ and $r_5$ with $\langle o_8, 3.8\rangle$, and $r_{16}$ reaches $r_{14}$ with $\langle o_1, 7.5\rangle$. Since the distance to the next entry $\langle r_3, 4.8\rangle$ in $M_1$ is larger than $\epsilon$, finally, $R_1 = \{\langle o_5, 3.5\rangle, \langle o_8, 3.8\rangle\}$ with $\langle o_4, 5.2\rangle$ and $\langle o_1, 7.5\rangle$ discarded.

*Complexity analysis*. Let $v'$ and $e'$ denote the numbers of vertices and edges, respectively, accessed during road network expansion; then, Algorithm 2 can be implemented with a time complexity of $\mathcal{O}(c(v'log(v')+e'))$.

### 3.4 Verification

This section presents our verification techniques for CGNN queries on road-social networks. Henceforth, the next NN of a query point $q_i$ refers to the object last obtained from $R_i$ as described in Section 3.3. The process of continuously obtaining the next NN for a query point is regarded as the expansion of that query point, and the expansion order determines the next query point to be expanded.

#### 3.4.1 Obtaining the Candidate Set

The goal of this process is to obtain a set $S$ of CGNN candidates that includes the final results. We assume that there

**Algorithm 3:** ObtainCandidates($Q, R$)

**Input**: $Q$: query points; $R$: objects of network distances
**Output**: $S$: candidate set; $CO$: common objects; $dst_j$: total network distances of $o_j$ to $Q$

1 $H \coloneqq \emptyset$; $S \coloneqq \emptyset$; $dst_j \coloneqq \infty$; $CO \coloneqq \emptyset$;
2 **foreach** $q_i \in Q$ **do**
3 Get $o^1_{q_i} \in R_i$; $E_i = \{o^1_{q_i}\}$; $H$.push($\langle q_i, dist(q_i, o^1_{q_i})\rangle$);
4 **while** $|\cap_{i=1}^{c} E_i| < j$ **do**
5 $e \leftarrow$ Min($H$); $q_{cur} \coloneqq e.q$; get $q_{cur}$'s next NN $o' \in R_{cur}$;
6 $E_{cur} \coloneqq E_{cur} \cup \{o'\}$; $H$.push($\langle q_{cur}, dist(q_{cur}, o')\rangle$);
7 **foreach** $o \in \cap_{i=1}^{c} E_i$ **do**
8 $CO \coloneqq CO \cup \langle o, dist_{sum}(o, Q)\rangle$;
9 **return** $S \coloneqq \cup_{i=1}^{c} E_i$; $dst_j \coloneqq dist_{sum}(o_j, Q)$; $CO$;

are $c$ query points with respect to $Q$, and the acquisition of $S$ consists of the following steps.

We obtain the first NN, denoted by $o^1_{q_i}$, from $R_i$ for each $q_i \in Q$, and place each into its own expansion list $E_i$. We then check whether there are $j$ intersections among $E_1$ to $E_c$. If not, we continue to expand the search space for the query point $q_i$ in sequence.

Note that before retrieving the NNs for each $q_i$, we can simply arrange all corresponding objects in the result set $R_i$ generated by Algorithm 2 in ascending order of the network distance. After $o^k_{q_i}$ is retrieved, we place it at the end of the expansion list $E_i$, which is actually an ordered list of $q_i$'s NNs, i.e., $E_i = \{o^1_{q_i}, o^2_{q_i}, \ldots, o^k_{q_i}\}$. This phase stops when there are $j$ intersections among $E_1$ to $E_c$, i.e., $|\cap_{i=1}^{c} E_i| = j$, which means that we have identified a set $CO$ of the first $j$ common objects included in the expansions of all $q_i$. Here, $CO$ is the set of current best CGNN candidates; then, we can compute the total network distance $dist_{sum}(o_j, Q)$ between the $j$-th common object $o_j$ and $Q$, denoted by $dst_j$.

**Theorem 2.** *Once the $j$-th common object $o_j$ in the expansions of all $q_i$ is identified, i.e., $\cap_{i=1}^{c} E_i = \{o_1, o_2, \ldots, o_j\}$, the top-$j$ CGNNs are contained in $S = \cup_{i=1}^{c} E_i$.*

Algorithm 3 details the acquisition of the candidate set (Lines 1-9). For each query point $q_i$, the algorithm finds the first NN and adds it to the expansion list $E_i$ for that query point. In addition, each query point is inserted into a min-heap ($H$) with weight $dist(q_i, o^1_{q_i})$ (Lines 2-3). Then, we obtain the next NN (e.g., $o'$) of the top element in $H$ and insert it into the corresponding expanded set $E_{cur}$ of the current query point $q_{cur}$. We also update the weight of $q_{cur}$ in $H$ to $dist(q_{cur}, o')$. These procedures are repeated until there are $j$ common objects in the expansion lists for all query points (Lines 4-6). Once the set $CO$ of common objects has been constructed, the total network distance $dst_j$ between $o_j$ and $Q$ is returned (Line 9).

**Example 5.** Consider a query $q = \langle Steve, 3, \text{“}cafe\text{”}, 10, 2\rangle$. Fig. 5 shows an outline of network distances between $Q$ (at $r_7, r_{15}, r_{11}$) and eligible objects. We now obtain the first NN for each $q_i \in Q$, i.e., $E_1 = \{o_5\}, E_2 = \{o_5\}, E_3 = \{o_6\}$, and $H = \{\langle q_1, 1.5\rangle, \langle q_2, 2.5\rangle, \langle q_3, 2.5\rangle\}$. The expansion of $Q$ continues until 2 common objects are found. At this point, $E_1 = \{o_5, o_6, o_8, o_4\}, E_2 = \{o_5, o_4\}, E_3 = \{o_6, o_5, o_4\}$, and $H = \{\langle q_2, 4.2\rangle, \langle q_1, 4.5\rangle, \langle q_3, 5.5\rangle\}$. Thus, $S = \{o_4, o_5, o_6, o_8\}$, and $CO = \{\langle o_5, 7.5\rangle, \langle o_4, 14.2\rangle\}$.

### 3.4.2 Verification

Our objective is to retain qualified objects that will be among the top-$j$ CGNNs from the candidate set $S$. That is, if there are only $j$ objects in $S$, then they are the CGNNs we seek. We now present our verification techniques in detail. Before verifying objects from $S$, we select a query point $q_i$ according to a certain expansion order of $Q$ to retrieve the next NN and then compute $dst = \sum_{i=1}^{c} dist(q_i, o_i)$, where

$$o_i = \begin{cases} o' & \text{if } o' \in E_i; \\ o^1_{q_i} & \text{otherwise.} \end{cases} \quad (6)$$

If $dst > dst_j$, we calculate the set $E'_i$ for each $q_i$.

$$E'_i = \begin{cases} \{o | o \in E_i, dist(q_i, o) < dist(q_i, o')\} & \text{if } o' \in E_i; \\ \emptyset & \text{otherwise.} \end{cases} \quad (7)$$

Then, let $S' = \cup_{i=1}^{c} E'_i$. For all $o \notin S \cap S'$, object $o$ can be discarded from $S$, as stated in Theorem 3.

**Theorem 3.** *Let $o'$ be the next NN and $dst = \sum_{i=1}^{c} dist(q_i, o_i)$. If $dst > dst_j$ and $\forall o \notin S \cap S'$, it is verifiable that object $o$ is not among the CGNNs and can be discarded from $S$.*

If $dst \le dst_j$, we need only to place $q_i$'s next NN $o'$ into its expanded set $E_i$ and then determine whether $o'$ is included in the expansions of all query points. If so, we replace $\langle o_j, dst_j\rangle$ with $\langle o', dst\rangle$ in the set of common objects, i.e., $CO$, where we update $dst_j$ with the newly elected $o_j$ after sorting; otherwise, we save an entry consisting of $o'$, the corresponding $dst$, and every current $E_i$ in the unpruned set $UP$. Note that this action enables us to continue verifying the qualified $o$ remaining in $S$ with both the expanding and backtracking strategies based on up-to-date $dst_j$ and not-yet-validated entries in $UP$. We proceed in this way until there are only $j$ objects in $S$, which are the top-$j$ CGNNs.

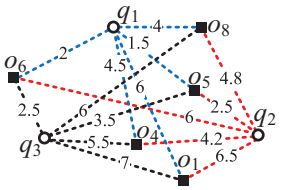



Fig. 5. Outline of network distances.

**Algorithm 4:** Verification($H, S, CO, R$)

**Input**: $H$: min-heap; $S$: candidate set; $CO$: common objects; $R$: objects of network distances
**Output**: CGNNs

1 **while** $|S| > j$ **do**
2 $e \leftarrow$ Min($H$); $q_{cur} \coloneqq e.q$; get $q_{cur}$'s next NN $o' \in R_{cur}$;
3 $dst = \Sigma^{c}_{i=1 \wedge o' \in E_i} dist(o', q_i) + \Sigma^{c}_{i=1 \wedge o' \notin E_i} dist(o^1_{q_i}, q_i)$;
4 **if** $dst > dst_j$ **then** $S \coloneqq S \cap (\cup_{i=1}^{c} E'_i)$; $\{o'\} \rightarrow E_{cur}$;
5 **else**
6 $\{o'\} \rightarrow E_{cur}$;
7 **if** $o' \in \cap_{i=1}^{c} E_i$ **then**
8 Update $S$ and $CO$; $\langle o_j, dst_j\rangle \leftarrow$ Sort($CO$);
9 **if** $\exists t \in UP$ **and** $t.dist > dst_j$ **then**
10 $E'_i \leftarrow t.E_i$; $S \coloneqq S \cap (\cup_{i=1}^{c} E'_i)$; $UP \setminus \{t\}$;
11 **else** $UP$.push($\langle o', dst, E_1, E_2, \ldots, E_c\rangle$);
12 $H$.push($\langle q_{cur}, dist(o', q_{cur})\rangle$);
13 **return** CGNNs $\coloneqq S$;

In Algorithm 4, Line 2 first extracts the head of min-heap $H$ and obtains its next NN in every round. Then, we apply our verification strategies to speed up the CGNN query processing. The value of $dst$ is the lower bound on the total network distance from $o'$ to all query points (Line 3). Line 4 verifies the objects that cannot be CGNNs by taking the intersection of $S$ with the union of the $E'_i$. If $dst \le dst_j$ and $o'$

is a common object, then Line 8 updates the current set $CO$ and $dst_j$. Line 12 updates the weight of $q_{cur}$ in $H$. In Lines 9-10, we use backtracking verification method to accelerate the convergence of qualified objects retained from $S$.

