Chen J, Zhang KQ, Ren T *et al.* GAM: A GPU-accelerated algorithm for MaxRS queries in road networks. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 37(5): 1005–1025 Sept. 2022. DOI 10.1007/s11390-022-2330-3

GAM: A GPU-Accelerated Algorithm for MaxRS Queries in Road Networks

Jian Chen (陈 剑), *Student Member*, *CCF*, Kai-Qi Zhang* (张开旗), Tian Ren (任 甜)
Zhen-Qing Wu (武震卿), and Hong Gao* (高 宏), *Distinguished Member*, *CCF*, *Member*, *ACM*

School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China

E-mail: chenjian@stu.hit.edu.cn; zhangkaiqi@hit.edu.cn; {rentian, wuzhenqing}@stu.hit.edu.cn
honggao@hit.edu.cn

Received March 22, 2022; accepted September 21, 2022.

Abstract In smart phones, vehicles and wearable devices, GPS sensors are ubiquitous and collect a lot of valuable spatial data from the real world. Given a set of weighted points and a rectangle r in the space, a maximizing range sum (MaxRS) query is to find the position of r, so as to maximize the total weight of the points covered by r (i.e., the range sum). It has a wide spectrum of applications in spatial crowdsourcing, facility location and traffic monitoring. Most of the existing research focuses on the Euclidean space; however, in real life, the user's moving route is constrained by the road network, and the existing MaxRS query algorithms in the road network are inefficient. In this paper, we propose a novel GPU-accelerated algorithm, namely, GAM, to tackle MaxRS queries in road networks in two phases efficiently. In phase 1, we partition the entire road network into many small cells by a grid and theoretically prove the correctness of parallel query results by grid shifting, and then we propose an effective multi-grained pruning technique, by which the majority of cells can be pruned without further checking. In phase 2, we design a GPU-friendly storage structure, cell-based road network (CRN), and a two-level parallel framework to compute the final result in the remaining cells. Finally, we conduct extensive experiments on two real-world road networks, and the experimental results demonstrate that GAM is on average one order faster than state-of-the-art competitors, and the maximum speedup can achieve about 55 times.

Keywords road network, maximizing range sum, GPU acceleration, pruning strategy

1 Introduction

Driven by the advance in networking and communications, it has become more and more pervasive to obtain spatial location data through various sensors, which has led to a surge in spatial data generated from every corner around the world [1]. This in turn promotes a variety of location-based services (LBS), such as tourism planning, urban mobility services [2] and navigation services [3]. The core of these applications is the efficient management and processing of massive spatial data. Therefore, a great deal of attention has been drawn into processing massive spatial data.

As one of the most fundamental operations of the spatial database, the MaxRS query [4] has caused extensive research in recent years. In the 2-dimensional (2D) space, given a set O of weighted points and a rectangle r with a user-specified size (e.g., $a \times b$), the MaxRS query is to find the optimal position for the rectangle r, so that the total weight of all points covered by r is maximized. This problem is widely used in many fields (e.g., facility location problems [5], spatial data mining [6], and crowdsourcing [7–9]), and two classic real-life scenarios are as follows.

Regular Paper
Special Section on Scalable Data Science
This work was supported in part by the Key Research and Development Plan of National Ministry of Science and Technology under Grant No. 2019YFB2101902, the National Natural Science Foundation of China under Grant Nos. U19A2059 and 62102119, and the CCF-Baidu Open Fund CCF-BAIDU under Grant No. OF2021011.

*Corresponding Author (Kai-Qi Zhang perfected the idea and experimental design proposed by Jian Chen, and provided the experimental platform for this research. Hong Gao participated in the revision and polishing of the full paper, and provided funding and equipment support for this research.)

• *Scenario* 1. When a tourist travels to a city, due to the limited reachable range of his/her daily activities, he/she often hopes to book the most representative hotel in the city in order to maximize the number of tourist attractions near the hotel.

• *Scenario* 2. Nowadays, the food delivery industry is booming, but many shops have limited delivery areas. Pizza shop managers usually want to find the best store location to maximize the number of customers that can be delivered.

Due to the diverse real-life application scenarios, many variants of MaxRS have recently been proposed and solved in the spatial database community (e.g., scalability [3], approximate solutions [10], moving objects [11], probability [12] and class-based constraints [13]). Although the above methods can effectively deal with MaxRS queries, they strictly assume that all points are in the Euclidean space; in other words, the query result is determined based on the straight-line distance between points, which may be significantly different from the actual distance between points. MaxRS queries in the Euclidean space fail to retrieve all user-satisfied results accurately. However, in most real-life LBS, almost all facilities (e.g., hotels, restaurants, and scenic spots) are adjacent to the road network. No matter whether walking or driving, the user's moving route must be constrained by the road network. Therefore, MaxRS queries in road networks are required to return more accurate and practical results, based on the road network distance [14] (i.e., the length of the shortest path connecting them), instead of the location.

Fig.1 gives an example of scenario 1 with four scenic spots (f_1, f_2, f_3, f_4) of unit weight 1. Suppose that the daily activity radius r of tourists is 1.5. In Fig.1(a), according to the principle of rectangle intersection [15], the MaxRS query in the Euclidean space will return s_1 (i.e., MaxRS is 4), and hotels in the range s_1 will be recommended to tourists. However, in Fig.1(b), the MaxRS query in the road network will return s_2 (i.e., MaxRS is 3), which is more accurate than s_1 in real life. Thus, tourists prefer to book hotels in s_2.

Up to now, there has been very little research on MaxRS queries in road networks. Although the existing studies [16, 17] propose general algorithms for processing MaxRS queries in road networks, the performance of these algorithms is inefficient due to the lack of consideration of the unique characteristics of the road network. When the data scale or r is large, it takes hundreds or even thousands of seconds to return all query results, which is not efficient enough to meet the actual needs of users in daily life. In a nutshell, the existing algorithms cannot handle MaxRS queries in road networks well.

MaxRS queries are expensive to compute in road networks. Therefore, like several other database operators (e.g., [18–21]), a natural solution is to exploit GPU to accelerate it. In a city with a grid-shaped road network, for scenario 2, an important observation is that the delivery range of most pizza shops only covers a few blocks nearby. Inspired by this, we can partition the entire road network into many cells, and then the MaxRS query can be performed concurrently in many cells with GPU. However, it is non-trivial to propose an efficient GPU acceleration algorithm. The challenges are as follows: 1) how to partition the road network to ensure the correctness of query results; 2) in general, for high throughput inevitably faces a multiplied workload,

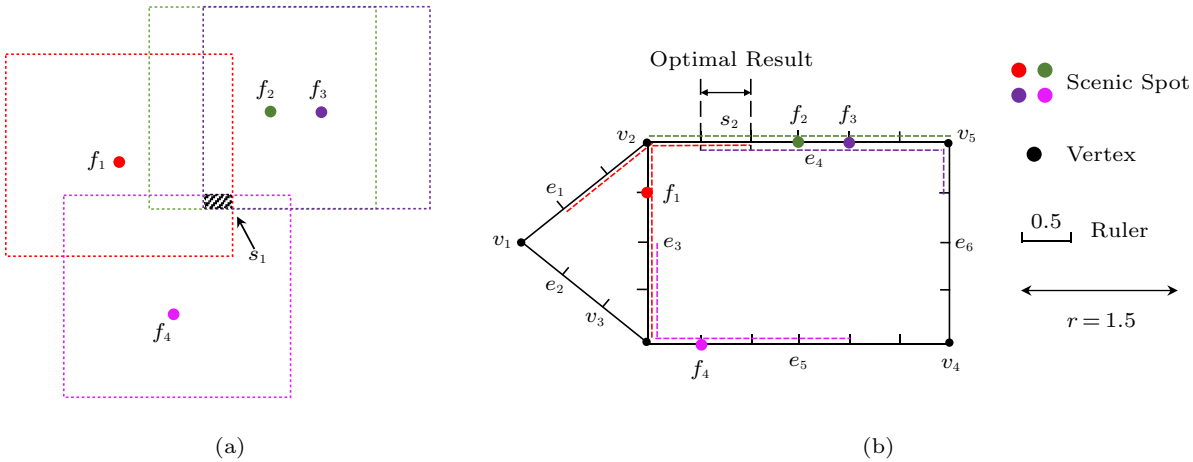


Fig.1. Examples of MaxRS queries. (a) MaxRS query in the Euclidean space. (b) MaxRS query in road networks.

how to safely prune unnecessary computations; 3) how to design an efficient storage structure and a computing framework to make full use of GPU resources.

In this paper, we first design a GPU-accelerated algorithm called GAM (GPU-accelerated MaxRS) to solve MaxRS queries in road networks efficiently. To handle above challenges, GAM partitions the entire road network into small cells by a grid adapted to r. Then we propose the multi-grained pruning technique to safely prune unpromising cells that contain no MaxRS result. For GPU acceleration, a two-level parallel framework is proposed to compute final results based on our GPU-friendly storage structure CRN. In extensive experiments, GAM is able to solve MaxRS queries in road networks efficiently.

Contributions. The specific contributions of this paper are summarized as follows.

• This paper proposes a novel GPU-accelerated algorithm, namely, GAM. To our knowledge, this is the first attempt to deal with MaxRS queries in road networks utilizing GPU.

• We adopt the grid partitioning technique to partition the entire road network into many small cells, and then theoretically prove the correctness of parallel query results by grid shifting.

• We propose an effective multi-grained pruning technique, by which the majority of cells can be pruned without further checking.

• We design a GPU-friendly storage structure CRN to facilitate quick access to the spatial data corresponding to each cell and store intermediate results, based on which a two-level parallel framework is proposed to compute final results in the remaining promising cells.

• We conduct extensive experiments on real-life road network datasets with different data scales, which shows that the GAM algorithm is significantly superior to the baseline algorithm.

2 Related Work

2.1 MaxRS Queries in the Euclidean Space

The predecessor of the MaxRS problem is the Max-enclosing rectangle problem, which is originally proposed in the field of computational geometry [15, 22]. This problem only focuses on maximizing the number of covered points without considering the weight of the points. Specifically, Imai and Asano [22] transformed this problem into finding connected components and a maximum clique of an intersection graph of rectangles in the plane, and devised an efficient algorithm with $O(n \log n)$ time complexity. Subsequently, Nandy and Bhattacharya presented an alternative algorithm with the same complexity in [15], which utilizes the interval tree data structure and the plane-sweep technique to locate the maximum enclosing rectangle over a set of points.

Although the above two algorithms provide the theoretical bound, they have a poor scalability and are not suitable for processing massive spatial data. Choi *et al.* [3] proposed the MaxRS query and provided a scalable external-memory algorithm, which is optimal in terms of the I/O complexity. From the perspective of probability sampling, Tao *et al.* [10] designed a $(1-\varepsilon)$-approximation algorithm with the time complexity of $O(n \log \frac{1}{\varepsilon} + n \log \log n)$.

Furthermore, there are many interesting variants of the MaxRS problem. Considering that objects will appear or disappear dynamically, Amagata and Hara [23] first solved the problem of monitoring MaxRS in spatial data streams, and designed an index-based algorithm to update the result in real time when the data changes. As a supplement to [23], Chen *et al.* [24] defined the rotating MaxRS problem, which allows non axis-parallel rectangles.

Driven by the widespread popularity of GPS-enabled mobile devices such as smart watches and vehicles, Hussain *et al.* [11] focused on the trajectory of the moving object, and investigated the Co-MaxRS (Continuous MaxRS) query problem. They utilized the kinetic data structures to solve this problem. Afterwards, Hussain *et al.* [13] defined a novel query, the C-MaxRS (Conditional MaxRS), for spatial data and devised efficient pruning strategies and updating strategies to quickly return results. Due to the inaccuracy of the location acquisition, Liu *et al.* [12] defined the probability MaxRS query for uncertain objects to find the rectangle solution with a probability higher than the user-specified threshold, which is proved to be #P-complete. And the PMaxRS framework based on pruning and refinement strategies was developed.

2.2 MaxRS Queries in Road Networks

In real life, users cannot move freely in a straight line in the space. Considering this point, Phan *et al.* [17] defined MaxRS queries in road networks for the first time and proposed the BTSF (B+-Tree Seg-File) algorithm. This algorithm consists of the following three main steps: 1) recursively generating segments for each facility according to the depth-first search strategy; 2)

inserting the generated segment into the Seg-File, and merging the segments of the same facility (three specific cases); 3) scanning the Seg-File data once, and using the line sweep method to calculate the optimal result on each edge.

