Tracery and Epithet system with scheduling for contention aware kNN query

D.Sandeep[1], Shaik Munsif[2], G Narendra Kumar[3], C Mahesh[4], B Phaneendra Kumar[5]

1. Asst. Professor, Santhiram Engineering College, Dept. of CSE.

2,3,4,5 B.Tech Student, Santhiram Engineering College, Dept. of CSE.

Abstract-The smart transportation systems, e.G., DiDi, Uber, have served as essential travel equipment for customers, which foster lots of research for the location-based totally queries on street network. In particular, given a set O of items and a question point q on a avenue network, the okay Nearest Neighbour (kNN) query returns the k nearest objects in O with the shortest road community distance to q. In literature, maximum existing answers for kNN queries tend to reduce the question time, indexing storage, or throughput of the kNN queries even as overlooking the correctness of the queries resulting from question-query and update-query conflicts. In our work, we endorse a grid- primarily based framework on war-aware kNN queries on moving objects which ambitions to optimize machine throughput at the same time as ensuring question correctness. In particular, we first suggest green index systems and new question algorithms that significantly enhance the throughput. We similarly present novel scheduling algorithms that aim to avoid conflicts and enhance the system throughput. Moreover, we devise approximate answers that provide a controllable trade-off among the warfare of kNN queries and device throughput. Finally, we suggest a cost-primarily based dispatching strategy to assign the kNN effects to the corresponding queries. Extensive experiments on real-world facts show the effectiveness and efficiency of our proposed answers over alternatives.

Index Terms—Serialisable *k*NN Query, Conflict-aware Scheduling, Cost-based Dispatching, Constrained Minimum Bipartite Matching.

1INTRODUCTION: Given a fixed O of moving items and a query factor q on a avenue network, the okay Nearest Neighbor (kNN) question returns the ok nearest objects in O with the shortest avenue community distance to q. The kNN query on moving gadgets finds many essential real global applications. For instance, in current ride-hailing services like DiDi and Uber, a vacationer may also request a taxi at his/her modern location, and DiDi or Uber then needs to find numerous vehicles in its fleet which might be the closest to this location. Apart from taxi-hailing, there are numerous different applications applicable to kNN query on moving gadgets, such as road-facet servicing to accidents, police dispatch to the close by accident, emergency, and crime incident. Consider the software of police dispatch: when an incident arises, the police gadget will want to locate the nearby police patrols and assign one or some patrols based on the call for to the request as soon as possible. Once a patrol is assigned to a request, he/she cannot be the end result of other queries and will now not be available until he/she finishes the previously assigned task.

In literature, there exists a plethora of research works [12], [13], [15], [19], [24] that address the kNN queries on avenue community with recognize to shifting objects. However, most existing solutions especially consciousness on reducing the question processing time to get the kNN results or cutting down the indexing cost to support efficient update of transferring objects, whereas in exercise the system throughput, with a purpose to be laid low with both update frequency and question performance as proven in [13], is a far more precious metric whilst evaluating system performance. On one hand, current state- of-the-art solutions, e.G., TOAIN [13], as we can display in our experiments, still do now not offer a good trade-off some of the throughput, query performance, and the replace frequency. On the alternative hand, as we will illustrate shortly, maximum existing works fail to don't forget the correctness of the question results, on the way to detriment the user experience provided by using the applications using kNN queries. In general, the correctness issue may be attributed to two essential factors.

The first factor with a purpose to result in the correctness issue is the updates of moving objects. In practice, for most of the moving objects, the places of objects are up to date periodically, where each update can be appeared as a snapshot of gadgets' modern places. For instance, each car in DiDi reports its location to the gadget in every 2 four seconds [4]. Generally, let us denote the minimum interval time that the device can capture the movement of the gadgets as To. In literature, maximum existing

solutions construct index structures for those moving items and then replace the index structures each T seconds, e.G., each eight seconds as reported in [13]. In any such case, T > To, i.E., the index replace is much less common than the updates of moving gadgets, so that you can undoubtedly affect the correctness of the kNN query results. Figure 1(a) suggests the impact of T on kNN queries using the Beijing road network. We set k = 5, and use the kNN solution at the query time because the ground-truth. Then, for numerous values of update time period T for an index structure, ranging from 1 to sixteen seconds, we compare the average distance between gadgets returned by way of the kNN question using the index structure and the query location to the equivalent within the ground-truth. As shown in Figure 1(a), whilst T increases, the common distances for kNN queries additionally increases: when we go back the kNN question with T = 8, the common distance of kNN question would growth by more than 50% as compared to the ground-truth. In real-world programs like taxi- hailing services, the growth of the distances commonly indicates longer ready time, and increasing waiting time via 50% could be unfavourable to consumer experience. Therefore, it is crucial to set T near To for real-global applications.

The second aspect that affects the correctness of the kNN queries is particularly due to the conflicts between queries. In literature, maximum current works discard the viable conflicts in kNN queries and the question answer results have no quality guarantee at all. In practice, upon getting the kNN results, the gadget will assign one the objects to the query based on a certain dispatching strategy. However, none of the prevailing work do not forget the dispatching procedure, consequently overlooks the war among queries. For instance, given a kNN question request by using a person, the ok taxis returned by using the kNN queries may additionally all be occupied via concurrent kNN queries requested by other users. Because an object within the kNN end result (e.G., a taxi) for a query q1 can be assigned to a query person (e.G., a customer), every other query close to q2, whose end result overlaps with that of q1, might not be executed until the final touch of q1. Hence, the ready time of consumer of q2 can be delayed, resulting in a poor person experience. As shown in Figure 1(b), when 10,000 queries are processed with the machine with 50,000 objects, we can take a look at that the war fee increases sharply with the growth of ok, and becomes greater than 40% whilstokay = 40. Therefore, it is essential to recall kNN question conflicts in the course of the query processing. In the paper, we formally outline the conflicts of queries and the serializable kNN queries which assure that the kNN queries will be battle-aware. The serializable kNN has robust correctness assurance and in mostrealistic scenarios, it may not need that robust guarantees. We therefore further propose approximate serializable kNN queries which provide a controllable trade off among the correctness and throughput of kNN queries. This motivates us to endorse a framework that advantage high throughput of kNN queries processing whilst making certain their correctness.