**Example 6.** Continuing *Example* 5, we obtain $q_2$'s next NN $o_8$ from $R_2$, and $dst = dist(q_1, o_8) + dist(q_2, o_8) + dist(q_3, o_6) = 11.3$. Since $dst < dst_2$, we insert $o_8$ into $E_2$ (i.e., $\{o_5, o_4, o_8\}$), push $\langle o_8, 11.3\rangle$ with each current $E_i$ into $UP$, and update $H = \{\langle q_1, 4.5\rangle, \langle q_3, 5.5\rangle, \langle q_2, 4.8\rangle\}$. Next, we obtain $q_1$'s next NN $o_1$ from $R_1$ and $dst = dist(q_1, o_1) + dist(q_2, o_5) + dist(q_3, o_6) = 11$. Now, we have $E_1 = \{o_5, o_6, o_8, o_4, o_1\}$, push $\langle o_1, 11, E_1, E_2, E_3\rangle$ into $UP$, and update $H = \{\langle q_2, 4.8\rangle, \langle q_3, 5.5\rangle, \langle q_1, 6\rangle\}$. Then, we obtain $q_2$'s next NN $o_6$ and $dst = dist(q_1, o_6) + dist(q_2, o_6) + dist(q_3, o_6) = 10.5$. Because $dst < dst_2$ and $o_6 \in \cap_{i=1}^{3} E_i$, we discard $o_4$ from $S$ (i.e., $\{o_5, o_6, o_8\}$) and update $CO = \{\langle o_5, 7.5\rangle, \langle o_6, 10.5\rangle\}$; we now have $dst_2 = 10.5$. Since there is an entry in $UP$ with $dst > dst_2$, e.g., $\langle o_8, 11.3, E_1, E_2, E_3\rangle$, we can obtain $E'_1$, $E'_2$ and $E'_3$ in accordance with the corresponding object and $E_i$ of that entry: $E'_1 = \{o_5, o_6\}$, $E'_2 = \{o_5, o_4\}$, and $E'_3 = \emptyset$ such that $S' = \{o_4, o_5, o_6\}$ and $S \cap S' = \{o_5, o_6\}$. Thus, the verified objects $o_5$ and $o_6$ are the top-2 CGNNs.

*Correctness and complexity*. By Theorem 2, the set $A_q$ of top-$j$ CGNN objects surely belongs to $S$. Thus, only objects that cannot be in $A_q$ are pruned by Theorem 3, and we will obtain the accurate top-$j$ objects in response to a CGNN query because the entire verification process of Algorithms 3 and 4 corresponds to the expansion of the set $R_i$ of objects within $\epsilon$ for each query point, requiring at most $|R|$ operations.

## 4 Algorithms for MCGNN Queries

Compared to CGNN queries, although MCGNN queries appear to only additionally takes into account the ratings of POIs. However, as illustrated in Example 1, the filtering-and-verification framework proposed for CGNN queries cannot completely solve MCGNN queries. Therefore, this section presents a brand new set of solutions to replace the verification techniques in Section 3.4, even the incremental and accumulative expansion strategy in Section 3.3.

Given $c$ attendees from $G_s^{k_{max}}(u_q, c)$ and the corresponding query points $Q$, a baseline method to complete the evaluation of a MCGNN query, retrieving the top-$j$ ranked objects in descending order of their overall scores, is to sort $dist_{sum}(o, Q)$ (in ascending order) and average rating $\overline{R_s}(G_s^{k_{max}}(u_q, c), o)$ (in descending order) of each object $o$ from $R$ in Section 3.3 into lists to find those with the $k$ greatest overall scores. In this section, we propose two methods that find top-$j$ objects more efficiently according to lists, and another two algorithms that solve MCGNN queries by effectively minimizing network traversal.

### 4.1 Threshold Selection Strategies

The baseline method of MCGNN above needs to traverse all the sorted lists and rerank the overall score results for each object to report the desired top-$j$ objects with greatest overall scores. However, using these individual orderings, we can perform much better in evaluating MCGNN queries by eliminating ergodic matching and combined score ranking. Thus, two more efficient methods are proposed.

*Threshold selection (TS)*. The main difference between *TS* and the baseline method of MCGNN is the stopping mechanism which decides when to stop doing sorted access to the lists. In sorted access, objects are accessed sequentially ordered by the scoring predicate, while for random access, objects are directly accessed by their identifiers. Random access allows for obtaining the overall score of an object right after it appears in one of lists. The stopping mechanism of *TS* uses a threshold which is computed using the last local scores seen under sorted access in the lists. *TS* works as follows:

1) Do sorted access in parallel to each of the $m$ sorted lists. As an object $o$ is seen under sorted access in one list, do random access to the other lists to find the local score of $o$ and compute its overall score. Maintain in a set $A_q$ the $j$ seen objects whose overall scores are the greatest among all objects seen so far.
2) For each list $L_i$, let $s_i$ be the last local score seen under sorted access in list $L_i$. Define the threshold to be $\delta = t(s_1, s_2, \ldots, s_m)$[5]. If $A_q$ involves $j$ objects with overall scores greater than or equal to $\delta$, then halt and return $A_q$. Otherwise, go to 1.

**Theorem 4.** *Using a monotone combining scoring function,* TS *correctly finds the top-$j$ answers.*

**Proof.** We need only show that every member of $A_q$ has an overall score at least as great as every object $z$ not in $A_q$. By definition of $A_q$, this is the case for each object $z$ that has been seen in running *TS*. So assume that $z$ was not seen and its local scores are $z_1, z_2, \ldots, z_m$. Therefore, the position of $z_i$ is greater than or equal to that of $s_i$ for every $i$. Hence, $t(z_1, z_2, \ldots, z_m) \leq t(s_1, s_2, \ldots, s_m) = \delta$, where the inequality follows by monotonicity of $t$. However, for each $o$ in $A_q$ we have overall score of $o$ greater than or equal to $\delta$. Thus, *TS* is correct. □

To solve MCGNN queries, let $s_1$ (resp. $s_2$) be the last local score seen under sorted access in list $L_1$ of $dist_{sum}(o, Q)$ (resp. $L_2$ of average rating), and threshold $\delta$ be $s_2/(\alpha \cdot s_1)$.

| $o$ | $dist_{sum}$ |
|---|---|
| $o_1$ | 5.0 |
| $o_2$ | 6.0 |
| $o_5$ | 6.5 |
| $o_3$ | 7.0 |

| $o$ | $\overline{R_s}$ |
|---|---|
| $o_3$ | 5.0 |
| $o_1$ | 4.8 |
| $o_4$ | 4.8 |
| $o_2$ | 4.5 |

Fig. 6. List $L_1$ of $dist_{sum}$ and List $L_2$ of average rating.

**Example 7.** Given a set $U$ of $c$ attendees, consider two lists $L_1$ and $L_2$ holding different rankings for the same set $R$ of objects based on two scoring predicates $dist_{sum}$ and $\overline{R_s}$, respectively. Fig. 6 depicts a partial lists where $\overline{R_s}$ produces score values in the range $[0, 5.0]$. Suppose we use the overall scoring function $\theta(o, U)$ in Equation 3 to find a top-1 object $o$, where $\alpha = 1$. *TS* first looks at the objects at position 1 in both lists and looks up their local scores in another list by random access to compute overall scores, i.e., $\theta(o_1, U) = 4.8/5, \theta(o_3, U) = 5/7$. But neither of them is as great as the threshold $\delta = 5/5$. Thus, *TS* does not halt at position 1 with $A_q = \{o_1\}$. At position

5. A combining scoring function $t$ is monotonic if $t(x_1, x_2, \ldots, x_m) \geq t(x'_1, x'_2, \ldots, x'_m)$ whenever the position of $x_i$ is less than or equal to that of $x'_i$ for every $i$.

2, $A_q$ still involves $\{o_1\}$ but now $\delta = 4.8/6$ is less than $\theta(o_1, U)$. So *TS* halts at position 2 and return $A_q = \{o_1\}$.

*Threshold ripple selection (TRS).* As random access could incur a large overhead (i.e., each sorted access in *TS* results in up to $m-1$ random accesses), *TRS* modifies threshold $\delta$ to better suit the new access strategy (i.e., with sorted access only) as opposed to *TS*, which works as follows:

1) For each list $L_i$, do $x_i$ sorted accesses in parallel to each of the $m$ sorted lists. As an object $o$ is seen in every $L_i$, compute its overall score. Maintain in a set $A_q$ the $j$ seen objects whose overall scores are the greatest among all objects seen so far.
2) Let $s_i^{n_i}$ be the last local score seen from $L_i$, where $n_i$ is the number of objects retrieved from that list, $s_i^1$ is the first local score in $L_i$. Define threshold to be $\delta = \max\{t(s_1^{n_1}, s_2^1, \ldots, s_m^1), t(s_1^1, s_2^{n_2}, \ldots, s_m^1), \ldots, t(s_1^1, s_2^1, \ldots, s_m^{n_m})\}$. If $A_q$ involves $j$ objects with overall scores greater than or equal to $\delta$, then halt and return $A_q$. Otherwise, go to 1.