In order to effectively support online search, Zhou and Wang [16] proposed an index (IND) algorithm based on a pre-computing index to efficiently calculate online MaxRS queries in road networks. The time complexity of this method is $O(\log |F|)$, and the index size is $O(|F|)$, linearly with the number of facilities $|F|$.

Feng *et al.* [25] studied the best region search (BRS) problem for the submodular monotonic aggregate scoring function, which is more flexible than the weight sum function. Considering the movement of objects, Chen *et al.* [26] concentrated on efficiently maintaining the BRS solutions in the road network space, and proposed a branch-and-bound algorithm based on preprocessing to dynamically monitor a fixed-size optimal region at different times.

2.3 GPU and CUDA Computing

Initially, GPU was designed for high-speed parallel graphics rendering on computers and mobile devices. Due to its huge parallelism, it is suitable for highly parallel computing tasks with simple logic control (e.g., reinforcement learning [27–29]). In order to simplify the development of GPU programs, NVIDIA has developed a parallel programming model CUDA [30].

From the hardware perspective, a GPU is usually composed of dozens of SMs (streaming multiprocessors), each of which contains multiple cores that run threads concurrently. All SMs share the global memory of the GPU, and each SM has its own shared memory. All threads execute the same instruction set for different data according to the SIMD (single instruction multiple data) model.

From the software perspective, the CUDA programming model is divided into two parts: host and device. The host program is executed on the CPU, and the device program (kernel function) is executed on the GPU. In order to facilitate GPU thread management, all threads are organized into a hierarchical structure, from top to bottom as grids, blocks, and threads respectively. A block is composed of a group of threads, and further, multiple blocks are combined to form a grid.

Combining software with hardware, SM is composed of multiple SPs [31]. When the program is executed on the GPU, CUDA blocks are distributed among GPU SMs, and CUDA threads in a block are run on SPs concurrently. The CUDA model puts 32 threads in a block into a warp to execute the same instruction synchronously, and an SM can only execute one wrap at a time.

2.4 Discussion

For massive spatial data, none of the existing algorithms can efficiently return MaxRS results in road networks. As the query radius r increases, the time cost of the segment generation based algorithm grows super-linearly. The index-based algorithm [16] requires expensive pre-computing cost, and no update operation is provided. Once the location of the facility is updated, the pre-computed index will become invalid. At this time, it is necessary to perform pre-computing from scratch again, and therefore it is not suitable for real-life interactive scenes where facilities are frequently updated.

For the segment generation based algorithm, an important observation is that both the segment generation step and the line sweep step involve executing the same procedure over different spatial data. For instance, we need to generate segments for each facility along the road network, and then sweep all edges to compute the final result. This possibility of data-level parallelism inspires us to leverage GPU techniques to speed up MaxRS queries in road networks. As such, this paper aims to devise a novel algorithm, which can utilize GPU to accelerate MaxRS queries in road networks efficiently.

3 Problem Formulation

In this section, we follow the definition in [17] to formally describe the problem. Table 1 summarizes the symbols used throughout this paper.

Definition 1 (Road Network). *A road network is represented as an undirected graph $G = (V, E)$, where V is a set of nodes (i.e., vertices), and E is a set of edges given as pairs of nodes. A node $v \in V$ has 2D coordinates: $loc(v) = (v.x, v.y)$. Nodes and edges are assigned unique identifiers. F is used to denote the set of facilities, of which each, denoted by f, is located on an edge and is associated with a positive weight $w(f)$.*

Definition 2 (Network Range and Query Radius). *Given a road network $G = (V, E)$ and a point p in G, the network range $p(r)$ of p contains any point whose network distance to p is no greater than r, where r is*

called the query radius. Formal representation of the network range is as follows.

$$p(r) = \{p' | dist(p, p') \leqslant r, p' \in P\},$$

where P is the set of all points on nodes or edges of the road network G, and $dist(p, p')$ is the shortest path length from p to p'.

Table 1. Summary of Symbols

Symbol	Meaning
$G = (V, E)$	Road network G with nodes V and edges E
$dist(p_1, p_2)$	Shortest network distance from location p_1 to location p_2
$dist_\epsilon(p_1, p_2)$	Euclidean distance between locations p_1 and p_2
P	Set of all points on nodes or edges of G
P^*	Set of optimal results
(α, β, r)-$Grid$	Grid partitioning with seed (α, β) and query radius r
G_0, G_1, G_2, G_3	Grid with seed $(0, 0), (2r, 0), (2r, 2r), (0, 2r)$
$C_{i,j}$	Cell in a grid partitioning
$p(r)$	$p(r) = \{p' \| dist(p, p') \leqslant r, p' \in P\}$
$r(p)$	$r(p) = \{p' \| dist_\epsilon(p, p') \leqslant r, p' \in P\}$
$R(p)$	$R(p) = \{p' \| \|p.x - p'.x\| \leqslant r$ and $\|p.y - p'.y\| \leqslant r, p' \in P\}$
$F_{p(r)}$	Set of facilities covered by $p(r)$ in the road network G
$F_{r(p)}$	Set of facilities covered by $r(p)$ in the road network G
$F_{R(p)}$	Set of facilities covered by $R(p)$ in the road network G
$\widehat{maxrs}$	Upper bound of the optimal result
$MaxRS_{\text{ct}}$	Current MaxRS result
$\mathcal{H}$	Cell-based header table in CRN
$\mathcal{L}$	Array-based adjacency list in CRN

Definition 3 (MaxRS Query in Road Network). *Given a road network $G = (V, E)$, a set of positive-weighted facilities F, and a user-specified query radius r, the MaxRS query in road networks is to find all points p^* in G such that the total weight of all facilities covered by $p^*(r)$ is maximized, namely,*

$$p^* = \arg\max_{p \in G} \sum_{f \in F_{p(r)}} w(f),$$

where $F_{p(r)}$ represents the set of facilities covered by $p(r)$ in the road network, which is described formally as follows.

$$F_{p(r)} = \{f | f \in p(r) \wedge f \in F\}.$$

An example is given to illustrate the MaxRS query in the road network. Specifically, Fig.1(b) shows a road network with five nodes, six edges, and four facilities. The edge and the facility information are shown in Table 2 and Table 3, respectively. Given a query radius $r = 1.5$, any position within $s_2 = f_1(r) \cap f_2(r) \cap f_3(r)$ can be the answer of this query, since the total weight of facilities that can reach these positions within the distance of 1.5 is the maximum (i.e., 3). Therefore, for scenario 1, the tourist can book any hotel within s to maximize the number of nearby attractions.

Table 2. Edge Information of Road Network in Fig.1(b)

Edge	Start Node	End Node	Edge Length
e_1	v_1	v_2	1.5
e_2	v_1	v_3	1.5
e_3	v_2	v_3	2.0
e_4	v_2	v_5	3.0
e_5	v_3	v_4	3.0
e_6	v_4	v_5	2.0

Table 3. Facility Information of Road Network in Fig.1(b)

Facility	Edge	Position in Edge	Facility Weight
f_1	e_3	0.5	1
f_2	e_4	1.5	1
f_3	e_4	2.0	1
f_4	e_5	0.5	1

4 GAM Algorithm

4.1 Overview of GAM

In this subsection, we provide an overview of our solution, as depicted in Fig. 2. Our algorithm, called GAM, solves the problem in two phases. The first phase partitions the road network into cells by a grid and selects the promising cells for further checking. The second phase computes optimal results in the promising cells based on a two-level parallel query framework. We highlight data transfer between the two phases with the red solid lines in Fig.2.

4.1.1 Partitioning and Pruning

There are two main steps in this phase.

• *Grid Partitioning.* To achieve parallel queries, an intuitive way is to partition the road network into many sub-regions and independently conduct the MaxRS query within each sub-region. Then, we integrate all the local results in each sub-region to get the global optimal result. The key challenge lies in how to partition the road network to ensure the correctness and completeness of query results. According to the query radius r, we use grids to partition the road network.

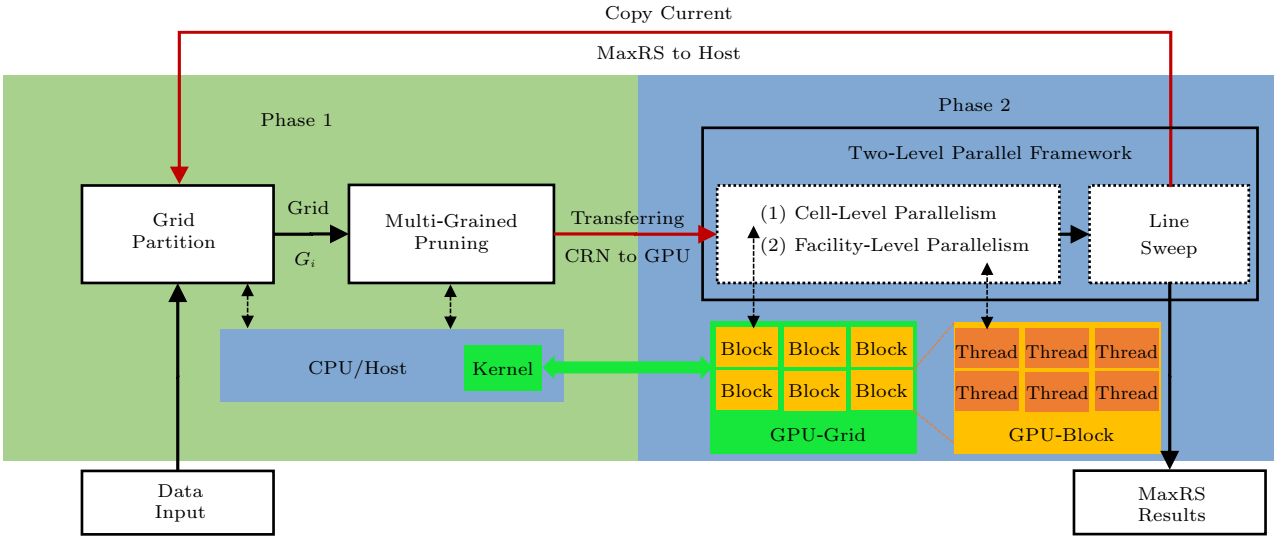


Fig.2. Overview of GAM.

Specifically, we select one seedpoint in the road network G, and then divide G into $\lceil l/4r \rceil \times \lceil w/4r \rceil$ disjoint cells of the same size along the two geospatial coordinates, where l and w are the latitude and the longitude range of the road network respectively. We map each facility, node and edge to the corresponding cell. After grid partitioning, we independently conduct the MaxRS query within cells in parallel, which ensures that parallel results are consistent with original results, as to be demonstrated in Subsection 4.2.1.

• *Multi-Grained Pruning.* There are a lot of cells that do not contain the optimal result in the grid, and therefore we do not need to further check them. Thus, pruning strategies are necessary to significantly reduce the computation of unpromising cells. For this purpose, we first compute the upper bound of the optimal result within each cell C, denoted as $ub(C)$. Supposing $MaxRS_{\text{ct}}$ is the optimal result found so far, we can compare $ub(C)$ with $MaxRS_{\text{ct}}$. As to be proved in Subsection 4.2.2, any unpromising cell with $ub(C)$ less than $MaxRS_{\text{ct}}$ can be safely pruned without further consideration, and the remaining promising cells are utilized for subsequent MaxRS queries in phase 2. Intuitively, we use the total weight of the facilities in C as the upper bound of C's optimal result. Furthermore, we divide each cell into fine-grained squares evenly to get a more accurate upper bound, as to be specifically introduced in Subsection 4.2.2.

4.1.2 Storage and Parallel Computing

Based on the promising cells returned by phase 1, the second phase of GAM constructs optimal results in a two-level parallel query framework using GPU. Considering the GPU hardware architecture, the second phase consists of a GPU-friendly storage structure and a two-level parallel framework.

• *GPU-Friendly Storage Structure.* In order to quickly access the road network data in each cell, we design the GPU-friendly storage structure CRN (Cell-based Road Network). It consists of three major components: the cell-based header table, the array-based adjacency list and the facility list. The cell-based header table stores the concise information of the road network data in each cell, and the array-based adjacency list is used to reconstruct the road network in a cell from the cell-based header table. Thus, we can associate the header table with the adjacency list to access any edge or node in each cell efficiently. The facility list stores the position and weight of facilities in each cell. The GPU-friendly CRN facilitates efficient processing of MaxRS queries in cells, as to be described in detail in Subsection 4.3.1.