2 FRAMEWORK

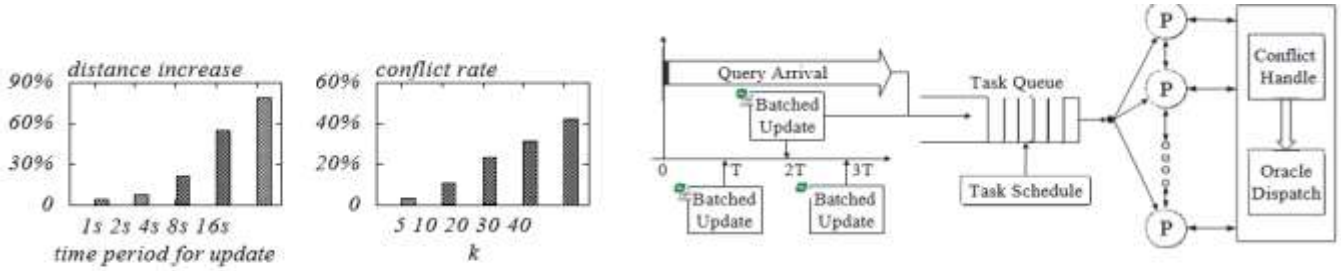
In this section, we introduce the general framework of our . We first gift the preliminary about some definitions and trouble statement. Then we introduce the machine model, observed by means of the throughput analysis.

Let G = (V, E) be a directed graph that represents a avenue network wherein V is the set of vertices and E is the set of road segments on this road community. For each edge e=(u,v) ϵ E, it is associated with a weight w(e), which represents the distance from u to v. Given two vertices s,t ϵ V, let P={e0,e1, … ei} be a path from s to t, then the distance of P is defined as $\sum_{i=0}^{l} w(ei)$.The shortest distance from s to t is defined as the minimum distance among all the paths from s to t, denoted as dist(s,t).On road networks, objects are often moving from on place to another. Typically it is difficult to monitor the location of objects continuously. Instead, for the objects that include GPSs the movement of these objects can be tracked periodically with every second.

Consider a set of M moving objects, where each object is located on road segment. Given an object o ϵ M located on a road segment e0=(u0,v0) such that the distance between o and v0 is w(o,v0). Given a query point q, which is located on a road segment eq=(uq,vq), the distance of vertex uq to q is w(uq,q). The distance from o to q is then:

Dist(q,o)= w(o,v0)+dist(u0,uq)+w(uq,q). (1)

Following previous works [13], [19], [24], we assume that the query locations and moving gadgets are all placed on vertices, ignoring the offset of the items to the vertices (resp. The vertices to question locations) on the street network. To explain, we are able to without



(a) Impact of T (b) Query conflicts (c) System Model

Fig. 1: Correctness issue in kNN queries and System Model

difficulty calculate the precise distance from the items to the question vicinity in keeping with Equation 1. The kNN query is defined as follows.

Definition 1 (kNN queries)**.***Given a query point q, a set M of moving objects on a road network GV, E), and an integer k ≤M , let dist(o, q) denote the distance of object o ϵ M to the query location q at the query time. The kNN query returns a set R ⊆M of k moving objects such that for all v ∈M \ R,dist(v, q) ≥ dist(u, q) for any u ∈R.*

After acquiring the kNN results for a specific query, a dispatching scheme will assign one of the results to the query. The dispatching scheme of our version will be introduced in Section 5. Here, we define such operation on moving object as the Oracle Dispatch.

Definition 2 (Oracle Dispatch)**.** *Given a query q and the corresponding kNN results R_q, the Oracle Dispatch assigns one moving object o ∈R_q to q, and marks o is an unavailable object*

Note that if a few queries share the equal objects, it would be feasible that one item is assigned to more than one queries at some point of dispatching. Next, we formally define a strict kNN-query Conflict as follow, and we will propose the approximate kNN question as a relaxation in Section 4.2.

Definition 3 (kNN-query Conflict)**.** *Given two kNN queriesq_a and q_b, let R_qa and R_qb be the kNN query answer for $q_a$and qb, respectivelyq_a conflicts with q_b if R_qa $R_q$bconflict ratio ρ between q_a and q_b is defined as follow*

$$\rho = \frac{|R_{q_a} \cdot R_{q_b}|}{K}$$

***Definition 4** (Update-query conflict)**.** Given an update request for a moving object o and a query q with R_q as the corresponding kNN results, there exists update-query conflict between q and o if o ∈ R_q.*

Notice that given a kNN question, there usually exists capability replace to one of the gadgets in kNN consequences by the oracle dispatch. Meanwhile, the replace to the kNN is generally unpredictable given that the replace may additionally be machine dependent, e.G., different systems may assign taxis to query users in unique ways, or may additionally contain human interactions, e.G., taxi drivers can also pick to accept or refuse the experience request. Strictly, if each of the moving object within the kNN outcomes is to be had for dispatch, that query will don't have any con- flict with others. Therefore, given a set of queries having no conflict with each other, they can be processed concurrently without any extra aid maintenance. Thus we define the serializable kNN queries as follow.

Definition 5 (Serializable kNNs)**.** *Given a set Q of kNN queries and U updates to the objects, we call these Q kNN queries and U updates are serializable if the objects' statuses (e.g., available / unavailable for the taxis) of their parallel processing results are equivalent to those of the sequential processing results.*

To assure the correctness of the kNN queries, we should keep away from the kNN query conflicts and question-replace conflicts. In the case whilst a conflict occurs to a query, we want to abort this query and re-run it, which is a waste of time. In Section 4, we can present a way to manage the conflicts with the intention to lessen the abort quotes and consequently enhance the throughput System Model.

2.2 System Model

Next, we present the system model to handle the kNN queries. Figure 1 (c) illustrates the system model to serve the correctness-aware kNN queries. It mainly includes two parts: the task arrival model and the task handling model.

Task Arrival Model. Following the previous work [13], we count on that the queries arrive at the gadget as a Poisson process. In the meantime, we assume that the system is periodically tracking the motion of the items, and the machine can seize the motion of the objects with each To seconds. For instance, To can be the GPS location sparkling periodicity. Then, for the moving objects, we count on the updates to items are available in batch each T seconds as proven in Figure 1 (c). Such an replace arrival model is referred to as the batched update version, which is likewise used in [13], and it's miles a realistic modelling for taxi-hailing services. Notice that T won't be similar to To since the update cost can be high and the device, therefore, techniques the replace in a more coarse-grained manner. We assume that the system start time is zero, and at every time l.T with $l \epsilon N^+$, the batched updates arrive, and the system immediately handles the updates before the query processing. Let Let $T_u$be the time for the batched updates. For the arrived queries in $[l \cdot T, (l + 1) \cdot T)$, it will then process the queries based on the updates at time $l \cdot T$.