**Theorem 5.** *Using a monotone combining scoring function,* TRS *correctly finds the top-$j$ answers.*

**Proof.** We need only show that every member of $A_q$ has an overall score at least as great as every object $z$ not in $A_q$. So assume that $z$ was not seen in every $L_i$ when *TRS* halts and the local score of $z$ are $z_1, z_2, \ldots, z_m$. Therefore, there exists at least one position of $z_i$ greater than or equal to that of $s_i^{n_i}$, e.g., $i = m$. Hence, $t(z_1, z_2, \ldots, z_m) \leq t(z_1, z_2, \ldots, s_m^{n_m}) \leq t(s_1^1, s_2^1, \ldots, s_m^{n_m}) \leq \delta$, where the inequality follows by monotonicity of $t$. However, for each $o$ in $A_q$ we have overall score of $o$ greater than or equal to $\delta$. Thus, *TRS* is correct. □

To solve MCGNN queries, let $s_1^{n_1}$ (resp. $s_2^{n_2}$) be the last local score seen in list $L_1$ of $dist_{sum}(o, Q)$ (resp. $L_2$ of average rating), and threshold $\delta$ be $\max\{s_2^1/(\alpha \cdot s_1^{n_1}), s_2^{n_2}/(\alpha \cdot s_1^1)\}$. Obviously, $n_i$ is a multiple of $x_i$, and *TRS* performs best when $x_1 = x_2$, which is explained in the experiments.

**Example 8.** Compared to *TS* in *Example* 7, suppose we access two objects from each list by *TRS*, i.e., $x_1 = x_2 = 2$. At the first stage, only object $o_1$ is seen in every list with overall score $\theta(o_1, U) = 4.8/5$, while the threshold $\delta$, i.e., $\max\{5/6, 4.8/5\}$, is equal to $\theta(o_1, U)$. So *TRS* halts here and returns $A_q = \{o_1\}$.

### 4.2 Threshold Expansion

The threshold expansion (TE) is based on the observation that the network traversal from each $q_i \in Q$ visits the vertices in increasing order of their network distances from $q_i$. Thus, MCGNN queries can be answered by concurrently and incrementally expanding the network around each $q_i \in Q$ and applying some top-$j$ aggregate query processing techniques [5], [6], [7] to guide and terminate the search early.

Algorithm 5 shows the pseudo-code of TE. Given a set $U$ of $c$ attendees and corresponding $L_s(U)$, TE computes $\theta(o, U)$ first for all objects on the edge $(n_l, n_r)$ containing $q_i \in L_s(U)$, where $n_l$ and $n_r$ are added to a priority queue $H$ storing tuples, e.g., $\langle n_l, dist(q_i, n_l), q_i \rangle$. $H$ keeps vertices $n_l$ visited by $q_i$ ordered on $dist(q_i, n_l)$. Thus, its top element corresponds to the next nearest vertex from any $q_i$. TE iteratively pops vertices from $H$, computes $\theta(o, U)$ on the adjacent edges, and adds the adjacent vertices that have not been visited from the same $q_i$ before to $H$. During the process, $j$-MCGNNs retrieved so far are maintained.

**Algorithm 5:** TE$(U, R_s, \epsilon)$

**Input**: $U$ : $c$ attendees; $R_s$ : rating records; $\epsilon$ : network distance threshold
**Output**: MCGNNs

```
1  H := ∅; θ_j := 0;
2  foreach q_i ∈ L_s(U) on (n_l, n_r) do
3      Compute θ(o, U) if ∀o on (n_l, n_r), dist_max(o, L_s(U)) ≤ ε;
4      Update top-j results and θ_j if necessary;
5      H.enqueue(⟨n_l, dist(n_l, q_i), q_i⟩, ⟨n_r, dist(n_r, q_i), q_i⟩);
6  while H ≠ ∅ do
7      ⟨n, dist, q⟩ ← H.dequeue();
8      if R̂_s / (α · dist) ≤ δ then terminate;
9      if n is not visited by q before then
10         foreach n_i ∈ Adj[n] do
11             if (n, n_i) is populated and not checked then
12                 Compute θ(o, U) if ∀o on (n, n_i),
                   dist_max(o, L_s(U)) ≤ ε;
13                 Update top-j results and θ_j if necessary;
14             H.enqueue(⟨n_i, dst, q⟩) if dst = dist + w(n, n_i) ≤ ε;
```

Let $\theta_j$ be the $j$-th greatest overall score of objects found. TE terminates when $\frac{\hat{R}_s}{\alpha \cdot \underline{dist}}$ is no greater than a threshold $\delta$, i.e., $\theta_j \cdot c$, where $\hat{R}_s$ denotes the upper limit of $\overline{R_s}$ range and $\underline{dist}$ is the network distance to the next vertex popped from $H$. If this condition is met, no unexamined edge can contain better solutions as guaranteed by the following theorem:

**Theorem 6.** *Given a set $U$ of $c$ attendees, let $(n_l, n_r)$ be an edge that does not contain any query point $q_i$. If $\forall q_i \in Q$, $\frac{\hat{R}_s}{\alpha \cdot \underline{dist}} \leq \delta$, then for any object $o$ on $(n_l, n_r)$, $\theta(o, U) \leq \delta / c$.*

**Proof.** Since no $q_i$ lies on $(n_l, n_r)$, the shortest path from each $q_i$ to any $o$ on the edge should pass either $n_l$ or $n_r$. Thus, $\forall q_i, dist(q_i, o) \geq \min\{dist(q_i, n_l), dist(q_i, n_r)\} = \underline{dist}, dist_{sum}(o, Q) \geq c \cdot \underline{dist}$. Moreover, $\hat{R}_s$ is always no less than $\overline{R_s}(U, o)$, as a result, $\theta(o, U) \leq \frac{\hat{R}_s}{\alpha \cdot c \cdot \underline{dist}} \leq \delta / c$. □

**Example 9.** In Fig. 1, given $\epsilon = 4.5$ and $\omega = \{\text{“}cafe\text{”}\}$, suppose that $q_1$ is on $(r_7, r_{13})$ with $w(q_1, r_{13}) = 1$ and $q_2$ is on $(r_7, r_{15})$ with $w(q_2, r_{15}) = 1.5$. Assume we want to find the 1-MCGNN by TE with the overall scoring function in Equation 3 (e.g., $\alpha = 1$). Since $(r_7, r_{15})$ is populated, TE applies shortest path queries to find $dist(q_1, r_7)$ and $dist(q_1, r_{15})$, and the overall score of $o_5$ can be computed that becomes the current 1-MCGNN with $\theta(o_5, U) = \frac{3.7}{4.5} = 0.82$ ($\delta = 1.64$). Now $H$ is initialized as $\{\langle r_{13}, 1, q_1 \rangle, \langle r_{15}, 1.5, q_2 \rangle, \langle r_7, 2, q_1 \rangle, \langle r_7, 2.5, q_2 \rangle\}$. Entries are popped in order, rejecting $o_4$ as $\theta(o_4, U) < 0.82$, until $\langle r_{11}, 4, q_1 \rangle$ is dequeued, TE terminates ($\frac{\hat{R}_s}{4} \leq \delta$).

### 4.3 Holistic Expansion

Unlike TE, where shortest path queries of all component network distances are performed to compute overall scores of objects once a populated edge is visited, the holistic expansion (HE) waits until the edge has been seen from all $q_i$. Therefore, HE avoids shortest path computations, that could incur a large overhead (e.g., random I/Os) since traversal of network vertices is carried out in a relatively unsystematic manner.

HE maintains a set $P$ of populated edges that have been visited from some $q_i$ and may contain objects with

**Algorithm 6:** HE($U, R_s, \epsilon$)

**Input**: $U$ : $c$ attendees; $R_s$ : rating records; $\epsilon$ : network distance threshold
**Output**: MCGNNs

```
1  H := ∅; θ_j := 0; P := ∅;
2  foreach q_i ∈ L_s(U) on (n_l, n_r) do
3      if (n_l, n_r) is populated then add (n_l, n_r) to P;
4      H.enqueue(⟨n_l, dist(n_l, q_i), q_i⟩, ⟨n_r, dist(n_r, q_i), q_i⟩);
5  while H ≠ ∅ do
6      ⟨n, dist, q⟩ ← H.dequeue();
7      if (P = ∅ ∧ R̂_s / (α · dist) ≤ δ) then terminate;
8      if n is not visited by q before then
9          foreach n_i ∈ Adj[n] do
10             if n_i is not visited by q before then
11                 H.enqueue(⟨n_i, dst, q⟩) if
                   dst = dist + w(n, n_i) ≤ ε;
12             if ∀q_i, (n ∨ n_i is visited by q_i) ∨ (q_i on (n, n_i)) then
13                 Compute θ(o, U) if ∀o on (n, n_i),
                   dist_max(o, L_s(U)) ≤ ε;
14                 Update top-j results and θ_j if necessary;
15             if (n, n_i) is populated then
16                 if ub(n, n_i) ≥ θ_j then add (n, n_i) to P;
17                 if (n, n_i) ∈ P ∧ ((ub(n, n_i) < θ_j) ∨ (ub(n, n_i) ≥ θ_j
                   is checked c times)) then
18                     delete (n, n_i) from P
```

overall score no smaller than $\theta_j$. Algorithm 6 shows the pseudo-code of HE, where Line 12 checks whether $n$ or $n_l$ has been visited from all $q_i$, so that the lower bound of overall score[6] for each object on $(n, n_l)$ can be derived. Thus, HE can terminate if (1) $P = \emptyset$ and (2) $\frac{\hat{R}_s}{\alpha \cdot \underline{dist}}$ is no greater than the threshold $\delta$ (i.e., $\theta_j \cdot c$). The former ensures we have visited and eliminated all edges that may contain a better solution in view of the component network distances available. When an adjacent edge $(n, n_i)$ is visited from the currently popped vertex $n$, we add it to $P$ if it is populated and $ub(n, n_i)$, the upper bound of any possible object on it (see Section 4.3.1), is no smaller than $\theta_j$. To compute $ub(n, n_i)$, for each $q_i$, if $n$ (resp. $n_i$) has been visited by $q_i$, we use the actual $dist(q_i, n)$ (resp. $dist(q_i, n_i)$); otherwise we use a lower bound which is equal to the last network distance popped from $H$ (i.e., $\underline{dist}$). If $(n, n_i)$ is already in $P$, but now $ub(n, n_i) < \theta_j$ or $ub(n, n_i) \geq \theta_j$ which has been checked $c$ times, we remove it from $P$; no object on $(n, n_i)$ can be the top-$j$. Therefore, $P$ initially grows as $\theta_j = 0$ and shrinks later as $\theta_j$ becomes tighter. As for the latter, due to Theorem 6, there exists no better object than current solution. When the two conditions are met, HE terminates with the correct top-$j$ answers.