• *Two-Level Parallel Framework.* Due to the special hardware architecture of GPU, we design a two-level parallel framework to compute optimal results based on CRN. Specifically, our two-level parallel framework consists of cell-level parallelism and facility-level parallelism. 1) *Cell-Level Parallelism.* After the road network is partitioned into cells, we distribute the computation of cells among CUDA blocks, and then the processing between cells is in parallel. 2) *Facility-Level Parallelism.* There are many facilities within each cell. The MaxRS query in the road network is further decomposed into two independent components, namely,

segment generation [17] and line sweep [32]. Thus, the segment generation for facilities can be performed by CUDA threads concurrently. Then, we find the local results in each cell by line sweep and put the result with the maximal range sum computed so far into set P^*. Finally, we return P^* as final results.

4.2 Phase 1: Finding the Promising Cells

4.2.1 Grid Partitioning

In this subsection, we first introduce the grid partitioning technique adapted to r, and then theoretically prove that the parallel results based on this partition are exactly the same as original results. The grid partitioning adapted to r is critical to solving the GPU-accelerated MaxRS query and defined as follows.

Definition 4 ((α, β, r)-Grid). *Given a point (α, β) in $\mathbb{R}^2$ and a user-specified query radius r, (α, β, r)-grid is the set of vertical and horizontal lines defined by:*

$$x = \alpha + k \times 4r,$$
$$y = \beta + k \times 4r,$$

respectively, where $k = -\infty, ..., -1, 0, 1, ..., +\infty$ and $4r$ is the distance between two adjacent horizontal or vertical lines. Point (α, β) is the seed of the grid.

From the above definition, (α, β, r)-grid partitions the road network into $\lceil \frac{l}{4r} \rceil \times \lceil \frac{w}{4r} \rceil$ disjoint cells, where l and w are the latitude and the longitude range of the road network respectively. Each cell $C_{i,j}$ is a $4r \times 4r$ square surrounded by two consecutive vertical lines and two successive horizontal lines, where i and j are the column number and the row number of the cell in the grid, respectively. For example, the shadow area is the cell $C_{0,1}$ of $(0, 0, r)$-grid in Fig.3. We denote $(0, 0, r)$-grid as G_0. Shifting the seed point $(0, 0)$ to the right by $2r$, and then moving vertically upward by $2r$, we get the seed point $(2r, 2r)$, and denote $(2r, 2r, r)$-grid as G_2. In the same way, we use G_1 and G_3 to represent $(2r, 0, r)$-grid and $(0, 2r, r)$-grid, respectively. Considering the example in Fig.3 again, the solid lines form G_0 and the dotted lines constitute G_2.

For the sake of illustration, the maximum reachable range of a point p is denoted as $r(p) = \{p' | \sqrt{(p.x - p'.x)^2 + (p.y - p'.y)^2} \leqslant r, p' \text{ in } G\}$, whose bounding rectangle is denoted as $R(p) = \{p' || p.x - p'.x| \leqslant r \text{ and } |p.y - p'.y| \leqslant r, p' \text{ in } G\}$. The positional relationship between $r(p)$ and $R(p)$ is shown in Fig.4. Obviously, for $\forall p' \in r(p)$, we can deduce that p' must be in the set $R(p)$. Besides, let $dist_\epsilon(p, p')$ denote the Euclidean distance between locations p and p' and $dist(p, p')$ denote the shortest network distance from location p to location p', and then we can derive Lemma 1.

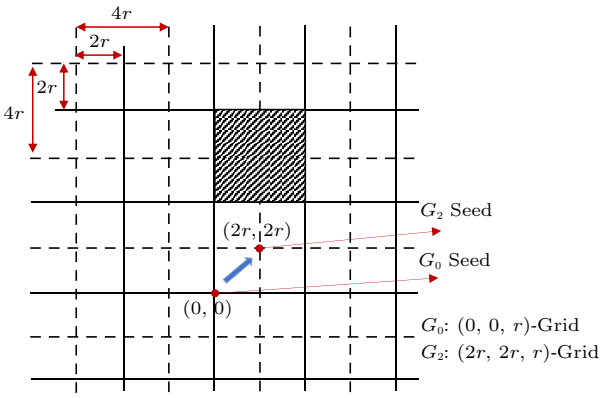


Fig.3. Grid partitioning example.

Lemma 1. *Let $F_{p(r)}, F_{r(p)}$, and $F_{R(p)}$ denote the set of facilities covered by ranges $p(r)$, $r(p)$, and $R(p)$ respectively. For any point p in G, they must satisfy: $F_{p(r)} \subseteq F_{r(p)} \subseteq F_{R(p)}$.*

Proof. According to the triangle inequality, for any point p, p' in G, the following equation always holds:

$$\begin{aligned} dist_\epsilon(p, p') &= \sqrt{(p.x - p'.x)^2 + (p.y - p'.y)^2} \\ &\leqslant dist(p, p'). \end{aligned}$$

Therefore, we have $p(r) \subseteq r(p)$. Furthermore, $F_{p(r)} \subseteq F_{r(p)}$. Since $r(p) \subseteq R(p)$ for any point p in G, obviously, $p(r) \subseteq r(p) \subseteq R(p)$, which leads to:

$$F_{p(r)} \subseteq F_{r(p)} \subseteq F_{R(p)}. \qquad \square$$

In a similar spirit to [10], an interesting insight is that the optimal result on any edge e always exists at a cell C in partition $G_i \in \{G_0, G_1, G_2, G_3\}$, as shown in the following Lemma 2.

Lemma 2. *Let P^* be the set of all optimal results. For any point $p^* \in P^*$ on an edge $e \in E$, there must be a road network partition $G_i \in \{G_0, G_1, G_2, G_3\}$, so that a certain cell C in G_i returns the optimal result p^*.*

Proof. For any optimal result $p^* \in P^*$, the maximum reachable range of p^* is denoted as $r(p^*)$, whose bounding rectangle is denoted as $R(p^*)$. According to Lemma 1: $F_{p^*(r)} \subseteq F_{r(p^*)} \subseteq F_{R(p^*)}$.

First, the top-left corner q of $R(p^*)$ must fall into a cell of the partition G_0. This cell is marked as a square $ABCD$ and further divided into four $2r \times 2r$ sub-squares, as shown in Fig.4(c), where they are sequentially numbered as 0, 1, 2, 3.

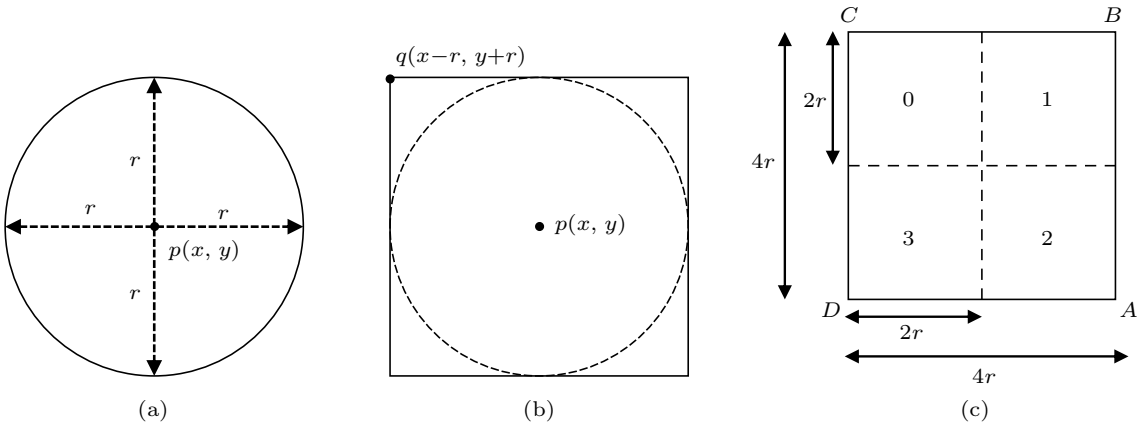


Fig.4. Demonstration of Lemma 1. (a) $r(p)$. (b) $R(p)$. (c) Cell-$ABCD$.

Case 1. If the point q falls into the sub-square 0, apparently, there is a cell (i.e., $ABCD$) of G_0 that completely contains $R(p^*)$.

Case 2. If the point q falls into sub-square 1, then we shift the seed point to the right by $2r$, and there is a cell in the grid G_1 that completely contains $R(p^*)$.

Case 3. If the point q falls into sub-square 2, similar to case 2, there is a cell in the grid G_2 that completely contains $R(p^*)$.

Case 4. If the point q falls into sub-square 3, similarly, there is a cell in the grid G_3 that completely contains $R(p^*)$.

In summary, if the point q is within the sub-square $i \in [0, 3]$, then the grid G_i has a cell that completely covers the facility set $F_{p^*(r)}$, that is, $\forall p^* \in P^*$, then, $\exists C$ in $G_i \in \{G_0, G_1, G_2, G_3\}$ covers facility set $F_{p^*(r)}$, and thus C can return the optimal result p^*. □

Based on Lemma 2, we can perform parallel MaxRS queries within each cell in four grids (G_0, G_1, G_2, G_3), and then integrate local results from cells to find all MaxRS results, which are consistent with original results. Namely, the parallel query does not miss any optimal result in the road network G. Next, we illustrate briefly the main steps to partition the road network by a grid.

Step 1: *Traverse the Facility Set F, for $\forall f \in F$.* First, the current f is mapped to the covering cell $C_{i,j}$ according to the following equations: $i = \lfloor \frac{f.x-\alpha}{4r} \rfloor$, $j = \lfloor \frac{f.y-\beta}{4r} \rfloor$, where $(f.x, f.y)$ is the coordinate of f. Then, we add f to the facility list of $C_{i,j}$, and at the same time, add the edge of f to the adjacency list of $C_{i,j}$.

Step 2: *Traverse the Node Set V, for $\forall v \in V$.* Similar to step 1, we first get the cell $C_{i,j}$ where v is mapped to according to the coordinates $loc(v) = (v.x, v.y)$, and then v and other nodes adjacent to v are added to $C_{i,j}$.

After finishing the grid partitioning of the road network, we can obtain the road network subgraph $G(C_{i,j})$ in each cell $C_{i,j}$. In addition, the complexity of the road network grid partitioning is: $O(|F| + |V|)$. There are three main benefits of grid partitioning to facilitate MaxRS queries in road networks, which are listed as follows.

1) *Conducive to Parallel Query.* Based on Lemma 2, after the road network is partitioned into grids, MaxRS queries in road networks can be conducted in parallel among cells, which is conducive to use GPU for parallel acceleration.

2) *Saving Computation.* By estimating the upper bound of the optimal result in each cell, unpromising cells can be safely pruned to save the computation overhead and improve the algorithm efficiency. Details of pruning strategy are introduced in Subsection 4.2.2.

3) *Reducing Memory Access Overhead.* The raw road network is relatively large, and even unable to be loaded into memory. In contrast, the road network subgraph $G(C_{i,j})$ after partitioning is usually very small. When the query radius r is small, $G(C_{i,j})$ can be directly loaded into the shared memory of the SM, so as to reduce the memory access overhead.

4.2.2 Multi-Grained Pruning

There are a large number of cells in the four partitions (G_0, G_1, G_2, G_3) where MaxRS queries need to be conducted independently, which is very time-consuming. However, due to many overlaps between the four partitions, many unpromising cells that contain no MaxRS results can be pruned without further checking. They not only lead to redundant computation, but also generate unnecessary data transmission between CPU and GPU. Therefore, it is necessary to

propose pruning strategies to reduce the unnecessary computation of unpromising cells.

Basic Pruning Idea. To avoid the computation of unpromising cells, for any cell C, we first estimate an upper bound $ub(C)$ of the MaxRS result in C, and then compare $ub(C)$ with the current MaxRS result $MaxRS_{\text{ct}}$. If $ub(C)<MaxRS_{\text{ct}}$, there is no better solution in C, and thus we prune this unpromising cell. A naive way to compute $ub(C)$ for C is to sum the weights of all facilities in C. Generally, let F_C denote the facility list covered by cell C.