Notice that if $T_u > T$, then, the system will have no throughput at all since it spends all time on the updates of the objects. Therefore, it is important to balance the update costs and the query processing time.

Task Handling Model. In this, we focus on processing the queries and updates with multiple processing units (as shown in figure 1(c)) to support concurrent execution (By setting the number of processing units as 1, it comes to the sequential execution). As mentioned earlier, updates are handled at the beginning if each time slot [l . T, (l+1) . T] with $1 \epsilon N^+$. For the queries , they are stored on a task queue, and a task schedule will assign the queries, to different processors. To process the queries, the system should finish the queries in response time Tr. This guarantees that for the earlier tasks that are not processed yet, it will have higher priority during the task scheduling. Since the queries are processed in parallel, there may exist kNN query conflicts and system includes a conflict handler to detect conflicts. . During the processing of a query, the conflict handler will detect whether there is a conflict between this query and any of the other queries being processed. Once a conflict is detected, this query will be aborted and the system will rerun the query. Notice that there will be no update-query conflict since the updates are processed before the queries in every time slot [l T, (l + 1) T) with l N^+. After obtaining the kNN results for each query, the oracle dispatch will assign each query an object based on certain strategies, which will be detailedly introduced in Section 5.

2.3 Throughput Analysis

In this subsection, we present the throughput calculation. Derived from TOAIN [13], the device throughput is laid low with both query processing time and the object update fee. Here, we only show the end result of the throughput analysis, in which the details may be located in [10]. Assume that the average query processing time for a kNN question is tq, the common cost for an update is tu, and the quantity of process units is p, then the device throughput, a.K.A., the range λ of queries that the system responds per unit time slot, satisfies the following

$$\lambda \leq \frac{T - t_u \cdot |M|/p}{T \cdot t_q/p} \qquad (2)$$

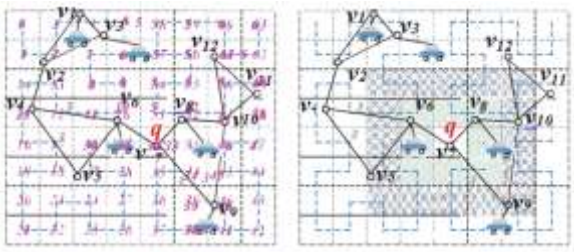
Alternatively, let λ be the query arrival rate (in number per second), t_q(resp. V_q) be the average (resp. variance) of query time using a single processing unit. Given the response time T_r, the average throughput (i.e., the largest query arrival rate) λ can be formulated by:

$$\lambda \leq \frac{2p^2(T_r - t_q)}{p^2 V_q + 2pT_r t_q + t_q^2(1-2p)} \qquad (3)$$

Combining Equations 2 and 3, we have:

$$\lambda \leq \min\left\{\frac{2p^2(T_r - t_q)}{p^2 V_q + 2pT_r t_q + t_q^2(1-2p)}, \frac{T - t_u \cdot |M|/p}{T \cdot t_q/p}\right\},$$

which is the throughput of our framework. Next, we present our framework to assist efficient query processing with very small update costs.



(a) Grid index (b) kNN query processing

Fig. 2: Example of GLAD index and kNN query processing

3.1 Grid Index for Updates

We first explain the details of our grid index for updates. Given a road network G, we partition the road network into $2^x 2^x$grids according to their latitude and longitude. For example, given the road network with 12 nodes in Figure 2, we divide the road network into 8 8 grids. Then, we make a one-to-one mapping from h [0, 2^{2x}1] to the grid cells by using the Hilbert curves. The main reason to maintain the Hilbert curves will be explained in Section 4. For now, we just assume that we have a one-to-one mapping.. Then for each $h \in [0, 2^{2x} - 1]$, we maintain a list L_h to denote the objects that fall into the corresponding grid cell mapping with Hilbert code h. Besides, for each moving object, we also maintain which grid cell the object currently falls into. Then, when the batched updates come at time $l \cdot T$, we update the list for all the objects and update the information of which grid the current object falls into. Denote the number of update requests at time l T as c_u. If c_u M , it will bewasteful to rebuild the list every T seconds.

As an alternative, we first replace the grid information for the object asking for an replace, i.E., we realize which mobile the object presently belongs to. Then, if the item moves from one grid to another, we simplest append the node into the listing similar to the new grid. This will result in redundant information. However, this will no longer affect the correctness of the query processing. To explain, given that for every item we hold an entry indicating the grid that it falls in, in the course of the exploration of grid (details in Section 3.2) we can clear out the redundant items through checking the corresponding entries. Also, to relieve the weight of such redundant data, we report the accumulated variety au of updates have been made to the grid index, i.E., each time there is an item shifting from one grid to another $a_u$is increased by one. And when $a_u = M$, we rebuild the index from scratch. By this strategy, we do not need to rebuild the grid index every T seconds and the complexity of each batched update can be bounded by $O(M)$, since each update of a moving object is at $O(1)$ amortized cost. To explain, the cost for each append is $O(1)$ and in total $O(a_u)$ cost, and to rebuild the index for $|M|$ objects, the cost is still $O(|M|)$. Since $a_u = |M|$, then the total update cost for $|M|$ objects

will be $O(|M|)$ which is optimal as by examining the location of each object, it already takes $O(|M|)$ cost.

3.2 Distance Index and kNN query processing

To answer the kNN queries, a naive answer is to do a Dijkstra search [8] from the query point and stops when okay nodes are retrieved. However, this answer will incur giant question overhead. Our primary remark for kNN question is that, in contrast to the objects which might be dynamically moving, the nodes on street networks aren't moving and we are able to pre-shop the distances between distinct nodes with some auxiliary index systems and reuse the index structure to answer the question. In the literature, there exists a plethora of index schemes for the distance queries. For instance, TOAIN integrates the Contraction Hierarchy [9] index structure to calculate the distances. In our solution, we integrate the contemporary distance labeling scheme H2H [16] for our kNN queries. We notice that the gap labeling may be modified to every other labeling scheme whichever is faster. Therefore, we treat the space labeling scheme as a black-field and use D(u, v) to indicate using the index to calculate the distance from u to v.