**Example 10.** Compared to TE in *Example* 9, $(r_7, r_{15})$ and $(r_{13}, r_{15})$ are added to $P$ as $\theta_1 = 0$. When $\langle r_{15}, 1.5, q_2 \rangle$ is dequeued, we can compute $\theta(o_4, U) \geq 0.61$ (i.e., $\theta_1$) since $r_{13}$ (resp. $r_{15}$) is visited by $q_1$ (resp. $q_2$). Next, $\langle r_7, 2, q_1 \rangle$ is popped, the actual 1-MCGNN $o_5$ with $\theta(o_5, U) = 0.82$ is found. Finally, HE terminates when $P$ becomes empty.

Objects on an edge are checked at most $c$ times and a vertex can be enqueued at most $c$ times.

#### 4.3.1 Upper Bound for Edges

Given a set $U$ of $c$ attendees and an edge $(n_l, n_r)$ with the component network distances from each $q_i \in L_s(U)$ to $n_l$ and $n_r$, we can compute the maximum possible $\theta(o, U)$ for any object $o$ on $(n_l, n_r)$. It is practical while solving MCGNN queries such that we can prune the populated edge without computing $o$'s actual overall score if it cannot contain any better solution. Thus, we study the possible range of $\theta(o, U)$. As $\overline{R_s}(U, o)$ may range from 0 to $\hat{R}_s$, now we discuss how the variation of $dist(q_i, o)$ affects $dist_{sum}(o, L_s(U))$ depending on (1) whether $q_i$ lies on $(n_l, n_r)$ and (2) the network distances from $q_i$ to $n_l$ and $n_r$. There are three kinds of network distance distributions for $dist(q_i, o)$, which appear as piecewise linear functions as $o$ moves from one end vertex to the other (e.g., $n_l$ to $n_r$). When $q_i$ lies on $(n_l, n_r)$, $dist(q_i, o)$ decreases first and then increases linearly; otherwise, it is divided into two cases: (1) $|dist(q_i, n_l) - dist(q_i, n_r)| = w(n_l, n_r)$, and (2) $|dist(q_i, n_l) - dist(q_i, n_r)| < w(n_l, n_r)$. The former indicates that the shortest path from $q_i$ to one end vertex passes through the other, and $dist(q_i, o)$ increases or decreases linearly and monotonically. The latter reveals that $dist(q_i, o)$ increases first and then decreases linearly.

In this way, we can find a sequence of splitting points on $(n_l, n_r)$ consisting of the end vertices and the extreme points of $dist(q_i, o)$ for each $q_i \in L_s(U)$. Thus, the position on $(n_l, n_r)$ can be determined by using splitting points such that $dist_{sum}(o, L_s(U))$ is minimum for any $o$ on the edge. In other words, the lower bound of $dist_{sum}(o, L_s(U))$ can be found by computing only the aggregate network distances at the splitting points. It is noting that the corresponding aggregate network distances between two adjacent splitting points can be kept constant, increasing or decreasing linearly. As a result, we can define the upper bound of $\theta(o, U)$ for any object $o$ on $(n_l, n_r)$ using $\hat{R}_s$ and the lower bound of $dist_{sum}(o, L_s(U))$.

## 5 Optimization Methods

In this section, we present three optimization methods to improve both CGNN and MCGNN query processing performance. Section 5.1 presents an efficient algorithm for reducing the search space before finding the most cohesive $k$-core. Section 5.2 develops a road network index, based on which we can progressively find the closest object candidates from each query point. Section 5.3 introduces a round-robin technique for increasing the overall query efficiency.

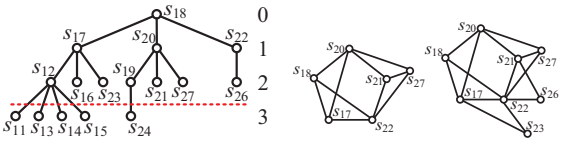
### 5.1 Social-distance-based Pruning (SD)

The pruning effect with $k$-core decomposition [10] alone is not significant. If we can identify vertices that must not appear in the results, it may greatly reduce the search space and computational complexity. We assume that the coreness of the most cohesive $k$-core must be no less than 2; otherwise, retrieving only $G_s^1(u_q, c)$ becomes less meaningful. From this perspective, we regard the shortest social distance as the minimum number of edges between two vertices.

**Theorem 7.** *Given a social network $G_s$, $v \in V_s$ must not be contained in $G_s^{k_{max}}(u_q, c)$ if the shortest social distance from $v$ to $u_q$ is no less than $c-1$.*

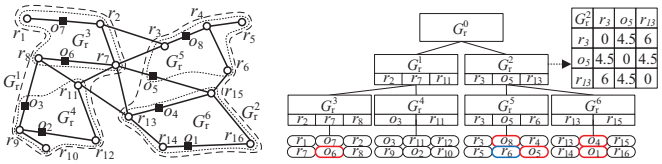
Based on Theorem 7, a social distance pruning strategy is proposed to refine the cardinality of the potential candidate set by eliminating more distant vertices before finding the most cohesive $k$-core. Given a query user $u_q$ and a constant

6. Note that the aggregate network distance can be improved since the edge can be later visited again via another path.



(a) BFS tree rooted at $s_{18}$. (b) $G_s^3$ and $G_s^2$.

Fig. 7. Social-distance-based pruning.



(a) Graph partition. (b) Index tree $I_{RN}$.

Fig. 8. Road network index.

$c$, we construct a breadth-first search (BFS) tree rooted at $u_q$; then, vertices at a tree height of at least $c-1$ can be pruned directly (the tree height is 0 at the root). When this pruning strategy is used to delete vertices, the degree of the remaining vertices in the underlying social network changes, and more vertices can then be pruned through $k$-core decomposition [10].

**Example 11.** Suppose that we wish to find $G_s^{k_{max}}(s_{18}, 4)$, a BFS tree rooted at $s_{18}$, as shown in Fig. 7(a). Vertices with a tree height of no less than 3, i.e., $s_{11}$, $s_{13}$, $s_{14}$, $s_{15}$, and $s_{24}$, can be pruned by Theorem 7. Furthermore, $s_{12}$ can be pruned via $k$-core decomposition since its degree falls from 5 (in Fig. 2) to 1. Finally, $G_s^3$ and $G_s^2$, both containing $s_{18}$, are obtained as shown in Fig. 7(b).

### 5.2 Road-network-based Indexing

To compute network distance more efficiently, a road network index $I_{RN}$ is adopted. We first construct a new road network $G_r' = (V_r', E_r')$ from $G_r$ as follows. Let $V_r' = V_r \cup O$. We add all edges in $E_r$ containing no object in $O$ to $E_r'$ with the same weight. The other edges in $E_r$ are subdivided into multiple sub-edges depending on the number of objects; the weights of resulting sub-edges are proportional.

*Index structure.* We adopt a state-of-the-art hierarchical tree structure [8] to index $G_r'$. A balanced search tree $I_{RN}$ satisfies the following properties. (1) Each node of $I_{RN}$ represents a subgraph (root corresponds to $G_r'$). The subgraph represented by a parent node is a supergraph of those represented by its child nodes. All subgraphs at the same level of $I_{RN}$ compose a partition of $G_r'$, i.e., any two subgraphs are disjoint and their union is $G_r'$. (2) Each nonleaf node has $f(\geq 2)$ children. (3) Each leaf node contains at most $\tau(\geq 1)$ vertices. All leaf nodes appear at the same level. (4) Each node has a border set (i.e., boundary vertices of a partition) and a distance matrix. In the distance matrix, for a non-leaf node (resp. leaf node), the columns/rows are all borders of its children (resp. columns are all vertices for this node), and the value of each entry is the corresponding network distance. (5) The occurrence list (i.e., $\mathcal{L}(n)$) for a leaf node is the list of objects in this node; for a nonleaf node, it is the list of its children that contain objects.

**Example 12.** Fig. 8(b) shows the tree $I_{RN}$ for a road network $G_r$. Each nonleaf node corresponds to a subgraph of $G_r$; e.g., $G_r^2$ corresponds to the right subgraph in Fig. 8(a). The borders of each node are shown in the rectangular boxes under that node. For instance, $G_r^2$ has 3 borders $\{r_3, o_5, r_{13}\}$, and its distance matrix is listed beside it. The vertices of each leaf node are shown in rounded rectangles; e.g., $G_r^6$ has 2 borders $\{r_{13}, r_{15}\}$ and 6 vertices $\{r_{13}, o_4, r_{15}, r_{14}, o_1, r_{16}\}$.