Lemma 3 (Naive Pruning Strategy). *Given the current MaxRS value $MaxRS_{\text{ct}}$ and a cell C, if $\sum_{f\in F_C} w(f)<MaxRS_{\text{ct}}$ holds, then C can be safely pruned.*

Proof. For any cell C in grid G_i, the local optimal result in the cell is denoted as p^*_C. Apparently, there must be $\sum_{f\in F_{p^*_C(r)}} w(f) \leqslant \sum_{f\in F_C} w(f)$, and thus $ub(C) = \sum_{f\in F_C} w(f)$. If $ub(C) < MaxRS_{\text{ct}}$ holds, we can derive that $\sum_{f\in F_{p^*_C(r)}} w(f) < MaxRS_{\text{ct}}$. Therefore, C contains no MaxRS results and can be safely pruned. □

We consider a cell $AEGH$ in Fig. 5(a) and suppose $MaxRS_{\text{ct}} = 10$. The weight of each facility is 1. The total weight of all facilities in this cell is $\sum_{f\in F_C} w(f) = 9$. There is no MaxRS result in this cell since $MaxRS_{\text{ct}} = 10$ is greater than $ub(C) = 9$. Hence, this cell can be safely pruned.

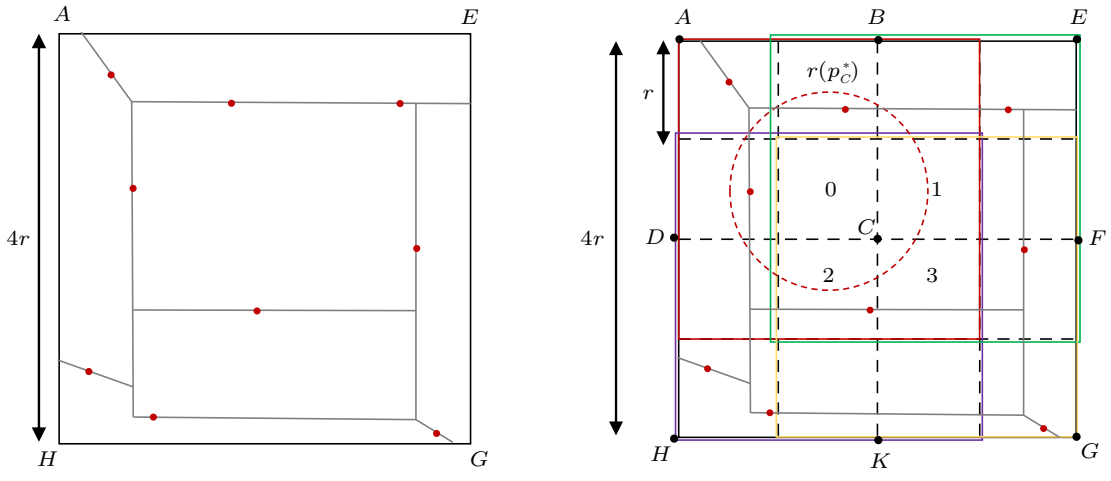
Enhanced Pruning. According to Lemma 3, given the current MaxRS value $MaxRS_{\text{ct}}$, and the upper bound $ub(C)$, any unpromising cell with $ub(C)$ less than $MaxRS_{\text{ct}}$ can be safely pruned without further consideration. However, $ub(C) = \sum_{f\in F_C} w(f)$ is very loose in reality so that only a small part of unpromising cells can be pruned. Hence, to estimate $ub(C)$ more accurately, we further divide the cell into fine-grained unit squares evenly. As shown in Fig.5(b), the cell is $4r\times 4r$, which is further divided into 16 unit squares of size $r\times r$ by six vertical and horizontal dotted lines.

Definition 5 (Largest Intersecting Square). *Given a cell C and user-specified r, let p^*_C be a local optimal result in C. After fine-grained division, the maximal reachable range $r(p^*_C)$ may intersect with nine unit squares at most, and the region composed of them is defined as the largest intersecting square.*

For instance, A, E, G, H in Fig.5(b) are four corners of the cell C. Obviously, the local optimal result p^*_C in this cell must be contained in one of the 16 $r\times r$ unit squares. Without loss of generality, let us suppose that p^*_C is contained in unit square 0, and the red region R_{ul} composed of nine unit squares intersecting with $r(p^*_C)$ is one of the largest intersecting squares. Likewise, the green region, the yellow region, and the purple region are also the largest intersecting squares in C, denoted as $R_{\text{ur}}, R_{\text{ll}}$, and R_{lr}, respectively. For each largest intersecting square, we sum the weights of the facilities located in this region. Then, the following Lemma 4 is derived.

Lemma 4 (Largest Intersecting Square Pruning Strategy). *Supposing $MaxRS_{\text{ct}}$ is the optimal result found so far, for any cell C, if $ub(C) = \max_{R\in\{R_{\text{ul}},R_{\text{ur}},R_{\text{ll}},R_{\text{lr}}\}}\{\sum_{f\in F_R} w(f)\}<MaxRS_{\text{ct}}$ holds, then C can be safely pruned without further consideration.*

Proof. As shown in Fig.5(b), when p^*_C is contained in unit square 0, 1, 2, and 3, $r(p^*_C)$ may intersect with



(a) (b)

Fig.5. Pruning example. (a) Cell. (b) Fine-grained cell.

nine small squares at most. There are four cases of upper bound estimation.

Case 1. If p^*_C is in the region $ABCD$, then $\sum_{f\in F_{p^*_C(r)}} w(f) \leqslant \sum_{f\in F_{R_{\rm ul}}} w(f)$.

Case 2. If p^*_C is in the region $BCFE$, then $\sum_{f\in F_{p^*_C(r)}} w(f) \leqslant \sum_{f\in F_{R_{\rm ur}}} w(f)$.

Case 3. If p^*_C is in the region $CDHK$, then $\sum_{f\in F_{p^*_C(r)}} w(f) \leqslant \sum_{f\in F_{R_{\rm ll}}} w(f)$.

Case 4. If p^*_C is in the region $CKGF$, then $\sum_{f\in F_{p^*_C(r)}} w(f) \leqslant \sum_{f\in F_{R_{\rm lr}}} w(f)$.

In summary, no matter which unit square p^*_C falls into, the following relationship must be satisfied:

$$\sum_{f\in F_{p^*_C(r)}} w(f) \leqslant \max_{R\in\{R_{\rm ul},R_{\rm ur},R_{\rm ll},R_{\rm lr}\}}\{\sum_{f\in F_R} w(f)\}.$$

Thus,

$$ub(C) = \max_{R\in\{R_{\rm ul},R_{\rm ur},R_{\rm ll},R_{\rm lr}\}}\{\sum_{f\in F_R} w(f)\}.$$

In other words, $\max_{R\in\{R_{\rm ul},R_{\rm ur},R_{\rm ll},R_{\rm lr}\}}\{\sum_{f\in F_R} w(f)\}$ is the upper bound estimation of the optimal result in this cell C. If $ub(C)<MaxRS_{\rm ct}$ holds, we can derive that:

$$\sum_{f\in F_{p^*_C(r)}} w(f) \leqslant ub(C) < MaxRS_{\rm ct}.$$

Therefore, any cell with $ub(C)$ less than $MaxRS_{\rm ct}$ will certainly contain no point with the maximal range sum greater than $MaxRS_{\rm ct}$ and the cells can be safely pruned. □

Let us consider the example in Fig.5(b) again. Assume $MaxRS_{\rm ct} = 5$ and the weight of each facility is 1. Based on Lemma 4, the total weight of the facilities in the largest intersecting squares $R_{\rm ul}, R_{\rm ur}, R_{\rm ll}$ and $R_{\rm lr}$ is 4, 4, 4, 3, respectively. Then, $ub(C) = \max\{4,4,4,3\} = 4$, instead of $ub(C) = 9$ by Lemma 3. Hence, this cell can be safely pruned since $MaxRS_{\rm ct} = 5$ is greater than $ub(C) = 4$. However, Lemma 3 cannot prune this cell in this case.

Conspicuously, the upper bound of Lemma 4 is more accurate than that of Lemma 3, and can be used to prune more unpromising cells that do not contain the query result of MaxRS in road networks. Based on the upper bound of the cell, we can further estimate the upper bound of the result in a grid G.

Lemma 5 (Grid Pruning Strategy). *Supposing $MaxRS_{\rm ct}$ is the MaxRS result found so far, for any grid partitioning $G \in \{G_0, G_1, G_2, G_3\}$, $ub(G) = \max_{C \text{ in } G}\{ub(C)\}$ is the upper bound of MaxRS results in a grid G. If $ub(G) < MaxRS_{\rm ct}$ holds, then all cells in G can be safely pruned without further consideration.*

Proof. For any cell C in grid G, the local optimal result existing in C is denoted as p^*_C, and the following equation always holds:

$$\sum_{f\in F_{p^*_C(r)}} w(f) \leqslant \max_{C\in G}\{ub(C)\}.$$

Thus, $ub(G) = \max_{C \text{ in } G}\{ub(C)\}$ is the upper bound of MaxRS results in G. If $ub(G)<MaxRS_{\rm ct}$ holds, we can derive that $\sum_{f\in F_{p^*_C(r)}} w(f)<MaxRS_{\rm ct}$. Therefore, any cell in G must not contain MaxRS results greater than $MaxRS_{\rm ct}$ and all cells can be pruned. □

Based on Lemma 5, we can directly prune all cells in a grid that do not contain query results. After the upper bound $ub(G)$ is computed, grids are processed by the descending order of $ub(G)$. Intuitively, the grid with a greater $ub(G)$ has a higher probability containing MaxRS results. Note that the upper bound estimation can be computed efficiently while performing grid partitioning. Before the query starts, we can heuristically select a cell with the largest $ub(C)$ to get $MaxRS_{\rm ct}$ for pruning. After the pruning is completed, the remaining promising cells are utilized for the subsequent MaxRS query in phase 2.

4.3 Phase 2: Computing Optimal Results

The first phase of GAM returns a set of promising cells which may contain MaxRS results. This phase conducts parallel MaxRS queries in all promising cells to get all MaxRS results. It is worth mentioning that phase 1 is usually efficient and occupies a small fraction of the overall query processing time. In contrast, the MaxRS query in phase 2 is typically very expensive to compute. We hence focus on accelerating phase 2 by GPU. For efficient performance, we first design a GPU-friendly storage structure CRN, and then propose a two-level parallel framework based on it.

4.3.1 GPU-Friendly Storage Structure

In order to facilitate efficient access to the road network data within each cell in GPU, we propose the CRN based on the cells partitioned by the grid, which covers the entire road network. Specifically, CRN consists of three major components: a cell-based header table $\mathcal{H}$, an array-based road network adjacency list $\mathcal{L}$ and an array-based road network facility list $\mathcal{F}$. We associate the table $\mathcal{H}$ with the list $\mathcal{L}$ to access any edge or node in each cell efficiently.

1) *Cell-Based Header Table.* We first devise a cell-based header table $\mathcal{H}$ consisting of all the cells in the ascending order of cell identifiers to store the concise information of the road network data in each cell. Each element in $\mathcal{H}$ is a quadruple of $(cid, offset, length, upper)$, where cid is the identifier of a cell. For any cell $C_{i,j}$ in G_i, its cid can be calculated according to the formula $cid = j \times GridDimX + i$, where $GridDimX$ is the number of cells in a row of the grid G_i. Furthermore, $offset$ and $length$ specify a continuous part in the vertex array of $\mathcal{L}$ starting from $offset$ to $offset + length - 1$, in which all the nodes are within C_{cid}. $upper$ is the upper bound of the MaxRS result in C_{cid}, which is estimated according to the pruning strategy in Subsection 4.2.2 to directly prune cells without the MaxRS result.

2) *Array-Based Road Network Adjacency List.* Although the nodes of the road network are spatially correlated, the degree of each node of the road network is usually not very large (generally less than 5). Namely, in real life, most road networks are sparse graphs. Due to the limited GPU global memory size, we utilize the adjacency list to represent the road network, and the space complexity is only $O(|V| + |E|)$. In the CUDA architecture, the global memory is regarded as an array, which can be efficiently read and written. Based on the above analysis, we design an array-based adjacency list to store the road network inside each cell, consisting of the vertex array, the edge array, the length array, the lock array and the segment list. Specifically, the vertex array and the edge array store all the vertices and edge information inside the cell respectively, and the length of each edge is recorded in the length array. In order to avoid multi-thread competition and ensure the thread safety, we use the lock array to record whether the segment list corresponding to each edge is performing read and write operations. In addition, the segment generated by the facility on each edge is stored in the segment list. Note that since each edge must appear twice in the edge array, we only attach a segment list behind one of the corresponding edges to save memory.