Given the index structure, a naive answer is to calculate the space from every item to the question point. However, this outcomes in needless computational expenses when you consider that some items some distance faraway from the query region are unlikely to be the kNN query answer. Motivated by means of this, we present a grid primarily based pruning method to reduce the query costs.

KNN Query Algorithm for framework

Input: Road network G, query point q, Distance scheme D, Grid index L
Output: kNN of q

1 Let h be the grid containing q, $H \leftarrow \emptyset$, $NH \leftarrow \{h\}$;
2 Let $C(q)$ be the candidate set of the kNN queries;
3 Let UB be the upper bound of the distance of the k-th nearest neighbor to q and initially set to $+\infty$;
4 Let $LB \leftarrow 0$ and $L_2(o, q)$ be the Euclidean distance between o and q;
5 **while** $UB > LB$ **do**
6 Let $L(NH)$ be the set of objects that fall in any grid in NH;
7 **for** $o \in L(NH)$ *with* $L_2(o, q) < UB$ **do**
8 Invoke $D(o, q)$ to calculate $dist(o, q)$;
9 Add o to $C(q)$;
10 $H \leftarrow H \cup NH$;
11 Update UB to the k-th shortest distance in $C(q)$;
12 Let NH be the set of grids that are neighbors of H but excluding the grids in H;
13 Let LB be the Euclidean distance from q to edge of NH.
14 Return top-k nodes in $C(q)$ with the shortest distance;

Query algorithm details. The pseudo-code of the kNN question algorithm is as proven in Algorithm 1. The main concept of the question set of rules is to use the Euclidean distance of the gadgets to avoid expanding the search space on the street network. We start from the grid of size 1 1, and then gradually increase the grid to 3 3, 5 5, till the kNN answers are found. Figure 2 (b) suggests an instance of the grid increasing. In the query set of rules, we maintain a hard and fast H to denote the set of grids which have been searched, first of all set to be . We in addition preserve a fixed NH to indicate the set of grids which can be associates of H, and initially set to be h (Algorithm 1 Line 1). We in addition maintain a hard and fast C(q) of candidates to the kNN query answer, first of all set to be . We then gradually increase the grids to be searched the use of pruning techniques as follows. At the beginning, we calculate the distances of all of the items in NH to the query location q using the distance labeling, and upload these objects to the candidate set C(q) (Algorithm 1 Lines 6-9). Then, we file the k-th shortest distance some of the points in the candidate set C(q) as the space top certain UB for the kNN question (Algorithm 1 Line 11). After that, we update H to encompass the grids in NH since the grids in NH were expanded. We further update NH to be the grids which can be pals of H except the grids in H ((Algorithm 1 Line 12)). Next, we report LB because the Euclidean distance from the threshold of NH to q. Notice that the Euclidean distance L2(o, q) from an object o to q is a lower certain of the shortest distance dist(o, q). Clearly, for all the nodes in the NH, the shortest distance must be no smaller than LB. In this way, if LB UB, all the nodes in NH need to not belong to kNN on the grounds that their distance to q is already no smaller than at Least k nodes. Therefore, we terminate the grid expansion (Algorithm 1 Line 5). Finally, we return the k nodes with the shortest distance in C(q) and go back them because the kNN query answer.

4. FRAMEWORK: CORRECTNESS AWARE MODELING

In this section, we introduce the concurrent version and analyze a way to provide accurate answer for the kNN queries. We first expect that the items will record its location within the system monitoring periodicity To. Then, we define T - correct kNN queries as follows

Definition 6 *(T -accurate kNN queries). Given a kNN query q requested at time tq, q is T -correct if the kNN are derived in step with the distances calculated with the latest index after the batch update within time slot [tq − T, tq).*

Obviously, the bigger T it's far, the less accurate the kNN queries are. Also, it's miles really worth to note that it is meaningless to update the index structure too frequently, i.E., putting index replace periodicity T < To, since we are able to gain at maximum To-correct kNN queries. To explain, the minimum batched update periodicity of the objects we can gain is To, and therefore, by using placing a small T , we might not have any replace in the periodicity and the accuracy is similar to the To-correct kNN question

answers. Obviously, To is typically not pretty large, e.G., may be as small as 0.05 second. If the update fee is excessive, the gadget will spend an excessive amount of time on the updates. For instance, as we are able to see in Section 6, the today's TOAIN [13] nonetheless incurs very high update value to gain low query processing time. In contrast, our grid-based indexing scheme bears very small replace cost and achieves a good stability among the replace value and query efficiency. In particular, whilst we set the index replace periodicity T = 0.25 seconds, our nevertheless achieves very excessive throughput and is 4x faster than TOAIN. By allowing the gadget to gain the pleasant possible accuracy with out sacrificing the throughput, we will offer more accurate kNN question answers and offer better user stories for the applications using the kNN queries.

4.1 Conflict-Aware Scheduling

To gain the fine feasible throughput, in our model we are able to use more than one processing gadgets to deal with queries con- currently. However, as we stated in Section 2, when more than one kNN queries are processed, we need to guarantee the kNN queries are processed in a serializable manner (Ref. Definition 5). To obtain this, when kNN queries are processed in parallel, we want to assure that there is no struggle on the kNN queries. Suppose there may be no scheduling to the coming queries, the queries may incur too many conflicts and therefore a high abort rate, degrading the sys- tem throughput. Therefore, we present a novel scheduling scheme for the concurrent kNN question processing. A chal- lenge for scheduling is that the scheduling algorithm must be light-weight while effective since (i) if the scheduling is too complex it can dominate the cost and bring no development to the throughput even supposing the abort price is reduced, (ii) if the scheduling is very simple, e.G., using Random assignment, the abort rate can be nevertheless very high, and does not help improve the throughput. Facing this dilemma, we present a novel scheduling algorithm based at the Hilbert curves. The main idea is that for the query points with near Hilbert curve numbers, they're highly probably to battle with every other, even as if the difference of Hilbert curve numbers are massive for the query points, it's miles highly possibly that these queries have no conflicts. Therefore, we schedule the queries which might be less likely to have conflicts concurrently, ensuing in smaller abort rate. We endorse two types of scheduling approaches: Hilbert workload-balanced scheduling and Hilbert distance-primarily based scheduling. We omit the information for the interest of space, and they are illustrated in our convention manuscript [10].