While traversing $I_{RN}$, we use a priority queue $PQ_i$ to maintain the nearest objects to each $q_i \in Q$. First, we locate the leaf node of $q_i$ and objects $O^*$ with leaf($n$) and construct the occurrence list in a bottom-up manner. Initially, we calculate the network distance from every object $o \in \mathcal{L}(\text{leaf}(q_i))$ to $q_i$, and place $\langle o, dist(q_i, o)\rangle$ in $PQ_i$. Next, we iteratively dequeue the first element $\langle e, dis\rangle$ of $PQ_i$ and address it separately according to whether $e$ is an object or a node. This process continues until an element with a $dis$ greater than $\epsilon$ is dequeued from $PQ_i$, indicating that all objects within the network distance threshold $\epsilon$ in $R_i$ have been retrieved. The same condition applies for a directed road network.

**Example 13.** Continuing *Example* 4, the range search process based on $I_{RN}$ is as follows. First, we construct the occurrence list $\mathcal{L}$ based on $\{o_1, o_4, o_5, o_6, o_8\}$, access $G_r^5 = \text{leaf}(r_6)$ as the current node pointer $T_n$, and determine the minimum network distance $T_{min}$ from $q_1$ to each border within $T_n$; here, $T_{min} = 0$ since $r_6$ is a border. Next, we push $dist(r_6, o_5) = 3.5$ and $dist(r_6, o_8) = 3.8$ into $PQ_1$ as $o_5, o_8 \in \mathcal{L}(G_r^5)$. Now, because $3.5 > T_{min}$, we update $T_n$ to $G_r^2$ and $T_{min} = dist(r_6, G_r^2) = 3.5$ and push $dist(r_6, G_r^6) = 1$ into $PQ_1$ as $G_r^6 \in \mathcal{L}(T_n)$. Then, we access $G_r^6$, and push $dist(r_6, o_4) = 5.2$ and $dist(r_6, o_1) = 7.5$ into $PQ_1$ as $o_4, o_1 \in \mathcal{L}(G_r^6)$. We now insert $\langle o_5, 3.5\rangle$ into $R_1$ because $3.5 \leq T_{min}$. Because $dist(r_6, o_8) > T_{min}$, $T_n$ is updated to $G_r^0$, $T_{min} = \infty$, and $dist(r_6, G_r^1) = 5$ is pushed into $PQ_1$ as $G_r^1 \in \mathcal{L}(T_n)$. Finally, we insert $\langle o_8, 3.8\rangle$ into $R_1$ and terminate the process because $dist(r_6, G_r^1) > \epsilon$.

### 5.3 Round-robin Optimization

As illustrated in Section 3.2, the CE algorithm places every possible expansion into a queue, and both the *Lc* and *Li* strategies may be repeatedly executed until the top-$j$ objects are found for the current user group in a CGNN/MCGNN query. If any selected user can be identified as unlikely to be included in the most cohesive $k$-core, the search space can be reduced in both the social and spatial domains, and the overall query performance can be improved. In this section, we first deduce an intrinsic distance restriction between query points and then apply this restriction to the maximal $f(\cdot)$ value in the heuristics before a tie in social distance can arise. The procedure is repeated in a round-robin fashion; in other words, the intrinsic distance restriction is re-applied w.r.t. each selected vertex until the valid most cohesive $k$-core is obtained. On the one hand, this allows us to filter the invitees to reduce the diversity of the generated candidates, eliminating the need to find and verify objects for $G_s^{k_{max}}(u_q, c)$ that are qualified in terms of social connections but not locations. On the other hand, the intrinsic distance restriction consolidates the attendees such that the objects of interest will be relatively close; thus, both the acquisition of the candidate set $S$ and the network expansion are sped up.

**Theorem 8.** *The network distance between the locations of each pair of users from $G_s^{k_{max}}(u_q, c)$ in road network $G_r$ must be less than or equal to $2\epsilon$.*

TABLE 2
Statistics of the Datasets

| Dataset | Vertices | Edges | $dg_{avg}$ | $dg_{max}$ | $h$ |
|---|---|---|---|---|---|
| California | 21,048 | 21,693 | 2.06 | 8 | - |
| San Francisco | 174,956 | 223,001 | 2.55 | 8 | - |
| Florida | 1,070,376 | 1,356,399 | 2.53 | 12 | - |
| Western USA | 6,262,104 | 7,624,073 | 2.43 | 14 | - |
| Facebook | 4,039 | 88,234 | 43.69 | 1,045 | 8 |
| Brightkite | 58,228 | 214,078 | 7.35 | 1,134 | 17 |
| Gowalla | 196,591 | 950,327 | 9.67 | 14,730 | 15 |
| Orkut | 3,072,441 | 117,185,083 | 76.28 | 33,313 | 7 |
| Twitter | 17,069,982 | 476,553,560 | 55.84 | 109,214 | 8 |

**Example 14.** Continuing *Example* 3, if $\epsilon = 4$, we can calculate $dist(r_3, r_9) = 9 > 2\epsilon$ from $I_{RN}$ because $s_{21}$ and $s_{26}$ are at $r_3$ and $r_9$, respectively. By Theorem 6, $H_2$ in Fig. 4 can be discarded via *Lc* or *Li* instead of during verification. Thus, only $G_s(H_1)$ is a valid solution for the most cohesive $k$-core.

## 6 Experimental Evaluation

This section evaluates the effectiveness and efficiency of our algorithms for both CGNN and MCGNN queries through comprehensive experiments.

### 6.1 Experimental Setting

*Datasets.* Five real-life social networks[7], Facebook (FB), Brightkite (BR), Gowalla (GO), Orkut (OR) and Twitter (TW), and four real road networks[8,9], California (CA), San Francisco (SF), Florida (FL) and Western USA (WU), are investigated in our experiments. CA and SF contain detailed street networks, whereas FL and WU consist only of highways and main roads. The statistics of these datasets are presented in Table 2, where $h$ denotes the average longest social distance.

As only CA has POI information with category name (e.g., hospital/school), we additionally map the real-world POIs and associated keywords from [14] to SF, FL and WU, which are extracted from OpenStreetMap[10]. The rating information[11] is extracted from Foursquare website, where social network user can be tagged a few ratings of POIs. In addition, we map each user $v_s$ to the location $v_r$ in the road network that matches the scale of his/her social network as follows: We first project the spatial locations into the range $[0, 1]$ in each dimension, and we generate $L_s(v_s)$ randomly (or by drawing from recent check-ins). If $v_r$ has the smallest Euclidean distance to $L_s(v_s)$ in the projection space, we assume that $v_r$ is the current location of $v_s$.

*Algorithms.* To our knowledge, no previous works have investigated the MCGNN problem on general road-social networks while [15] only investigated the CGNN problem. In this paper, we implement and evaluate two algorithms for CGNN queries and five algorithms for MCGNN queries as described in Table 3. Since the CE algorithm introduced in Section 3.2 is relatively slow even on a small road-social network, as illustrated by Exp-1 and Exp-2, we employ the optimal heuristic selection strategy *Li* as the baseline algorithm in this empirical study on the problem of finding the most cohesive $k$-core and retrieving the top-$j$ objects with or without multi-criteria, respectively. In Table 3, we also define the abbreviation for each technique used in the considered algorithms.

7. http://snap.stanford.edu/data/index.html
8. https://www.cs.utah.edu/ lifeifei/SpatialDataset.htm
9. http://www.dis.uniroma1.it/challenge9/index.shtml
10. http://www.openstreetmap.org
11. https://archive.org/details/201309_foursquare_dataset_umn

TABLE 3
Summary of Algorithms

| Technique | | Description |
|---|---|---|
| SD | | social-distance-based pruning (Theorem 7) |
| $\epsilon$NN | | range search for nearest neighbors within a threshold $\epsilon$ by the road network index (Section 5.2) |
| IR | | intrinsic distance restriction between users in $Q$ (Theorem 8) |
| **Query** | **Algorithm** | **Description** |
| CGNN query | CGNN | consists of *Li* strategy (Equation 5) with SD, range filter (Algorithm 2), and verification techniques (Theorems 2 and 3) |
| | CGNN-opt | CGNN+$\epsilon$NN+IR; incorporates IR into *Li* strategy for alternative validation, and objects within $\epsilon$ are computed with $\epsilon$NN strategy instead of range filter (Algorithm 2) adopted in CGNN |
| MCGNN query | TRS | consists of *Li* with SD and IR, and the threshold ripple selection (Theorem 5) |
| | TE | consists of *Li* with SD and IR, and the threshold expansion (Theorem 6 and Algorithm 5) |
| | HE | consists of *Li* with SD and IR, and the holistic expansion (Algorithm 6) |
| | TE-opt | the network distances from any $q_i$ to vertices/objects are computed over $I_{RN}$ |
| | HE-opt | $dist_{sum}(o, Q)$'s lower bound is computed over $I_{RN}$ in addition to the network distances from any $q_i$ to vertices/objects |

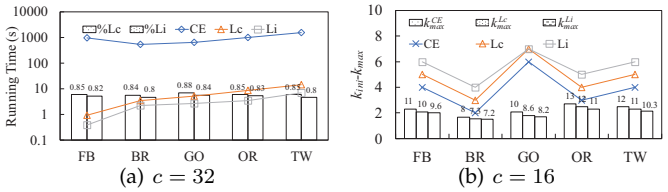
*Parameters.* We conducted experiments in different settings by varying 7 parameters, including the number of attendees $c$, the degree distribution of query user $u_q$, the network distance threshold $\epsilon$, the number of qualified objects $j$, the object ratio (number of objects to number of edges), the tradeoff $\alpha$, and the number of sorted accesses performed at one time (i.e., $x_1$ and $x_2$, in the threshold ripple selection). The default values of $c$, $j$ and $\alpha$ were 4, 3 and 1, respectively, and both $x_1$ and $x_2$ were equal to 3; in CA, SF, FL and WU, the default values of $\epsilon$ were 1, 4, 40 and 150 km, and the default object ratios were 0.1, 0.1, 0.01 and 0.01, respectively.