3) *Facility List.* The facility list stores all facilities covered by each cell in the road network continuously. Each facility f is a triplet of $(fid, position, weight)$, representing the detailed information of f as defined in Section 3. Note that facility requests from CUDA threads can be coalesced in Subsection 4.3.2, because facilities in the same cell are stored continuously in the global memory.

For example, given the query radius $r = 0.5$, according to the partition steps in Subsection 4.2.1, the road network is partitioned into cells by grid G_0 in Fig.6. For cell $C_{0,0}$, we can obtain its $cid = 0$ by the mapping function, and further index into the cell-based header table $\mathcal{H}$ to obtain the concise information $(0, 0, 4, 3)$ of the road network in $C_{0,0}$. Then, we use $offset$ and $length$ to index the detailed information of $C_{0,0}$ in the array-based adjacency list $\mathcal{L}$. The red dashed area and the blue dashed area are the specific information of the road network subgraphs $G(C_{0,0})$ and $G(C_{1,0})$, respectively. All facilities are kept in the facility list. f_1, f_2, f_{13} are located in $C_{0,0}$ and f_3, f_4 are located in $C_{1,0}$.

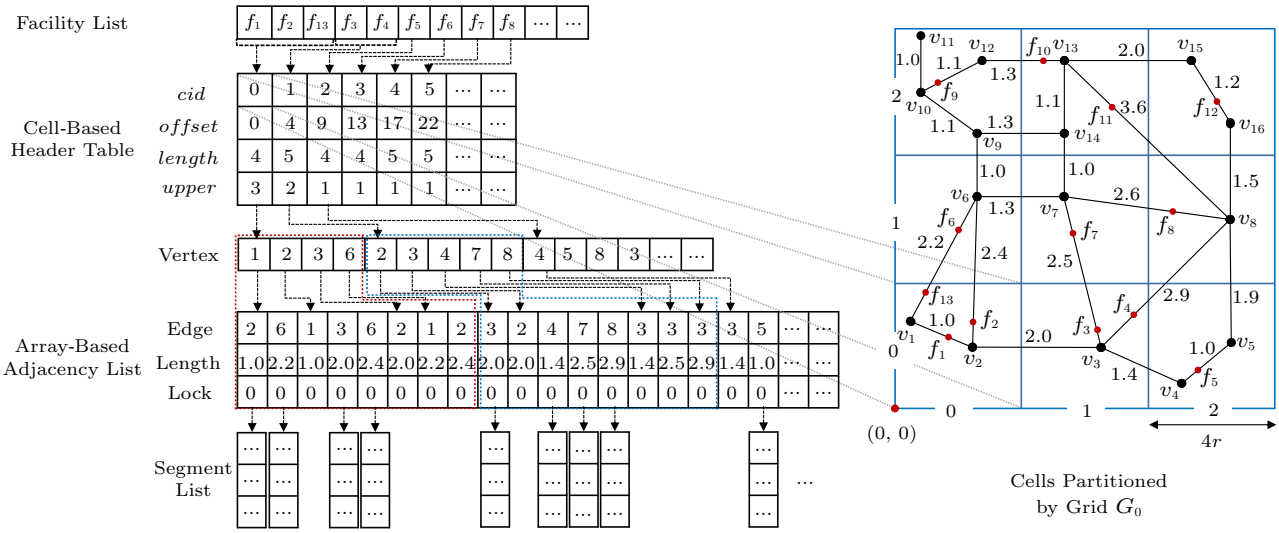


Fig.6. CRN example.

4.3.2 Two-Level Parallel Framework

The input of this step consists of the promising cells and the query radius r. The output is composed of all MaxRS results from all promising cells. The lock array and segment list is initially set to 0 s. After segment generation, it will be parsed by line sweep to get all MaxRS results contained in the promising cells.

A straightforward way to perform the MaxRS query on GPU is to treat each promising cell query independently using one CUDA thread. However, the efficiency of this parallelism may vary a lot with user-specified r. For example, when r increases, the number of cells $N = \lceil \frac{l}{4r} \rceil \times \lceil \frac{w}{4r} \rceil$ will rapidly decrease, which obviously leads to a degradation of the parallel performance. Considering the GPU hardware architecture, the GPU is composed of dozens of SMs, and each SM contains multiple SPs (CUDA cores). When a CUDA program is running on the GPU, CUDA blocks are handled by SMs concurrently, and all threads of a CUDA block execute on SPs within an SM in parallel. Therefore, only mapping promising cells to CUDA threads cannot fully utilize the hardware resources of GPU, resulting in poor parallel performance. A two-level parallel query framework is thus proposed to address these problems.

As shown in Fig.7, we first distribute each promising cell in the road network grid to a CUDA block (the first level) and there are many facilities in each promising cell. For MaxRS queries in road networks, the cost of the segment generation may become the bottleneck of query processing, as analyzed in [16, 17]. We hence handle the segment generation of a facility by a CUDA thread to further accelerate the MaxRS query within a promising cell (the second level), which forms a two-level parallel query framework as follows.

1) *Cell-Level Parallelism.* After the road network is partitioned into cells, we distribute the computation of cells among CUDA blocks, and thus the processing between cells is in parallel.

2) *Facility-Level Parallelism.* Within each cell, the MaxRS query in road networks is further decomposed into two independent components, namely, segment generation [17] and line sweep [32]. Thus, the segment generation for facilities can be performed by CUDA threads concurrently, and then we integrate all local results in each cell to get final results by line sweep.

We first organize the promising cells in the grid into CRN and transfer it to the global memory of the GPU. Note that the road network subgraph $G(C)$ covered by a cell C is stored continuously within CRN in the global memory of the GPU. We then perform MaxRS queries for the promising cells on the GPU in parallel. Specifically, we assign facilities covered by a promising cell to one CUDA block (handled by a GPU SM), which will then be distributed uniformly among CUDA threads. In a CUDA block, for each facility f, a CUDA thread tries to generate segments for f along the road network until the distance reaches query radius r. In our implementation, like the GPU accelerated graph search, we maintain a stack for each facility recording the nodes to be expanded and the remaining network radius in the global memory of the GPU. For the generated segment s in edge e, we first check if the lock bit of e is 0. If so, we set this lock bit to 1, then s is recorded into the segment list of this edge, and the merge strategy in [17] is used to merge these existing segments with the new segment. Note that in this step, data requests from CUDA threads can be coalesced, because facilities and nodes in the same promising cell are stored continuously in the global memory.

After the segment generation is completed, we transfer the segment list of CRN back to the CPU,

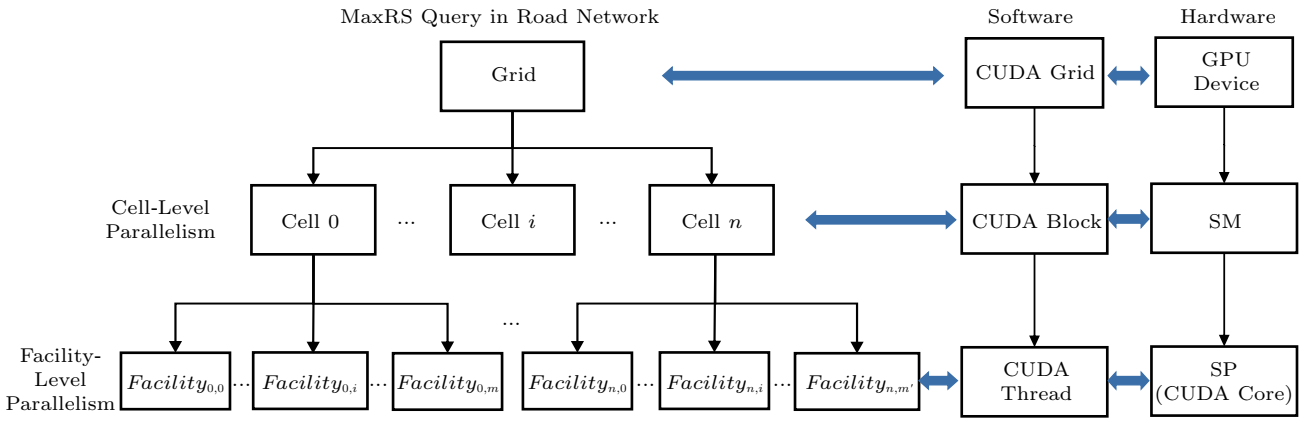


Fig.7. Two-level parallel query framework.

and the CRN is cleared out of the global memory. We find the local result in each edge's segment list by line sweep [32], which is the line version of algorithm plane sweep. Then we put the result with the maximal total weight computed so far into set P^*. Finally, P^* containing all MaxRS results in the network will be returned.

To facilitate efficient access to facilities in one promising cell, we can directly store them in a smaller but faster shared memory associated with each GPU SM when r is relatively small. It is worth noting that our query framework is hardware-conscious and can be easily extended to multiple GPUs by assigning a grid $G_i \in \{G_0, G_1, G_2, G_3\}$ to one GPU, which can further improve parallel query performance. In short, our query framework not only makes better use of GPU resources, but also effectively solves the MaxRS problem in road networks. We use CUDA proposed by NVIDIA as the programming model on the GPU.

4.4 Algorithm Details

In this subsection, we provide some details of GAM in Algorithm 1. It takes a query radius r and a road network dataset G as input and returns a set P^* with the maximum range sum.

Algorithm 1. GAM

```
   Input: query radius r, road network G;
   Output: MaxRS result set P*;
1  P* ← ∅;
2  for seedPoint_i ∈ {(0,0),(0,2r),(2r,0),(2r,2r)} do
3      G_i ← GridPartition(G, seedPoint_i);
4      Pruning(G_i);
5      Copy CRN to device;
6      for cell in G_i parallel do                    // Cell-level
       Parallelism
7          for f ∈ F_cell parallel do                 // Facility-level
           Parallelism
8              GenerateSegment(f);
9          end
10     end
11     for e ∈ E do                  // Update P* by line sweep
12         P' ← LineSweep(e.segmentList);
13         if P'.MaxRS > P*.MaxRS then
14             P* ← P';
15         end
16         else if P'.MaxRS = P*.MaxRS then
17             P* ← P* ∪ P';
18         end
19     end
20 end
21 return P*;
```

P^* is initialized to an empty set in line 1. In lines 2 and 3, for any $seedPoint_i \in \{(0,0),(0,2r),(2r,0),(2r,2r)\}$, GAM calls *GridPartition*, introduced in Subsection 4.2.1, to partition the whole road network G into many cells and estimate the upper bound of the maximum range sum for each cell in the meanwhile. In line 4, we prune the unpromising cells in G_i that do not contain any optimal result by comparing $ub(C)$ with $MaxRS_{\text{ct}}$. CRN is constructed for remaining promising cells and copied to the GPU's global memory in line 5.

In lines 6–10, promising cells in G_i are processed in parallel. Within each cell, we invoke the method *GenerateSegment* in [17] to generate segments for each facility f concurrently, and insert segments into the *SegmentList* of CRN. In lines 11 and 12, we use the existing method *LineSweep* in [32] to find the local optimal results in each edge. Then, we update the result with the maximal total weight computed so far into set P^* in lines 13–18. Finally, P^* containing all MaxRS results in the network will be returned in line 21.

We omit the detailed description of *SegmentList* and *LineSweep* here. Next, we will prove the correctness of GAM.

Theorem 1. *GAM (i.e., Algorithm 1) returns all optimal results of the MaxRS query in road networks correctly.*

Proof. Intuitively, the correctness of GAM is guaranteed by the following three aspects.

1) Based on Lemma 1 and Lemma 2, for any optimal result p^* on an edge $e \in E$, there must be a cell C in $G_i \in \{G_0, G_1, G_2, G_3\}$ returning p^*.

2) Based on Lemma 4 and Lemma 5, the pruning strategies prune the unpromising cells not containing any optimal result p^* correctly.

3) For a promising cell C, after segment generation, the line sweep can compute the total weight of points in this cell exactly, and the result with the maximal total weight is always kept in P^*.