4.2 Approximate kNN queries

As we mentioned in Section 1, some applications there can also not want strict battle of kNN queries. However, unlike the present paintings that handles kNN queries overlooking the conflicts and correctness, we propose an approximate version of the serializable kNN queries. The approximation gives a controllable trade-off among the question correct- ness and gadget throughput.

Definition 7 *(ρ-approximate serializable kNN). Given a hard and fast Q of kNN queries, these Q kNN queries are ρ-approximate serializable if the queries are processed in parallel whilst allowing that the kNN answer of a question q bears at maximum ρ k shared gadgets with different concurrent queries.*

As we can see, the ρ-approximate serializable-kNN allows some of the kNN effects to be shared through different kNN queries. But the ratio of the shared objects should be lim- ited to a positive degree, i.E., ρ. Fortunately, in exercise the approximation tuning requires minor amendment to our war handler. In particular, in serializable kNN question processing, when an item is lower back as the answer through a kNN query, then the current query aborts and re-runs the query from scratch. In the ρ-approximate serializable kNN question processing, it counts the number c of gadgets which are shared by means of different kNN queries and aborts except the number c is larger than ρ k. As we will see in our experiment, the approximate definition additionally helps reduce the conflicts and the abort rate. We will also take a look at the impact of ρ to the query accuracy and question throughput.

5 FRAMEWORK: COST-BASED DISPATCH STRATEGY

In this section, we present the cost-primarily based dispatch strategies to assign an object to a given query. In general, after several moving gadgets are retrieved with the query algorithm, the device will choose one among the gadgets based on a certain approach because the response to a given query. Then this item might be marked as an occupied one, and will be available after it finishes the request task, e.G., a taxi arrives the destination of a journey. A naive approach is to randomly select up one item from the question results. However, such approach fails to recall the tour distance optimization, which targets to minimize the total driving distance of moving objects, thus, reduces the average waiting time of the request tasks. Hence, we study the the cost-based dispatch strategywhich is also known as the assignment problem.

5.1 Constrained Minimum Bipartite Matching

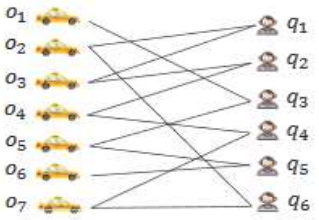
Specifically, given a set of moving gadgets and a set of requests on the road network, we strive to discover a matching between the moving items and requests such that the total travel prices of the matched pairs are minimal. In the ultimate decade, a similar problem has attracted much interest from many researchers, named the Minimum Bipartite Matching (MBM) problem, which aims to find a perfect matching with minimal general distance among the matched pairs of a service set and a user set in a dimensional space [23]. In our work, a bipartite graph can be fashioned with the set of moving gadgets and the query points wherein the weighted edges represent the road

network distances between them. However, not like the conventional MBM hassle wherein for every pair of vertices from two aspects there exists an edge connecting them, in our work, we can most effective locate edges from each query to all its kNN results. Formally, we define the kNN bipartite graph as follow.

Definition 10 (Competitive Ratio)**.** *The competitive ratio of an matching algorithm A for the CMBM problem is as follow*

$$CR(A, kBG(M \cup Q, D)) = \max \frac{Cost(H)}{Cost(OPT)} \qquad (4)$$

Where kBG(M∪Q,D)is the k NN bipartite graph formed ***via*** *a set M of moving* ***objects*** *and a set Q of query* ***points*** *that arrivein an arbitrary order, His the marching return* ***by*** *algorithm A, and OPT refers to the optimal solution with offline globalinformation.*



(a) 2NN Bipartite Graph

	o1	o2	o3	o4	o5	o6	o7
q1	-	5	4	-	-	-	-
q2	-	-	1	6	-	-	-
q3	4	-	-	-	3	-	-
q4	-	-	-	1	-	-	7
q5	-	-	-	-	2	6	-
q6	-	6	-	-	-	-	2

(b) Matrix for Matching

Fig. 3: Dispatch Examples

To explain, the aggressive ratio measures the value ratio among the result of an online algorithm and the optimal result received by using an MBM set of rules in which the offline data is given in advance. Note that during our frame- work, there could be a batched update each time interval T . Thus the aggressive ratio is calculated for the set Q of queries arriving inside the equal time interval, in which the places of the moving gadgets do not change throughout that time interval.

5.2 Greedy Partial Matching Algorithm

A naive algorithm for our CMBM problem is the Greedy Nearest Neighbor algorithm, which matched every new arrival question to its currently nearest available moving object. For example, given the 2NN bipartite graph proven in Figure 3 (a), the matching result obtained by the Greedy Nearest Neighbour algorithm is{(q1,o3),(q2,o4),(q3,o5),(q4,o7),(q5,o6),(q6,o2)}. The total dispatching cost is at 34, which is much larger than the one for the optimal solution, and the competitive ratio is2.3. In fact, the worse case competitive ratio for the Greedy Nearest Neighbour algorithm is huge, which is proven to be 2c 1, where c = H is the maximum cardinality of the matching [11]. However, as discussed in [20], the idea of greedy is not always the worst, and in practice works well since the worst case will only appear with a very small probability. Note that, in our concurrent model, there will be a batch of queries being processed at the same time. Thus, we can employ the greedy idea to perform the matching based on the batched queries and their kNN results. Consequently, we propose a greedy partial matching algorithm which finds the optimal solutions for periodically batched queries.

Note that during our CMBM problem, the cardinality constraint restricts the wide variety of queries to be served. If the queries processed simultaneously percentage the equal kNNs, there might arise the case that a number of the queries fail to be assigned as their kNNs have been matched to other queries. Fortunately, as our model can provide guarantee for the conflict-aware kNN queries, the maximum cardinality con- straint can be ensured accordingly. Next, we prove that with the correctness guarantee,i.e.0≤ρ <1, we can achieve the maximum cardinality to be the number of queries, i.e.,|H|=|Q|. That is, for each query we guarantee that there will be a moving object assigned to it.