All programs were implemented in standard C++ and compiled with G++ in Linux. All experiments were performed on an Ubuntu Linux System with an Intel Xeon E7-4820 2 GHz CPU and 1 TB of memory. The query user $u_q$ was randomly assigned 25 times in each vertex degree interval (totally 4 in Exp-3), then algorithms were tested for each value of different parameters (with the others set to defaults) respectively. The results shown in each experiment were the average of 100 independent tests. Due to the space limitation, we present more detailed experimental results and discussions in the supplemental material.

### 6.2 Performance Evaluation for CGNN Queries

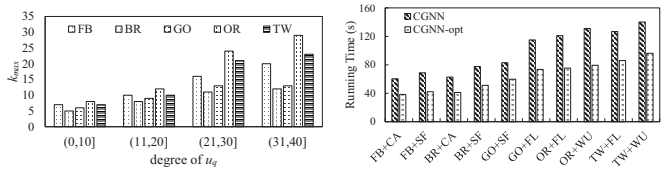
We investigate the efficiency of two algorithms, CGNN and CGNN-opt, for CGNN queries listed in Table 3, and then compare each under different settings.

*Exp-1*: We evaluated the exact algorithm and the two heuristics for finding the most cohesive $k$-core described in Section 3.2, namely, CE, *Lc* and *Li*, on the five social networks for $c = 32$. The accuracy (%*Lc* or %*Li*) is 1 if the entire user group obtained by *Lc*/*Li* is included in the CE



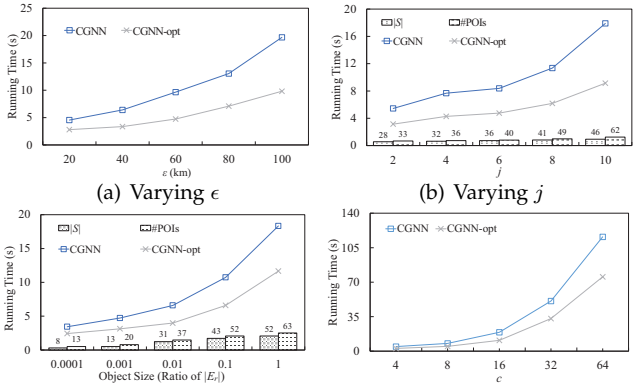

(a) $c = 32$ (b) $c = 16$

Fig. 9. Query performance and accuracy between CE and heuristics.




(a) Varying $u_q$'s degree interval (b) Scalability ($c = 64$)

Fig. 10. $k_{max}$ with degree distribution and scalability for CGNN queries.




(a) Varying $\epsilon$ (b) Varying $j$

(c) Varying object size (d) Varying $c$

Fig. 11. Query performance for CGNN queries (datasets: OR+FL).

results; otherwise, it is proportional to the number of users included. Fig. 9(a) indicates that CE is three to four orders of magnitude slower than the other two methods because CE enters the next iteration if it fails in the current round, while *Lc*/*Li* ends once the size expands to $c$. Although FB is small, CE takes more time on it than on BR or GO because SD depends on the social distance and $c$. As shown in Table 2, $h = 8$ in FB, which is sufficient to support good SD performance only for $c \leq 9$. The average accuracies of *Lc* and *Li* are essentially stable at approximately 0.9, but *Li* is nearly an order of magnitude faster. Thus, *Li* is selected for all the algorithms listed in Table 3.

*Exp-2*: We examined $k_{ini} - k_{max}$, representing the difference between $cn(C_{max}(u_q, c))$ and the coreness of the attendees, with respect to various social networks. Fig. 9(b) shows that the curves and the bars (e.g., $k_{max}^{Li}$, denoting the coreness $k_{max}$ retrieved by *Li*) of *Lc* and *Li* are very close for various social networks with $c = 16$. This experiment again demonstrates the effectiveness of *Lc*/*Li* for CGNN queries.

*Exp-3*: We evaluated $k_{max}$ with respect to the degree distribution of $u_q$, representing the variation of $c$, in the five social networks, as shown in Fig. 10(a). $k_{max}$ increases when the degree distribution of $u_q$ spans a larger interval, which corresponds to CE/*Lc*/*Li* finding the maximum coreness in $C_{max}(u_q, c)$. For FB, OR and TW, $k_{max}$ is always close to the upper bound of interval, followed by GO. From the average degrees in Table 2, we conclude that FB, OR and TW are of relatively high density, affecting the performance of SD.

*Exp-4*: We explored the CGNN query performance $c = 64$ of CGNN and CGNN−opt, on real-world scale-commensurate social-road networks: FB+CA, FB+SF, BR+CA, BR+SF, GO+SF, GO+FL, OR+FL, OR+WU, TW+FL and TW+WU. Fig. 10(b) shows that the algorithms can satisfy user requirements for great numbers of attendees and that the query performance is rather stable as the scale of the road-social network varies.

*Exp-5*: We evaluated the efficiency and efficacy of CGNN and CGNN−opt for varying $\epsilon$. In Fig. 11(a), all curves (running times) are rising slowly with increasing $\epsilon$. Since the variation of $\epsilon$ does not affect the acquisition of candidate set $S$ and the visited objects expanded during subsequent verification, the difference in running times is mainly determined by the efficiency of object expansion from query points. Thus, in CGNN−opt, $\epsilon$NN significantly accelerates the acquisition of objects within $\epsilon$, due to $I_{RN}$.

*Exp-6*: We examined the effect of $j$ on CGNN and CGNN−opt, as shown in Fig. 11(b). Here, the query performance is dominated by the road network techniques because finding the top-$j$ objects requires only a few seconds due to the relatively small values of $c$ and $\epsilon$; in particular, CGNN−opt is twice as fast as CGNN. In addition, the candidate size $|S|$ and the total number of visited objects (#POIs) increase with increasing $j$, where the latter is slightly larger than the former because our verification technique discards most candidate objects.

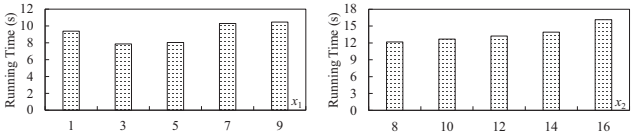
*Exp-7*: We evaluated the effect of the ratio of the number of objects to the number of edges (i.e., $E_r$) on CGNN and CGNN−opt, as shown in Fig. 11(c). This ratio varies from 0.0001 to 1 in FL. We then selected the objects with the keyword "restaurant" within default $\epsilon$. In the case where the top-$j$ objects are ensured to be found, both the running times of CGNN and CGNN−opt are rising. The smaller the ratio, the more sparsely the objects are uniformly distributed. Hence, the efficiency of range filter in Section 3.3 is almost unchanged when $\epsilon$ is fixed to the default, but the average cost of verification tends to be lower due to fewer POIs, as observed from $|S|$ and total number of expanded objects.

*Exp-8*: We examined the query processing time with respect to the number of attendees. Fig. 11(d) shows the results; all curves increase as $c$ increases. As long as *Li* is fast enough, optimizations of the road network dominate the CGNN query efficiency. The more invitees there are, the more objects need to be validated. In particular, CGNN−opt is twice as fast as CGNN because $\epsilon$NN uses the distance matrix of $I_{RN}$, eliminating the need for computing network distances incrementally and accumulatively.

### 6.3 Performance Evaluation for MCGNN Queries

We investigate the efficiency of the five algorithms, namely, TRS, TE, HE, TE−opt and HE−opt, for MCGNN queries listed in Table 3, then compare each under different settings.

*Exp-9*: We evaluated the impact of two parameters, i.e., $x_1$ and $x_2$, denoting the number of sorted accesses performed on each sorted list at one time, in the threshold ripple selection (*TRS*) described in Section 4.1. As shown in Fig. 12, we set $x_2 = 3$ (resp. $x_1 = 10$) and vary $x_1$ (resp. $x_2$) in range $[1, 9]$ (resp. $[8, 16]$). Larger $x_1/x_2$ may take longer to find objects seen in every list $L_i$, especially with more lists, i.e., $m$. Thus, we set $x_1 = x_2 = 3$ for TRS in the subsequent experiments. It is noting that we use *TRS* on behalf of the entire threshold selection strategies since its performance is superior to that of threshold selection (*TS*) in Section 4.1 according to our experimental observation.




(a) Varying $x_1$ ($x_2 = 3$) (b) Varying $x_2$ ($x_1 = 10$)

Fig. 12. Evaluation of threshold ripple selection: $x_1$ and $x_2$.

*Exp-10*: We explored the MCGNN query performance for $c = 64$ of TRS, TE, HE, TE−opt and HE−opt, on real-world scale-commensurate social-road networks: FB+CA, FB+SF, BR+CA, BR+SF, GO+SF, GO+FL, OR+FL, OR+WU, TW+FL and TW+WU. Fig. 13 shows that the algorithms can satisfy user requirements for great numbers of attendees and that the query performance is rather stable as the scale of the road-social network varies.

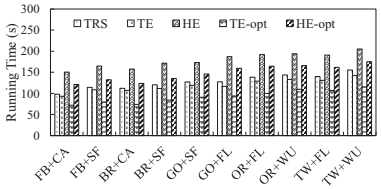
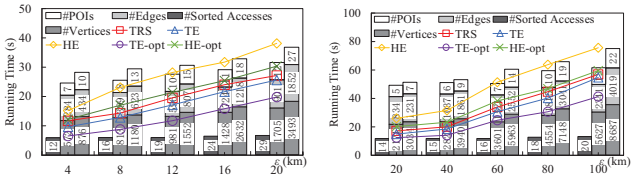



Fig. 13. Scalability for MCGNN queries ($c = 64$).