To sum up, all optimal results of the MaxRS query are kept in P^* returned by GAM. □

5 Experiment

5.1 Datasets and Experimental Settings

Platform. All experiments are implemented in Python 3.8.5 with CUDA 8.0. The experiments are executed on DELL Precision Tower 3620 Workstation (Intel® Xeon® E3-1225 v6 @ 3.30 GHz 3.31 GHz (4 cores) + 32 G RAM + 64-bit windows 10). We equip the computer with a NVIDIA GeForce RTX 2080 graphics card, which has 2 944 CUDA cores and 11 G graphical memory.

Datasets. In the experiment, considering various real application scenarios, we use two real road network datasets with different data scales, ranging from the county to the state. They are Oldenburg (OL) [33] and California (CA) road networks. OL is generated based on the real road network of Oldenburg with 100 km^2 in [33], which contains 6 105 vertices and 7 035 edges. The average degree of the nodes in this network is 2.304. CA is a popular and widely-used real road network dataset, which is generated based on the real road network of California State in USA with 100 000 km^2 and contains 87 635 real facilities. This dataset is obtained from [34], which contains 21 048 vertices and 21 693 edges. Table 4 provides more detailed information of the road network datasets. In order to facilitate data processing, we normalize the coordinate range of all datasets to [0, 100 00].

Table 4. Cardinalities of Real Datasets

Dataset	Number of Nodes	Number of Edges	Average Length	Average Degree
OL	6 105	7 035	73.679	2.304
CA	21 048	21 693	0.016	2.061

Same as the previous work [16, 17], the facilities are generated randomly in the road network with uniform distribution in terms of the road network distance, and their weights are within the range of (0, 50].

Competitors. Since our GPU acceleration method refers to the idea of segment generation, we compare GAM with the segment generation based algorithm [17] (denoted as SEG) mentioned before. Although the method in [17] is based on external memory storage, we implement it in memory to make the experiment more reasonable and optimize it by pruning redundant segment generation. To illustrate the effectiveness of our multi-grained pruning technique and two-level parallel query framework, we also consider two variants of GAM: GAM-noPrun and GAM-CPU as follows.

• *GAM.* The GPU-accelerated method for MaxRS queries in road networks is based on our two-level parallel query framework, equipped with the grid partitioning technique, and the multi-grained pruning technique.

• *GAM-noPrune.* This is GAM without the multi-grained pruning technique, where all cells in $\{G_1, G_2, G_3, G_4\}$ need to be processed.

• *GAM-CPU.* This is the CPU version of GAM using multi-threading, where each cell in $\{G_1, G_2, G_3, G_4\}$ is assigned to a thread running on the CPU (i.e., only one-level parallel) for segment generation and line sweep.

• *SEG* [17]. This method extends the rectangular intersection of the traditional query into the intersection of segments in the road network, consisting of two steps: segment generation and line sweep.

Metrics. For efficiency comparison we collect the following measures:

• execution time: the wall-clock elapsed time;

• speedup: the ratio of the running time of algorithm A (before acceleration) and algorithm B (after acceleration), namely: $\frac{T_B}{T_A}$;

• pruning ratio: let $|F|_{\text{before}}$ and $|F|_{\text{after}}$ be the numbers of facilities for generating segments before and after the pruning respectively. This metric is computed by $\frac{|F|_{\text{before}}-|F|_{\text{after}}}{|F|_{\text{before}}}$.

Parameters. In the experiments, we evaluate the performance of GAM in terms of several aspects: the effect of query radius r (efficiency), the effect of the number of facilities $|F|$ (scalability), and the effect of pruning (pruning). Let R be the coordinate range of the road network, and we set the query radius $r = \theta \times R$, where θ is changed among 0.5%, 1%, 2%, 3% and 4%. Apart from this, we vary $|F|$ among 20k, 40k, 60k, 80k and 100k. The concrete settings of parameters are shown in Table 5 and the default values are underlined. All the parameters in the datasets and queries use the default values (if not explicitly specified). Note that, before querying, the road network is pre-loaded into memory.

Table 5. Parameter Settings

Parameter	Description	Used Value
θ	Query range size	0.5%, 1%, <u>2%</u>, 3%, 4%
$\|F\|$	Number of facilities	20k, 40k, <u>60k</u>, 80k, 100k

5.2 Experiment 1: Effect of Query Range Size

The first experiment evaluates the performance of GAM, GAM-CPU and SEG on varying the query range size θ from 0.5% to 4%. Fig.8 illustrates the experimental results conducted on OL and CA.

As depicted in Fig.8(a) and Fig.8(b), with a greater value of θ, the query time of the SEG increases rapidly, and its growth trend is approximately super-linear. Note that when θ is particularly small (e.g., $\theta = 0.5\%$), the efficiency of GAM-CPU is roughly the same as that of GAM which is about one order of magnitude faster than SEG. However, with the increase of θ (e.g., $\theta = 4\%$), the time of GAM-CPU quickly increases to approach that of SEG. In contrast, no matter how θ changes, the time of GAM grows slowly.

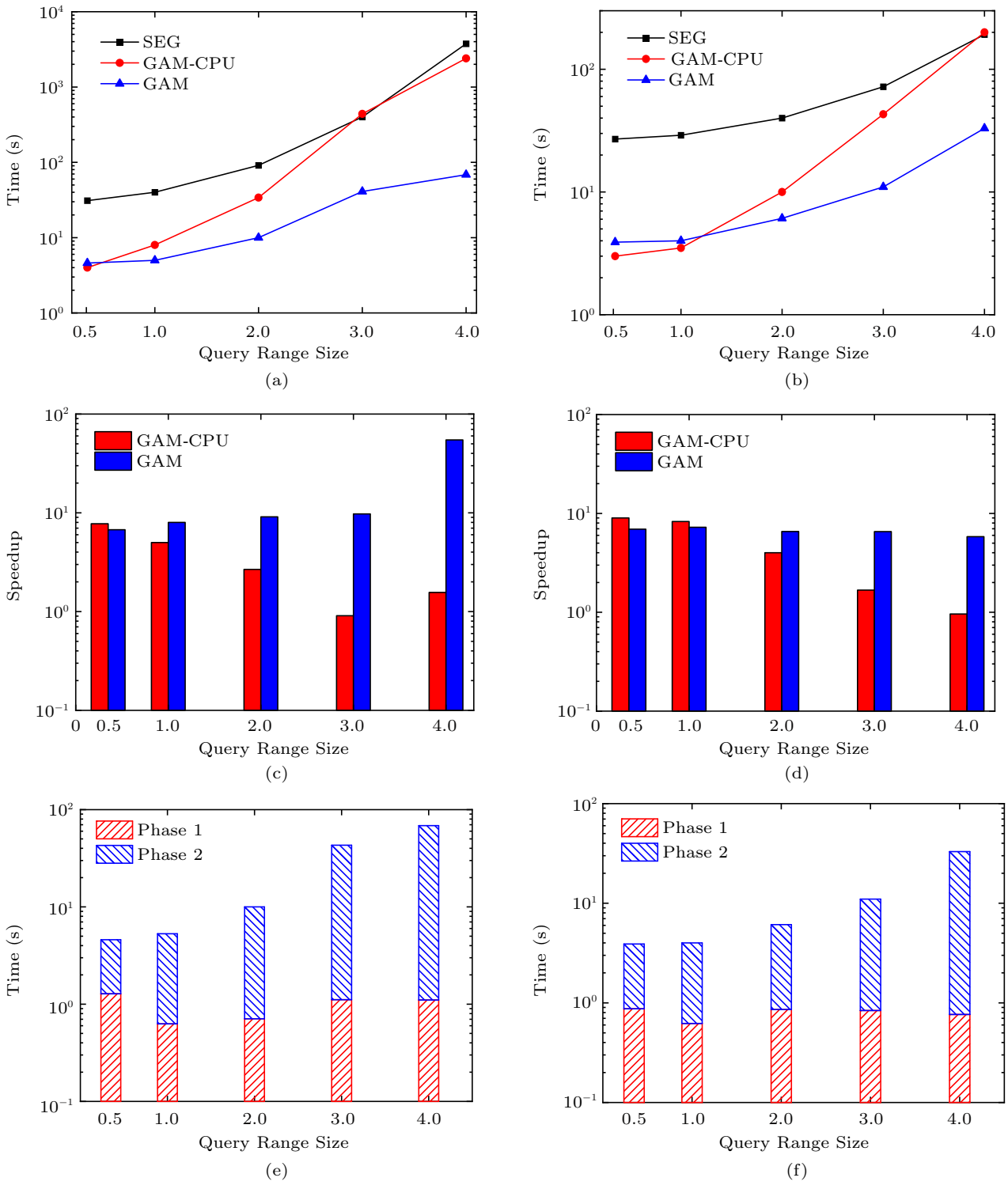


Fig.8. Effect of the query range size. (a) CA query time. (b) OL query time. (c) CA speedup. (d) OL speedup. (e) CA decomposition. (f) OL decomposition.

This is because when θ is very small (e.g., $\theta = 0.5\%$), the data is partitioned into many cells by a grid, and the number of segments that need to be generated and swept is very small. Therefore, the advantage of our two-level parallelism is not significant, and the parallel efficiency of GAM-CPU is roughly the same as that of GAM. Besides, there are additional overheads in the GAM algorithm such as grid partitioning and data

transmission between GPU and CPU. Therefore, GAM-CPU is even slightly faster than GAM. However, with θ increases, the number of cells decreases rapidly, resulting in a rapid growth of GAM-CPU query time.

As shown in Fig.8(c) and Fig.8(d), with the increase in θ, the speedup of GAM becomes more and more obvious, while the speedup of GAM-CPU has an apparently downward trend. As analyzed in Subsection 4.3.2, the number of cells partitioned by the grid decreases rapidly as r increases, which obviously leads to a rapid decrease in the speedup of GAM-CPU only with one-level parallelism. Conversely, due to two-level parallelism for each facility in a cell, GAM still maintains a high and stable speedup.

Fig.8(e) and Fig.8(f) show the query time decomposition of our algorithm GAM with the growing of θ. It consists of two parts: phase 1 and phase 2. Experimental results show that phase 2 accounts for the majority of the total time. It also demonstrates that the cost of phase 1 basically levels off and phase 2 spends more time with the growing of θ. Apparently, the cost of phase 1 (i.e., grid partitioning and pruning) can be ignored for the MaxRS query in road networks.

5.3 Experiment 2: Effect of the Number of Facilities

The second experiment evaluates the performance of GAM, GAM-CPU and SEG on the varying number of facilities $|F|$ (i.e., scalability) from 20k to 100k. Fig.9 illustrates the experimental results conducted on OL and CA.

As depicted in Fig.9(a) and Fig.9(b), when the number of facilities increases, the query time of the SEG grows rapidly, while the running time of the GAM-CPU increases relatively slowly. Apparently, our algorithm GAM always outperforms all the other algorithms in terms of the query time. In addition, our algorithm has a nearly linear scalability. This shows that GAM based on the two-level parallel framework has better scalability than GAM-CPU based on the cell-level parallelism.

As shown in Fig.9(c) and Fig.9(d), with the increase in the number of facilities, the acceleration of GAM becomes more and more obvious, while the speedup of GAM-CPU has not changed significantly. This is because the number of cells has not changed with the growing of $|F|$. Thus the one-level parallelism is not affected by the number of facilities. But at this time, there are more facilities in each cell, and GAM generates segments in parallel for each facility. This reflects the advantages of our two-level parallel framework. Compared with the CA road network, the OL road network is smaller in scale. At the same $|F|$, the OL speedup is smaller than the CA speedup.

Fig.9(e) and Fig.9(f) give the query time decomposition of our algorithm GAM with the growing of $|F|$. It consists of two parts: phase 1 and phase 2. Experimental results show that phase 2 accounts for the majority of the total time. It also indicates that the cost of phase 1 basically levels off and phase 2 spends more time with the growing of $|F|$. Obviously, the cost of phase 1 (i.e., grid partitioning and pruning) can be neglected for the MaxRS query in road networks.

5.4 Experiment 3: Effect of Pruning Strategy

The third experiment evaluates the performance of the pruning strategy on OL and CA.