Next, we introduce the greedy partial matching algo- rithm to clear up the constraint minimum bipartite matching in our model. The primary concept is that we first carry out a batch of queries concurrently, the ones arrive the machine in the equal time slot (l T, (l + 1) T).Then, we generate the kNN bipartite graph primarily based on the processed queries and the corresponding results, followed by way of the premiere matching computation making use of the Hungarical Algorithm [11] to do the matching of moving objects for the batched queries. Note that, an object may be marked as unavailable once it's miles assigned to a question. Be aware that, the wide variety of batched queries does not need to be the same as the number of the processing units. In fact, it could be larger. We notate the wide variety of batched queries for greedy partial matching as ϕ. To explain, we maintain a listing for the kNN outcomes of the queries which have been processed. And after very ϕ number of queries get their results, we generate the kBG for the ones queries and objects. Then, we calculate the most useful matching answer for those batched queries primarily based on the Hungarical Algorithm. Note that, most effective the gadgets in the kNN consequences of the batched queries might be involved within the calculation. Finally, we update the supply of the matched items, and we iteratively carry out the question and dispatch in the batched manner. For example, think that we set varphi = 2 for the queries in Figure 3. Then the matching result comes to be (q1, o2), (q2, o3), (q3, o5), (q4, o4), (q5, o6), (q6, o7). And the dispatching price and competitive ratio are 18 and 1.2 respectively. The grasping partial algorithm receives extra global information than the greedy nearest neighbor set of rules, and therefore, can reap better performance in phrases of competitive ratio. However, the computation fee is a great deal higher. A particular discussion will be provided in our experimental study, and we compare the grasping partial set of rules with two other solutions: random assign and greedy nearest neighbor algorithms.

TABLE 1: Parameter Settings (default values in **boldface**)

Parameter	Description	Values
k	Input parameter of kNN	5,**10**,20,30,40
T_r	Response time (ms)	0.1,**0.4**,1.6,6.4
T	Update periodicity (s)	0.25, **1**, 4, 16
$\|M\|$	Number of moving objects	5,**10**,15,20 (×10000)
ρ	Approximation ratio	**0**, 0.2, 0.5, 1

TABLE 2: Road Networks and Preprocessing Cost of GLAD

Symbol	Network	#Edges	#Nodes	Space	Time
NY	New York	733,846	264,346	400MB	85.99s
BJ	Beijing	775,176	296,710	682MB	190.17s
BJ1	BJ Sub-network	33,608	12,832	6MB	1.84s
BJ2		175,180	66,546	81MB	25.39s
BJ3		774,660	296,381	682MB	189.85s

6 EXPERIMENTAL STUDY

In this section, we experimentally evaluate our framework against the state-of-the-art on real-world road networks. All methods are implemented in C++ and com- piled with full optimizations. All experiments are conducted on a Linux machine with two CPUs, each with 10-cores clocked at 2.90GHz and 192GB memory. We employ the Cilk Plus scheme for concurrent query processing. We repeat each set of experiments 10 times and report the average results.

6.1 Experimental Settings Dataset and Query set. We conduct our experiments on two real street networks: the Beijing (BJ) avenue community and the New York (NY) street community, which are also used in [13]. The BJ network includes 296K nodes and 775K edges; the NY dataset consist of 264K nodes and 733K edges. For NY, we attain a real dataset from NYC Open Data [2], which has 18K taxi trajectories. We map the starting point of each trajectory to the nearest vertex on the road community. Then, the query locations are 1/2 generated from a random starting points of the trajectories and half generated from a random vertex on the street community. For BJ dataset, we acquire a fixed of Point of Interests (POIs) from the Open Street Map [3]. Similarly, we map each POI to the nearest vertex on BJ street community. Then, query places are half generated from random POIs, and half of generated from random vertices on road network. For each scenarios, the starting function of moving objects are generated uniformly from the road network. We observe that the identical setting is used in [13]. In addition, to assess the impact of various sizes of road networks, we extract three sub-networks from the BJ dataset following the same approach in [13]. Specifically, we steadily expand the street community from the center of the Beijing city to get positive number of nodes and the corresponding edges. These sub-networks are labeled as BJ1, BJ2, and BJ3 with growing size of nodes and edges. Table 2 suggests the particular sizes for extraordinary datasets.

Preprocessing Cost. Next, we file the preprocessing cost . Note that the index includes a grid index for the moving gadgets and an H2H index for the street network. The H2H index is a hierarchical 2-hop labeling index [16], which debts for the essential space consumption of our index. Table 2 indicates the information of the index length and preprocessing time for different avenue networks, in which Space (resp. Time) shows the index length (resp. Preprocessing time). From the results, we are able to see that index length is not any greater than 700MB, which can be easily equipped into the primary reminiscence of (almost) all current commodity servers. Besides, the preprocessing time to build the index is likewise very small.

6.2 Correctness Study

Methods and parameter settings. We compare our approach in opposition to the state-of-the-art TOAIN to evaluate the through- put. Since TOAIN is carried out with single-thread, when comparing towards TOAIN, we repair p = 1, i.E., the usage of a single thread, for . Since TOAIN can song the replace price and query performance via placing their parameters, we use three versions of TOAIN, denoted as TOAIN-1, TOAIN- 2, and TOAIN-three 3, whose putting gives high question effi- ciency however also high update value, medium query performance with medium replace value, and low question efficiency with low replace fee, respectively. In other cases, we run our with complete parallelization the usage of forty threads. For the grid size, we set the scale to be 100m in all of our experiment. For our , the parameter settings are proven in Table 1. When in any other case explicitly specified, the default settings are proven in boldface.

In this section, we first experimentally evaluate the correctness of our solution towards the ultra-modern TOAIN. As referred to in Section 1, there are two factors that will have an effect on the correctness. The first issue is the update time T , which influences the correctness of the query answer. Actually, the current strategies can reduce the effect of the problem by making the update time to a small value. Therefore, we will examine the impact of T to and TOAIN in the subsequent set of experiments while comparing the performance. The second aspect, in particular, conflicts amongst queries, that impacts the correctness of kNN queries, are unnoticed through present solutions.

To simulate the conflicts, we expect that a batch of kNN queries arrive and for each query we randomly assign one of the kNN outcomes to the query. If a question conflicts with the others, this type of question violates the serializability. We report the ratio of the quantity of queries that do not violate kNN serializability over the whole processed queries as the accuracy. Figure 5 suggests the accuracy of the kNN queries with the alternate of question sizes on the two datasets. As we can see, while the batched question size increases from 20, 000 to 80, 000, the accuracy of TOAIN will degrade to much less than 50% considering they do not consider the conflicts. In contrast, our continuously affords 100% acuracy because it handles conflicts at some stage in the question processing. Since the

battle of the queries will result in query fails and want to rerun the query and brings extra ready time. While such query fails reach 50%, it will bring about horrific consumer experience. By coping with the conflicts, avoids such correctness issues and consequently is a more preferred choice.