*Exp-11*: We evaluated the efficiency and efficacy of TRS, TE, HE, TE−opt and HE−opt for varying $\epsilon$. In Fig. 14, all curves (running times) and the bars denoting the numbers of accessed vertices/edges during expansion in TE and HE are rising with increasing $\epsilon$; but neither the number of sorted accesses in TRS nor the numbers of common visited objects in TE and HE are not sensitive to $\epsilon$ since greater $dist_{sum}$ dominates the overall score of POI. In TE−opt and HE−opt, $I_{RN}$ significantly accelerates the computation of network distances from any $q_i$ to vertices/objects within $\epsilon$ and the computation of lower bound against $dist_{sum}(o, Q)$, respectively. Moreover, HE is relatively slower than the other algorithms, mainly due to the large part of the road network it has to explore from all query points, although it avoids shortest path computations that could incur a large overhead. For TE and HE, the same vertices and edges can be visited by different query points, except that TE does not need to recompute $dist_{sum}(o, Q)$ but HE may check that for same object $o$ at most $c$ times.

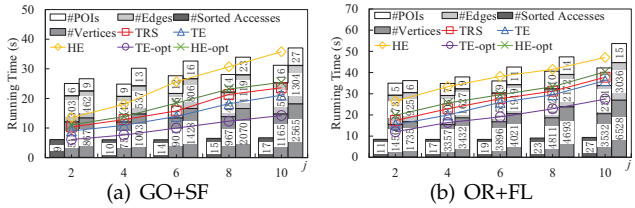



(a) GO+SF (b) OR+FL

Fig. 14. MCGNN query performance with respect to $\epsilon$.

*Exp-12*: We examined the effect of $j$ on the five algorithms for MCGNN queries, as shown in Fig. 15. Since the dominating cost is the road network expansion, the performance scales well with $j$ and the running time increases slowly as $j$ increases; in particular, the number of vertices/edges accessed during the network expansion in TE and HE increases linearly with $j$, but the number of corresponding common visited objects is not particularly sensitive to this parameter, where its change is very small or even constant compared to the change in the number of vertices/edges; so is the number of sorted accesses in TRS.

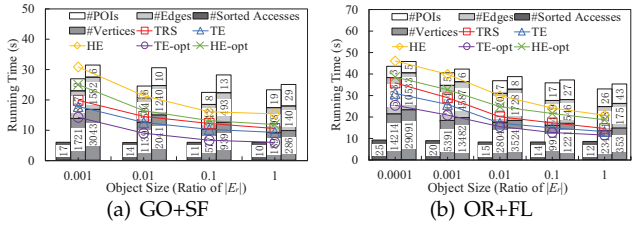
*Exp-13*: We evaluated the effect of the ratio of the number of objects to the number of edges (i.e., $E_r$) on the five




(a) GO+SF (b) OR+FL

Fig. 15. MCGNN query performance with respect to $j$.

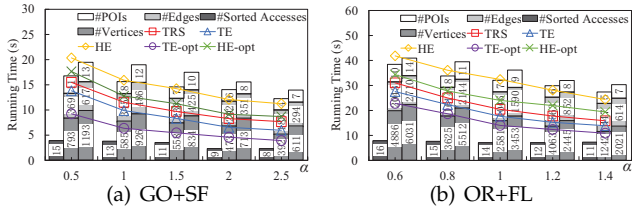
algorithms for MCGNN queries, as shown in Fig. 16. The ratio varies from 0.001 to 1 in SF and from 0.0001 to 1 in FL. We then selected the objects with the keyword "restaurant" within each default $\epsilon$. The running times of all algorithms are decreasing because objects are more densely and uniformly distributed. Hence, the average cost of network expansion tends to be lower because more POIs derive the top-$j$ results faster, i.e., earlier termination, as observed from the number of vertices/edges accessed during the network expansion in TE and HE, and the number of sorted accesses in TRS. However, the number of common visited objects is increased due to the increasing relative density of POIs scale against $|E_r|$, i.e., the ratio.




(a) GO+SF (b) OR+FL

Fig. 16. MCGNN query performance with respect to the object ratio.

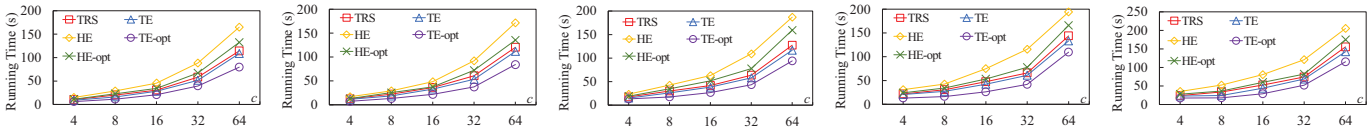
*Exp-14*: We examined the effect of $\alpha$ on the five algorithms for MCGNN queries, as shown in Fig.17. As $\alpha$ is in the denominator of Equation 3 and $\alpha > 1$ increases the importance of $dist_{sum}$ over $\overline{R_s}$, which means that POI with lower total travel cost (i.e., closer to $Q$) can obtain higher overall score with increasing $\alpha$. Thus, not only does the number of sorted accesses in TRS decrease, so does the number of POIs and vertices/edges accessed during network expansion in TE and HE, resulting in reductions in the running times.




(a) GO+SF (b) OR+FL

Fig. 17. MCGNN query performance with respect to $\alpha$.

*Exp-15*: We evaluated the query processing time with respect to the number of attendees (i.e., $c$) on five real-world scale-commensurate social-road networks: FB+SF, BR+SF, GO+FL, OR+WU and TW+WU. Fig. 18 shows the results; all curves increase as $c$ increases. As long as *Li* is fast enough, the dominating cost for MCGNN query efficiency is the road network expansion. The more invitees there are, the more objects need to be validated. TE's execution cost becomes higher due to the larger number of shortest path queries it has to perform. The cost of HE increases fast with $c$, because the same vertices/edges and their corresponding objects are checked multiple times (at most $c$) and the cost of each check is directly proportional to $c$. Among the algorithms, TE and TE−opt are superior to the other three; in particular,




(a) FB+SF (b) BR+SF (c) GO+FL (d) OR+WU (e) TW+WU

Fig. 18. MCGNN query processing time with respect to $c$.

`TE-opt` is twice as fast as `HE` due to the distance matrix in $I_{RN}$ and the earlier termination condition.

## 7 Related Work

*Geo-social query processing.* [16] formulates a framework for geo-social query processing that builds queries based on atomic operations, by which some complex queries (e.g., nearest friends and range friends) can be answered. Recently, Li *et al.* [17] have studied spatial-aware interest group queries in LBSN and presented efficient processing algorithms. A geo-social $k$-cover group query [18] retrieves a minimum user group in which each user is socially related to at least $k$ others and their associated regions (e.g., familiar/service regions) jointly cover all the given query points. [19], [20] find the largest community, which has minimum radius in Euclidean space, containing query user. [21] studies the $k$NN search on road networks incorporating social influence. Given a location $q$ and a constant $k$, a socio-spatial group query [22] returns an approximate group of users such that each has no social relationship with at most $k$ others in the group on average and their total distance to $q$ is minimized. This approach focuses only on Euclidean spatial distances, and the assembly point is designated in advance; if queries with the same parameters are sent by diverse users, the results are identical.

*$k$-core.* $k$-core computation, first introduced by Seidman [4], is a fundamental graph problem with a wide spectrum of applications, such as network analysis [23], graph clustering [24], and network visualization [25]. A linear-time in-memory algorithm for computing core numbers of all vertices in a graph is presented in [9]. I/O-efficient algorithms for core number computation on graphs that cannot fit in the main memory of a machine are proposed in [26], [27]. Whether $k$-core decomposition of large networks can be computed on a consumer-grade PC is explored in [10]. Local computation and estimation of core numbers are studied in [28], [29]. Algorithms for core number maintenance on dynamic graphs are proposed in [30]. However, the most cohesive $k$-core model is better suited to personalized user group queries (i.e., attendees invited by different query users are varied), and concurrently has more desirable properties, i.e., society, cohesiveness, connectivity and maximization.

*Nearest neighbor search.* As one of the most important queries among NN search variants, a group nearest neighbor (GNN) query [1] retrieves the point(s) in a given set of points $P$ with the smallest sum of distances to all points in another given set of points $Q$. An aggregate nearest neighbor (ANN) query [2], which returns the point(s) in $P$ that minimizes an aggregate function with respect to $Q$, was subsequently proposed as an extension of GNN query, which is equivalent to ANN query with an aggregate function of *sum* exclusively. As the methods of [1], [2] are applicable only in Euclidean space, [31] investigated the solution of ANN queries in road networks. However, this approach is not applicable in the case of edge weights that are disproportionate to the corresponding physical lengths, and the query performance is inefficient when the scale of $Q$ is large. Moreover, [32] addresses the problem of ANN query monitoring for moving objects in Euclidean space. [33] discusses ANN queries with moving query points. [34] explores ANN queries for query points with location privacy concerns. [35], [36] addresses ANN queries on uncertain databases and graphs. In these works, neither the social connectivity among the spatial query points in $Q$ nor textual descriptions of the spatial objects in $P$ are considered.

Our work, being more flexible and scalable, is totally different from that described above: (1) the assembly points are analyzed dynamically with regard to the optimal attendees, (2) the unique social topology can be adequately considered due to the limited number of attendees, (3) the core number $k$ is self-optimized to obtain the highest familiarity, and (4) distance restrictions based on a combination of *sum* and *max* functions are simultaneously applied to road networks. In real life, CGNN queries are often the most natural way to express the requests of an activity initiator or mobile user who wishes to organize an offline activity. To our knowledge, this paper proposes the first practical algorithm for solving the CGNN problem on general road-social networks.