1) *Effect of θ on the Pruning Strategy.* Fig. 10 gives the pruning performance of GAM on road network datasets by varying the query range size θ from 0.5% to 4%. Specifically, as depicted in Fig.10(a) and Fig.10(b), with θ increasing, GAM-noPrune grows approximately super-linearly. In clear contrast, GAM grows more slowly than GAM-noPrune and has similar trends to GAM-noPrune. Fig.10(c) and Fig.10(d) reflect that the enhanced pruning strategies (i.e., LIS and the grid pruning strategy) are much better than the naive pruning strategy all the time. The pruning ratio of both has a slight downward trend with the increasing of θ. The reason is that the upper bound of the optimal result in the cell or grid becomes looser as θ becomes larger, which is consistent with our expectation.

2) *Effect of $|F|$ on the Pruning Strategy.* Fig.11 investigates the pruning performance of the algorithms on road network datasets with the variation of the number of facilities $|F|$ from 20k to 100k. Conspicuously, as shown in Fig. 11(a) and Fig. 11(b), our algorithm GAM and GAM-noPrune have a nearly linear scalability. Moreover, GAM is further improved by 2x–3x of GAM-noPrune, and the growth trend of GAM is slower. In Fig.11(c) and Fig.11(d), the enhanced pruning strategies (i.e., LIS and the grid pruning strategy) are much better than the naive pruning strategy all the time. When $|F|$ increases, the pruning ratio of both has a little fluctuation while basically levels off. This is because the accuracy of the upper bound of the optimal result in the cell or grid is not affected by $|F|$. This is consistent with our expectation.

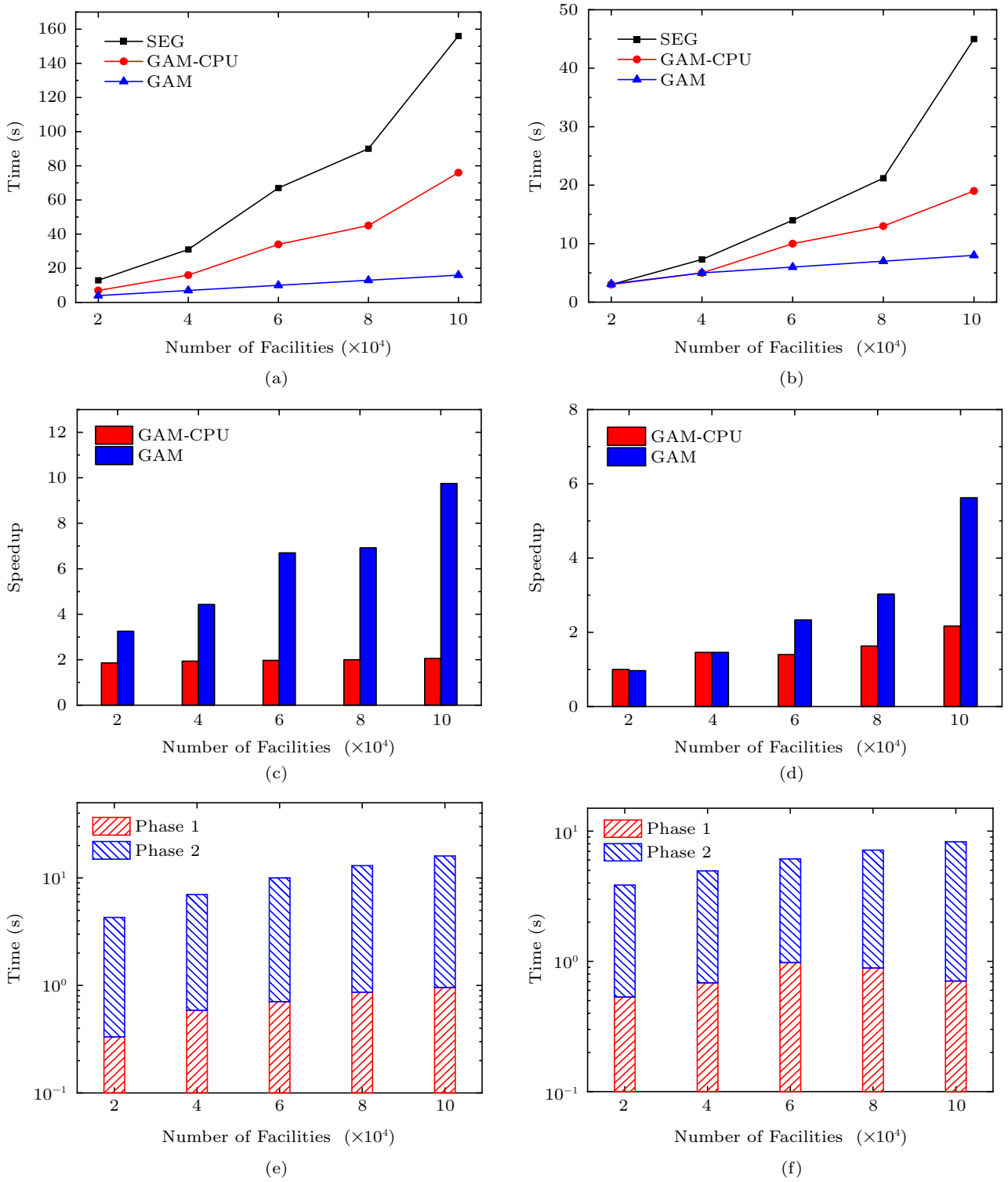


Fig. 9. Effect of the number of facilities. (a) CA query time. (b) OL query time. (c) CA speedup. (d) OL speedup. (e) CA decomposition. (f) OL decomposition.

5.5 Summary

As illustrated in the experiments, in terms of query efficiency, our algorithm GAM outperforms the other algorithms significantly.

Under the same query range size θ, the performance promotion of GAM is up to about one order of magnitude. Under the same number of facilities $|F|$, with

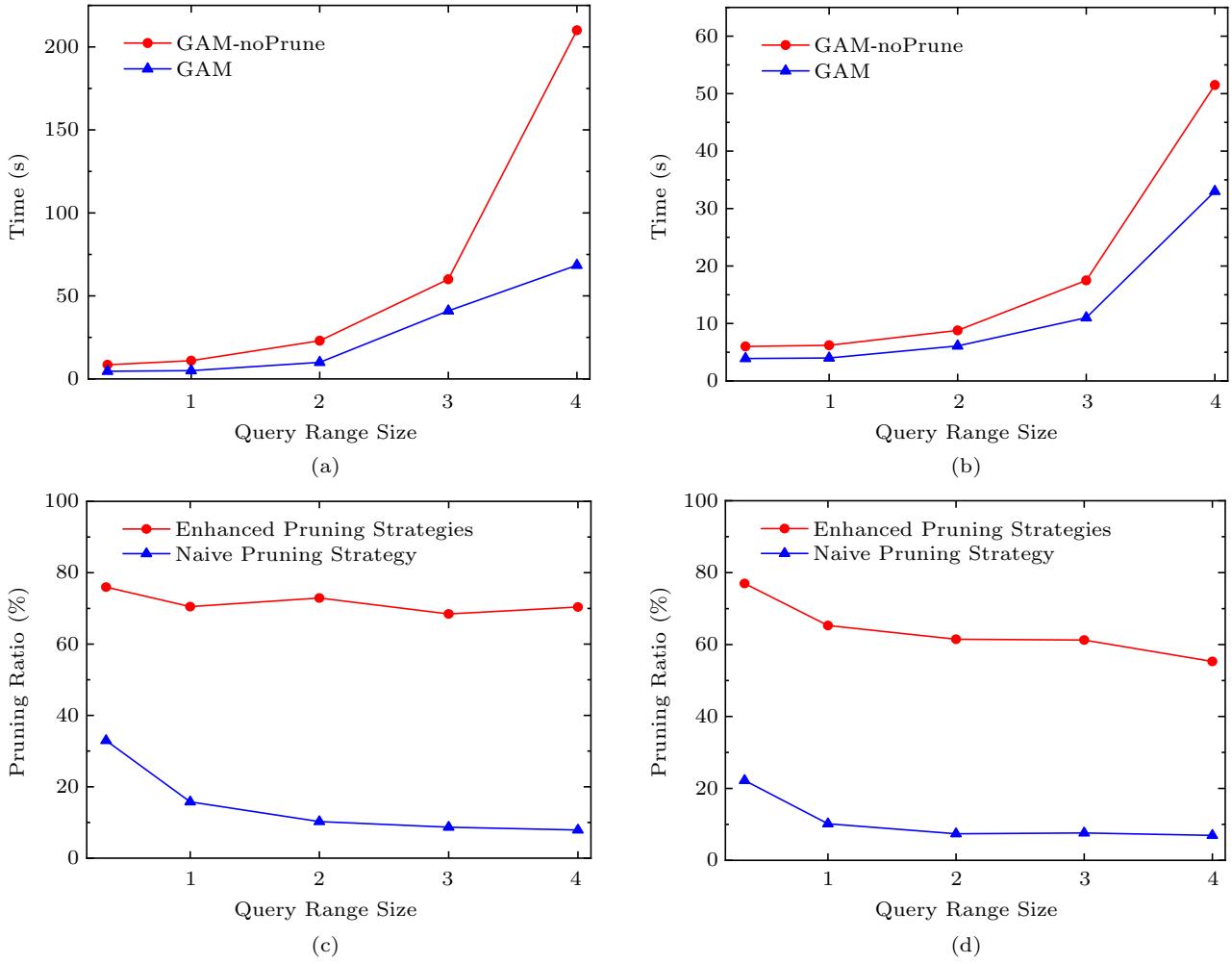


Fig.10. Effect of θ on the pruning ratio. (a) CA query time. (b) OL query time. (c) CA pruning ratio. (d) OL pruning ratio.

a greater number of facilities $|F|$, GAM has a nearly linear scalability and the speedup of GAM is getting higher and higher. Decomposing the query time of GAM, we find that the cost of phase 1 basically levels off and phase 2 spends more time with the growing of θ or $|F|$. Phase 2 accounts for the majority of the total time, in contrast, the cost of phase 1 can be ignored for the MaxRS query in road networks.

6 Conclusions

In this paper, we studied the problem of MaxRS queries in road networks. We showed that existing approaches had a lot of limitations, such as scalability and efficiency. Then, we presented GAM, a novel GPU-accelerated algorithm, which can efficiently handle MaxRS queries based on the proposed two-level parallel framework. The experimental results demonstrated that GAM is on average 10 times faster than state-of-the-art competitors, and the maximum speedup can achieve about 55 times. Enhanced pruning strategies save about 60%–80% computation overhead for Oldenburg and California road networks respectively compared with the native pruning strategies, which further improves the efficiency by 2–3 times over GAM-noPrune.

References

[1] Manyika J, Chui M, Brown B *et al.* Big data: The next frontier for innovation, competition, and productivity. Technical Report, McKinsey Global Institute, 2011. https://www.mckinsey.com/˜/media/mckinsey/business%20functions/mckinsey%20digital/our%20insights/big%20data%20the%20next%20frontier%20for%20innovation/mgi_big_data_full_report.pdf, Jul. 2022.

[2] Tong Y, Zeng Y, Zhou Z, Chen L, Ye J, Xu K. A unified approach to route planning for shared mobility. *Proc. VLDB Endow.*, 2018, 11(11): 1633-1646. DOI: 10.14778/3236187.3236211.