6.3 Throughput study

In this set of experiments, we evaluate the impact of dif- ferent parameters to the query throughput and TOAIN. We first have a look at the impact of the update periodicity T to the query throughput. As we can see from Figure 4, there is little impact of the update periodicity to the throughput of our since uses a very powerful index scheme for updates. In contrast, the throughput of TOAIN is severely affected by T . For instance, when T is no large than 4, TOAIN-1 (Ref. Section 6.1 for the details) incurs zero throughput since it bears high replace price, and therefore spend all of the time updating the index structure, leaving no time to method queries. For TOAIN-2 and TOAIN-3, even though their update fees reduce, the throughputs also degrade due to the fact the reduced replace fees come at scarifying the question performance. When T = 0.25, the throughput of our is 4x (resp. 20x) better than TOAIN-2 (resp. TOAIN-3). Even beneath other settings, our still provides better throughput. However, due to the fact a smaller T normally shows a higher accuracy of the kNN answers, our is extra preferred desire because it significantly outperforms the modern-day while providing correct kNN consequences whilst the gadgets are moving.

Clearly, the query time and update time influences the throughput of each and TOAIN. Therefore, we fur- ther investigate the kNN common query time and update time. For the question time, we randomly generate 100,000 queries and file the common. For the update, we calculate the av- erage update time of every object the use of 10 batched updates. As we will see from Figure 6, while has the identical update value as TOAIN (with TOAIN-three), the question time of is lots smaller than that of TOAIN. When the query time is similar, TOAIN bears an awful lot higher udpate cost than. This indicates our framework gains a miles higher trade-off some of the question efficiency, update cost, and throughput.

Apart from T , we in addition look into the impact of the reaction time, the enter parameter ok, and the range of moving gadgets as shown in Figures 4 (b)-(d). The foremost observations are that: (i) to offer comparable reaction time, our always provides higher throughput than TOAIN in all settings, and the throughput of isn't always extensively suffering from small reaction time; (ii) with the boom of okay, the kNN question overhead will increase, however nevertheless appreciably outperforms TOAIN while ok reaches 40; (iii) When the range of items will increase as much as 50, 000, still outperforms TOAIN in all settings.

6.4 Scheduling and Concurrent Query Processing

In this set of experiments, We first show the throughput with concurrent query processing with 40 cores in various settings. The results are shown in Figures 7(a) - (d). The throughput of with the concurrent query processing is significantly higher than the single-thread version, which is expected since it reduces the average query time significantly by exploring multi-cores. Besides, with the distance-based scheduling method, the throughput further improves over the random scheduling based method.

In addition, we evaluate the effectiveness of the pro- posed approximate kNN queries. We change the parameterρ (Ref. Sections 2 and 4.2 for the definition.) and see the impact to the conflicts. Our results in [10] show that by allowing approximation, we can significantly reduce the abort rate, and with the increase of ρ, the number of query aborts significantly reduce. Our distance-based scheduling method further helps reduce the abort rate. Further, we also evaluate the scalability performance of GLAD with vary size of BJ road networks. We draw similar conclusion as those mentioned in the previous experiments that GLAD can achieve high system throughput. Besides, it is highly adaptable to different size of road networks. For the interest of space, we omit the details of these two sets of experiments.

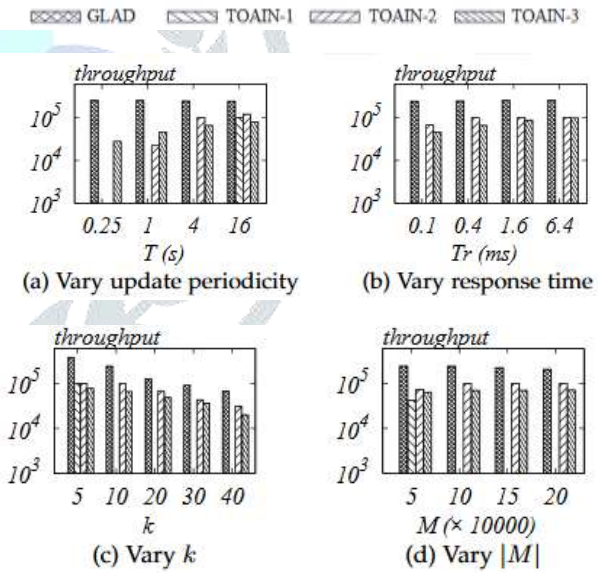


Fig. 4: Throughput comparison with TOAIN on NY dataset

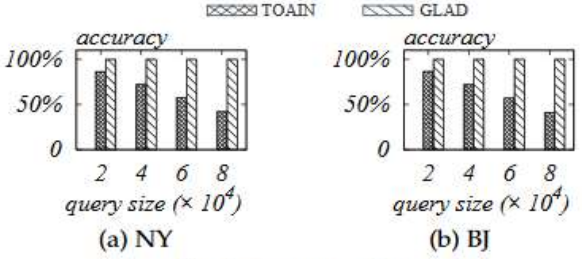
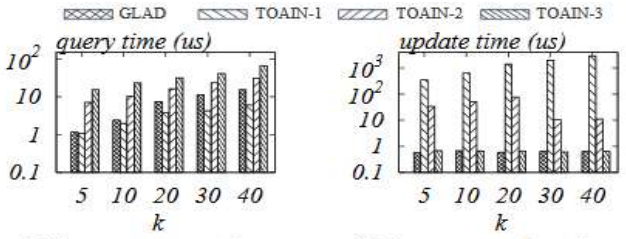