## 8 Conclusions

In this paper, we define two pragmatic query types, namely, CGNN and MCGNN queries, to identify suitable spatial-textual objects as assembly points for a group of optimal attendees over road-social networks. To our knowledge, no previous solution has addressed such types of scenarios in automated offline activity planning services based on the social and geospatial relationship of activity attendees. We show that both problems are nontrivial, and (1) for CGNN queries, an efficient filtering-and-verification framework is devised; (2) for MCGNN queries, two selection-based and two expansion-based threshold algorithms are proposed. Moreover, it is shown that with optimizations, the performance of both queries is improved and less time is required to find optimal solution in both social and spatial domains. Experimental results for real-world road-social networks significantly demonstrate that our approaches are highly scalable and robust in both efficiency and efficacy. A possible direction for future work is to integrate other user attributes, e.g., user preferences, to filter activity attendees. Potential future research also includes joint social and road processing on networks stored in a distributed manner.

## References

[1] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis, "Group nearest neighbor queries," in *ICDE*, 2004, pp. 301–312.

[2] D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui, "Aggregate nearest neighbor queries in spatial databases," *TODS*, vol. 30, no. 2, pp. 529–576, 2005.

[3] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query processing in spatial network databases," in *VLDB*, 2003, pp. 802–813.

[4] S. B. Seidman, "Network structure and minimum degree," *Social networks*, vol. 5, no. 3, pp. 269–287, 1983.

[5] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," in *PODS*, 2001, pp. 102–113.

[6] B. Tang, S. Han, M. L. Yiu, R. Ding, and D. Zhang, "Extracting top-k insights from multi-dimensional data," in *SIGMOD*, 2017, pp. 1509–1524.

[7] K. Mouratidis and B. Tang, "Exact processing of uncertain top-k queries in multi-criteria settings," in *VLDB*, 2018, pp. 866–879.

[8] R. Zhong, G. Li, K.-L. Tan, L. Zhou, and Z. Gong, "G-tree: An efficient and scalable index for spatial search on road networks," *TKDE*, vol. 27, no. 8, pp. 2175–2189, 2015.

[9] V. Batagelj and M. Zaversnik, "An o(m) algorithm for cores decomposition of networks," *CoRR, cs.DS/0310049*, 2003.

[10] W. Khaouid, M. Barsky, V. Srinivasan, and A. Thomo, "K-core decomposition of large networks on a single pc," *PVLDB*, vol. 9, no. 1, pp. 13–23, 2015.

[11] S. Luo, Y. Luo, S. Zhou, G. Cong, J. Guan, and Z. Yong, "Distributed spatial keyword querying on road networks." in *EDBT*, 2014, pp. 235–246.

[12] M. Kolahdouzan and C. Shahabi, "Voronoi-based k nearest neighbor search for spatial network databases," in *VLDB*, 2004, pp. 840–851.

[13] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[14] T. Abeywickrama, M. A. Cheema, and D. Taniar, "K-nearest neighbors on road networks: A journey in experimentation and in-memory implementation." in *PVLDB*, vol. 9, no. 6, 2016, pp. 492–503.

[15] F. Guo, Y. Yuan, G. Wang, L. Chen, X. Lian, and Z. Wang, "Cohesive group nearest neighbor queries over road-social networks," in *ICDE*, 2019, pp. 434–445.

[16] N. Armenatzoglou, S. Papadopoulos, and D. Papadias, "A general framework for geo-social query processing," *PVLDB*, vol. 6, no. 10, pp. 913–924, 2013.

[17] Y. Li, D. Wu, J. Xu, B. Choi, and W. Su, "Spatial-aware interest group queries in location-based social networks," *DKE*, vol. 92, pp. 20–38, 2014.

[18] Y. Li, R. Chen, J. Xu, Q. Huang, H. Hu, and B. Choi, "Geo-social k-cover group queries for collaborative spatial computing," *TKDE*, vol. 27, no. 10, pp. 2729–2742, 2015.

[19] Y. Fang, R. Cheng, X. Li, S. Luo, and J. Hu, "Effective community search over large spatial graphs." in *PVLDB*, vol. 10, no. 6, 2017, pp. 709–720.

[20] Y. Fang, Z. Wang, R. Cheng, X. Li, S. Luo, J. Hu, and X. Chen, "On spatial-aware community search," *TKDE*, vol. 31, no. 4, pp. 783–798, 2019.

[21] Y. Yuan, X. Lian, L. Chen, Y. Sun, and G. Wang, "Rsknn: knn search on road networks by incorporating social influence," *TKDE*, vol. 28, no. 6, pp. 1575–1588, 2016.

[22] D.-N. Yang, C.-Y. Shen, W.-C. Lee, and M.-S. Chen, "On socio-spatial group query for location-based social networks," in *SIGKDD*, 2012, pp. 949–957.

[23] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani, "Large scale networks fingerprinting and visualization using the k-core decomposition," in *NIPS*, 2006, pp. 41–50.

[24] C. Giatsidis, F. D. Malliaros, D. M. Thilikos, and M. Vazirgiannis, "Corecluster: A degeneracy based graph clustering framework." in *AAAI*, vol. 14, 2014, pp. 44–50.

[25] F. Zhao and A. K. Tung, "Large scale cohesive subgraphs discovery for social network visual analysis," *PVLDB*, vol. 6, no. 2, pp. 85–96, 2012.

[26] J. Cheng, Y. Ke, S. Chu, and M. T. Özsu, "Efficient core decomposition in massive networks," in *ICDE*, 2011, pp. 51–62.

[27] D. Wen, L. Qin, Y. Zhang, X. Lin, and J. X. Yu, "I/o efficient core graph decomposition at web scale," in *ICDE*, 2016, pp. 133–144.

[28] W. Cui, Y. Xiao, H. Wang, and W. Wang, "Local search of communities in large graphs," in *SIGMOD*, 2014, pp. 991–1002.

[29] M. P. OBrien and B. D. Sullivan, "Locally estimating core numbers," in *ICDM*, 2014, pp. 460–469.

[30] Y. Zhang, J. X. Yu, Y. Zhang, and L. Qin, "A fast order-based approach for core maintenance," in *ICDE*, 2017, pp. 337–348.

[31] M. L. Yiu, N. Mamoulis, and D. Papadias, "Aggregate nearest neighbor queries in road networks," *TKDE*, vol. 17, no. 6, pp. 820–833, 2005.

[32] H. G. Elmongui, M. F. Mokbel, and W. G. Aref, "Continuous aggregate nearest neighbor queries," *GeoInformatica*, vol. 17, no. 1, pp. 63–95, 2013.

[33] J. Li, J. R. Thomsen, M. L. Yiu, and N. Mamoulis, "Efficient notification of meeting points for moving groups via independent safe regions," *TKDE*, vol. 27, no. 7, pp. 1767–1781, 2015.

[34] T. Hashem, L. Kulik, and R. Zhang, "Privacy preserving group nearest neighbor queries," in *EDBT*, 2010, pp. 489–500.

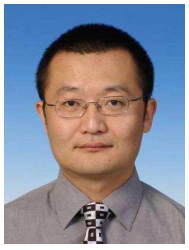
[35] X. Lian and L. Chen, "Probabilistic group nearest neighbor queries in uncertain databases," *TKDE*, vol. 20, no. 6, pp. 809–824, 2008.

[36] Z. Liu, C. Wang, and J. Wang, "Aggregate nearest neighbor queries in uncertain graphs," *WWW*, vol. 17, no. 1, pp. 161–188, 2014.


**Fangda Guo** received the BE degree in software engineering from Northeastern University, China, in 2013, and double MS degrees both in software engineering and computer engineering from Northeastern University, China, and University of Pavia, Italy, respectively. He is currently working toward the PhD degree in the School of Computer Science and Engineering, Northeastern University, China. His research interests include graph databases, LBSN, social network analysis and computer vision.

**Ye Yuan** received the BS, MS, and PhD degrees in computer science from Northeastern University, China, in 2004, 2007, and 2011, respectively. He is currently a professor in the School of Computer Science and Engineering, Northeastern University, Shenyang, China. His research interests include graph databases, probabilistic databases, and social network analysis.

**Guoren Wang** received the BSc, MSc, and PhD degrees in computer science from Northeastern University, China, in 1988, 1991, and 1996, respectively. He is currently a professor in the School of Computer Science and Technology, Beijing Institute of Technology, China. His research interests include XML data management, query processing and optimization, bioinformatics, high-dimensional indexing, parallel database systems, and cloud data management. He has published more than 100 research papers.



**Lei Chen** received the bachelor's degree in computer science from Tianjin University, China, in 1994, the master's degree in computer science from the Asian Institute of Technology, in 1997, and the PhD degree in computer science from the University of Waterloo, Canada. He is currently a professor of computing science with the Hong Kong University of Science and Technology, China. His research interests include multimedia databases, graph databases, uncertain, and probabilistic databases.

**Xiang Lian** received the BS degree from the Department of Computer Science and Technology, Nanjing University, in 2003, and the PhD degree in computer science from the Hong Kong University of Science and Technology, Hong Kong. He is currently an assistant professor in the Department of Computer Science, Kent University. His research interests include probabilistic/uncertain data management, probabilistic RDF graphs, inconsistent probabilistic databases, and streaming time series.

**Zimeng Wang** received the BE degree in computer science and technology from Northeastern University, China, in 2017. He is currently working toward the master degree in the School of Computer Science and Engineering, Northeastern University, Shenyang, China. His current research interests include big data processing framework and parallel computing.