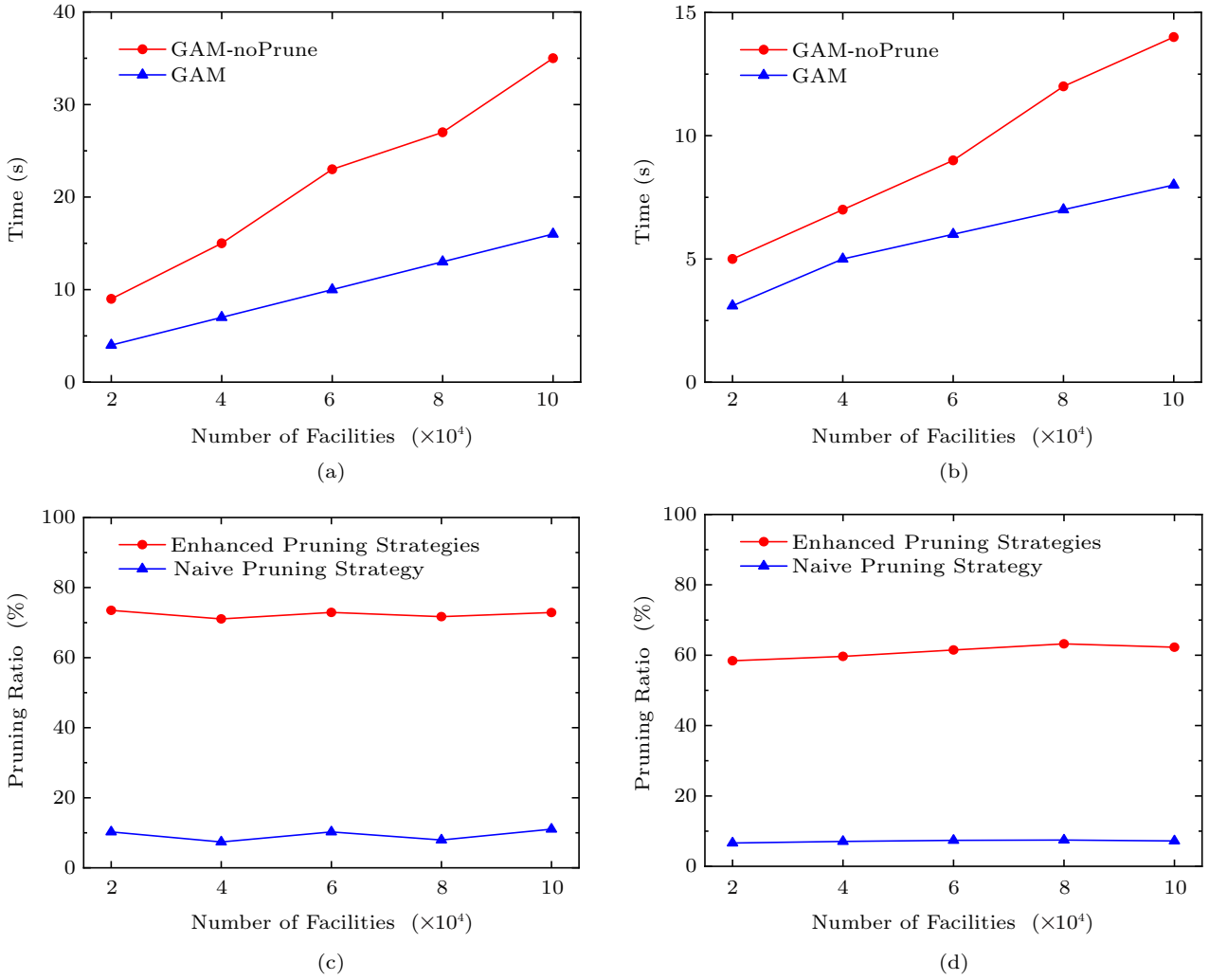


Fig.11. Effect of $|F|$ on the pruning ratio. (a) CA query time. (b) OL query time. (c) CA pruning ratio. (d) OL pruning ratio.

[3] Choi D, Chung C, Tao Y. A scalable algorithm for maximizing range sum in spatial databases. *Proc. VLDB Endow.*, 2012, 5(11): 1088-1099. DOI: 10.14778/2350229.2350230.

[4] Choi D, Chung C, Tao Y. Maximizing range sum in external memory. *ACM Trans. Database Syst.*, 2014, 39(3): Article No. 21. DOI: 10.1145/2629477.

[5] Abellanas M, Hurtado F, Icking C, Klein R, Langetepe E, Ma L, Palop B, Sacristán V. Smallest color-spanning objects. In *Proc. the 9th Annual European Symposium on Algorithms*, Aug. 2001, pp.278-289. DOI: 10.1007/3-540-44676-1_23.

[6] Tiwari S, Kaushik S. Extracting region of interest (ROI) details using LBS infrastructure and web-databases. In *Proc. the 13th IEEE International Conference on Mobile Data Management*, Jul. 2012, pp.376-379. DOI: 10.1109/MDM.2012.29.

[7] Chai C, Fan J, Li G. Incentive-based entity collection using crowdsourcing. In *Proc. the 34th IEEE International Conference on Data Engineering*, Apr. 2018, pp.341-352. DOI: 10.1109/ICDE.2018.00039.

[8] Chai C, Li G, Li J, Deng D, Feng J. Cost-effective crowdsourced entity resolution: A partial-order approach. In *Proc. the 2016 International Conference on Management of Data*, Jun. 26-Jul. 1, 2016, pp.969-984. DOI: 10.1145/2882903.2915252.

[9] Li G, Chai C, Fan J, Weng X, Li J, Zheng Y, Li Y, Yu X, Zhang X, Yuan H. CDB: Optimizing queries with crowd-based selections and joins. In *Proc. the 2017 International Conference on Management of Data*, May 2017, pp.1463-1478. DOI: 10.1145/3035918.3064036.

[10] Tao Y, Hu X, Choi D, Chung C. Approximate MaxRS in spatial databases. *Proc. VLDB Endow.*, 2013, 6(13): 1546-1557. DOI: 10.14778/2536258.2536266.

[11] Hussain M M, Islam K A, Trajcevski G, Ali M E. Towards efficient maintenance of continuous MaxRS query for trajectories. In *Proc. the 20th International Conference on Extending Database Technology*, Mar. 2017, pp.402-413. DOI: 10.5441/002/edbt.2017.36.

[12] Liu Q, Lian X, Chen L. Probabilistic maximum range-sum queries on spatial database. In *Proc. the 27th ACM*

SIGSPATIAL International Conference on Advances in Geographic Information Systems, Nov. 2019, pp.159-168. DOI: 10.1145/3347146.3359376.

[13] Hussain M M, Mostafiz M I, Mahmud S M F, Trajcevski G, Ali M E. Conditional MaxRS query for evolving spatial data. *Frontiers Big Data*, 2020, 3: Article No. 20. DOI: 10.3389/fdata.2020.00020.

[14] Yiu M L, Mamoulis N. Clustering objects on a spatial network. In *Proc. the ACM SIGMOD International Conference on Management of Data*, Jun. 2004, pp.443-454. DOI: 10.1145/1007568.1007619.

[15] Nandy S C, Bhattacharya B B. A unified algorithm for finding maximum and minimum object enclosing rectangles and cuboids. *Computers & Mathematics with Applications*, 1995, 29(8): 45-61. DOI: 10.1016/0898-1221(95)00029-X.

[16] Zhou X, Wang W. An index-based method for efficient maximizing range sum queries in road network. In *Proc. the 27th Australasian Database Conference*, Sept. 2016, pp.95-109. DOI: 10.1007/978-3-319-46922-5_8.

[17] Phan T K, Jung H, Kim U M. An efficient algorithm for maximizing range sum queries in a road network. *The Scientific World Journal*, 2014, 2014: Article No. 541602. DOI: 10.1155/2014/541602.

[18] He B, Yang K, Fang R, Lu M, Govindaraju N, Luo Q, Sander P. Relational joins on graphics processors. In *Proc. the 2008 ACM SIGMOD International Conference on Management of Data*, Jun. 2008, pp.511-524. DOI: 10.1145/1376616.1376670.

[19] He B, Lu M, Yang K, Fang R, Govindaraju N K, Luo Q, Sander P V. Relational query coprocessing on graphics processors. *ACM Transactions on Database Systems*, 2009, 34(4): Article No. 21. DOI: 10.1145/1620585.1620588.

[20] Bogh K S, Chester S, Assent I. Work-efficient parallel skyline computation for the GPU. *Proc. VLDB Endow.*, 2015, 8(9): 962-973. DOI: 10.14778/2777598.2777605.

[21] Dong K, Zhang B, Shen Y, Zhu Y, Yu J. GAT: A unified GPU-accelerated framework for processing batch trajectory queries. *IEEE Transactions on Knowledge and Data Engineering*, 2020, 32(1): 92-107. DOI: 10.1109/TKDE.2018.2879862.

[22] Imai H, Asano T. Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *Journal of Algorithms*, 1983, 4(4): 310-323. DOI: 10.1016/0196-6774(83)90012-3.

[23] Amagata D, Hara T. Monitoring MaxRS in spatial data streams. In *Proc. the 19th International Conference on Extending Database Technology*, Mar. 2016, pp.317-328. DOI: 10.5441/002/edbt.2016.30.

[24] Chen Z, Liu Y, Wong R C W, Xiong J, Cheng X, Chen P. Rotating MaxRS queries. *Information Sciences*, 2015, 305: 110-129. DOI: 10.1016/j.ins.2015.02.009.

[25] Feng K, Cong G, Bhowmick S S, Peng W, Miao C. Towards best region search for data exploration. In *Proc. the 2016 International Conference on Management of Data*, Jun. 26-Jul. 1, 2016, pp.1055-1070. DOI: 10.1145/2882903.2882960.

[26] Chen Z, Yuan Q, Liu W. Monitoring best region in spatial data streams in road networks. *Data Knowl. Eng.*, 2019, 120: 100-118. DOI: 10.1016/j.datak.2019.03.002.

[27] Liu J, Chai C, Luo Y, Lou Y, Feng J, Tang N. Feature augmentation with reinforcement learning. In *Proc. the 38th IEEE International Conference on Data Engineering*, May 2022, pp.3360-3372. DOI: 10.1109/ICDE53745.2022.00317.

[28] Chai C, Cao L, Li G, Li J, Luo Y, Madden S. Human-in-the-loop outlier detection. In *Proc. the 2020 ACM SIGMOD International Conference on Management of Data*, Jun. 2020, pp.19-33. DOI: 10.1145/3318464.3389772.

[29] Chai C, Liu J, Tang N, Li G, Luo Y. Selective data acquisition in the wild for model charging. *Proc. VLDB Endow.*, 2022, 15(7): 1466-1478. DOI: 10.14778/3523210.3523223.

[30] Cook S. CUDA Programming: A Developer's Guide to Parallel Computing with GPUs (1st edition). Morgan Kaufmann, 2012.

[31] Xu Y, Wang R, Goswami N, Li T, Qian D. Software transactional memory for GPU architectures. *IEEE Comput. Archit. Lett.*, 2014, 13(1): 49-52. DOI: 10.1109/L-CA.2013.4.

[32] De Berg M, Van Kreveld M, Overmars M, Schwarzkopf O. Computational Geometry: Algorithms and Applications. Springer, 2000.

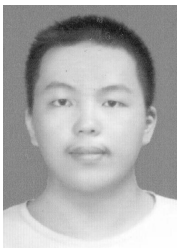
[33] Brinkhoff T. A framework for generating network-based moving objects. *GeoInformatica*, 2002, 6(2): 153-180. DOI: 10.1023/A:1015231126594.

[34] Li F, Cheng D, Hadjieleftheriou M, Kollios G, Teng S. On trip planning queries in spatial databases. In *Proc. the 9th International Symposium of Advances in Spatial and Temporal Databases*, Aug. 2005, pp.273-290. DOI: 10.1007/11535331_16.

Jian Chen received his B.S. and M.S. degrees in software engineering from Harbin Institute of Technology, Harbin, in 2015 and 2021, respectively. He is a Ph.D. candidate in the School of Computer Science and Technology, Harbin Institute of Technology, Harbin. He is a student member of CCF. His research interests include query processing and spatio-temporal data management.

Kai-Qi Zhang received his B.S. and Ph.D. degrees in computer science from Harbin Institute of Technology, Harbin, in 2013 and 2020, respectively. He is currently a lecturer in the School of Computer Science and Technology, Harbin Institute of Technology, Harbin. His main research interests include query processing, massive data management and data-intensive computing.

Tian Ren received her B.S. and M.S. degrees in software engineering from Harbin Institute of Technology, Harbin, in 2016 and 2022, respectively. She is currently working toward her Ph.D. degree in the School of Computer Science and Technology, Harbin Institute of Technology, Harbin. Her research interests include query processing and data quality management.



Zhen-Qing Wu is an undergraduate student in the School of Computer Science and Technology, Harbin Institute of Technology, Harbin. His research interests include spatio-temporal data management and massive data management.

Hong Gao received her B.S. and M.S. degrees in computer science from Heilongjiang University, Harbin, in 1988 and 1991, respectively. And she received her Ph.D. degree in computer science from Harbin Institute of Technology, Harbin, in 2004. She is currently a professor with the School of Computer Science and Technology, Harbin Institute of Technology, Harbin. Her research interests include query processing, sensor networks, and massive data management.