Fig. 5: Accuracy Comparison



(a) Average query time (b) Average update time

Fig. 6: Query time and update time on NY dataset

6.5 Dispatching Solutions

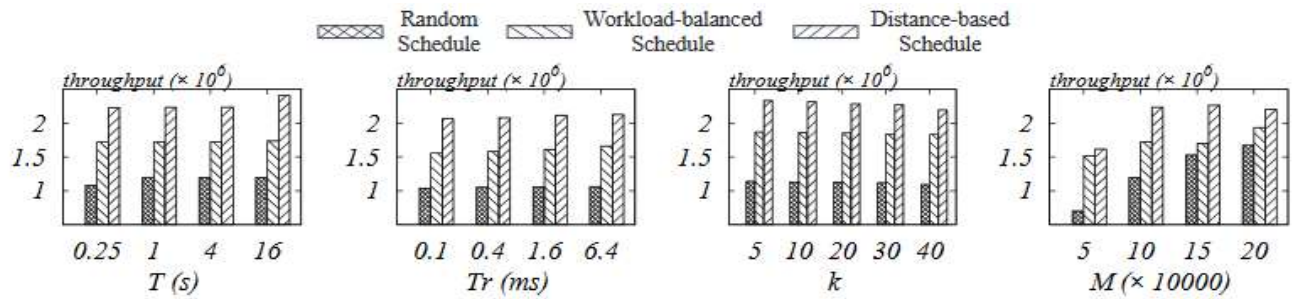
In this section, we report the experiment for three types of dispatching algorithms: Greedy Partial (GP), Greedy Nearest Neighbor (GNN), and Random Assign. We evaluate the aggressive ratio and the average computational price of three algorithms, in which the whole offline top-rated dispatching value for a given set of queries is calculated by way of the Hungarical Algorithm. Table 3 suggests the comparison of these 3 algorithms. Here, the overall quantity of queries being processed is 20000, and as for greedy partial algo- rithm, the parameter ϕ is about to be a hundred. From the result in Table 3, we can see that, the GP set of rules beats the other algorithms in phrases of aggressive ratio, at the same time as its computational fee is huge. Besides, increasing the range of ok brings little variation on the aggressive ratio, but results in dramatic boost of time consumption. We observe from this set of experiments that both grasping algorithms can obtain good competitive ratio, that is inline with the analysis in [20]. Next, we study the competitive ratio for the GP algorithm varying two parameters. We increase the size of queries to do matching, even as the batched size is ready as ϕ = a hundred. As we can see from Table 4, the competitive ratio growth along with the number of queries. This is because once the gadgets are assigned to the previous queries, the queries acting later and closer might ought to discover further gadgets. The more queries, the further objects might be assigned, as a way to lead to the increase of the total price. As for the parameter ϕ, Table 5 suggests that, when we boom the batched size, the aggressive ratio will decrease, that is reasonable. However, the computational fee will increase with larger length of batched queries to perform the most appropriate matching. With the support of the concurrent question processing, our model can intrinsically get more global data in the time to finish simplest one query. Thus the Greedy Partial set of rules would work well with a right the dimensions of batched queries. Consequently, from the results, we have a look at that ok = 5 or k = 10, and ϕ = one hundred might be the desired parameter placing for the dispatching solution, which can offer a better aggressive ratio sacrificing an appropriate computational cost.

7 RELATED WORK

7.1 kNN Queries

A most sincere answer is the Dijkstra algorithm [8], that's the maximum famous set of rules for the single source shortest distance computation algorithm in a graph. The Dijkstra algorithm needs no additional indexing structure, but the on-the-fly query processing cost is still too high. ROAD [7] extends the easy Dijkstra set of rules through aug- menting a hierarchical shape that walls a graph into numerous subgraphs and forms a bigger graph hierarchically. ROAD can speed up the Dijkstra with the aid of skipping the enlargement of subgraphs which contain no moving items. However, it'll degenerate to the Dijkstra algorithm whilst the moving objects are calmly distributed, accordingly performs poorly. G-tree [24] adopts a similar graph partition and hierarchical structure as ROAD at the kNN query, associated with a border set and corresponding distance matrix on every subgraph. G-tree solutions kNN query from a top-down way and calculates the space among vertices primarily based on the distance matrix that stores the distances between boarders and vertices. G-tree is more green than ROAD in query processing however involves extra sizable update price. V-tree [19] improves G-tree with the aid of appending local nearest active vertex desk on each subgraph that only vertices containing moving objects are stored, such that it can keep away from duplicated computation for unnecessary vertices.

A most latest work on kNN question on street network is the TOAIN [13], which is based on the Contraction Hierarchy (CH) [9]. It builds a SCOB index on the shortcut graph of the original graph that pre-computes the okay nearest downhill objects w.R.T. Nodes at the shortcut graph. TOAIN performs the kNN query with a Dijkstra search on the shortcut graph from the question vertex and receives the outcomes from the precomputed nearest downhill gadgets. TOAIN differs from the



(a) Vary update periodicity (b) Vary response time (c) Vary k (d) Vary $|M|$

Fig. 7: Throughput with 40 cores

TABLE 3: Comparison of different algorithms

k	ompetitive Ratio			omputational Cost (us)		
	GP	NN	andom	GP	GNN	andom
	.852	018	897	2.376	0.006	018
)	.846	009	810	1.037	0.008	020
)	.845	008	0.902	78.055	0.008	024
)	.845	007	.317	36.214	0.010	025
)	.845	007	5.149	247.741	0.012	026

TABLE 4: Query size

ompetitive Ratio	
Query	GP
000	1.233
0000	1.236
5000	2.536
0000	3.846
5000	4.312

TABLE 5: varyφ

ompetitive Ratio	
ϕ	GP
100	846
200	837
400	825
800	795
1600	776

Preceding solutions not simplest within the index shape and question method however also in the consideration of the throughput optimization. However, the correctness problems are unnoticed in TOAIN (in addition to any preceding answer). In our work, we suggest a greater sophisticated device version that takes the correctness problem into account. Meanwhile, our GLAD is efficient in both question and update which can extensively accelerate the system throughput

7.2 Shortest Path Queries

Road community queries have been studied for decades, specially the shortest direction question for single pair, which find the shortest direction for two vertices [21], [22], [24]. Dijkstra set of rules is the most straightforward manner however renders large price in exploring the graph. Bidirectional-Dijkstra [17] cuts down the price by means of invoking two Dijkstra search from the supply and destination simultaneously. Contraction Hierarchies (CH) [9] pre-computes the distances among numerous vertices based on a total order, and add shortcuts at the graph in order that the Bidirectional-Dijkstra handiest go to node in ascending order of the nodes, reducing question time for shortest direction and distance queries . TRN [6] is an index Based approach that imposes a grid on the road network, which pre-computes the shortest paths from within every grid cell to a hard and fast of vertices which might be deemed critical for the corresponding cell. Spatially Induced Linkage Cognizance (SILC) [18] pre-computes the all-pairs shortest paths in the road network, and stores the shortest paths in a concise form for green question processing. Apart from the studies work mentioned above, many other research work were proposed for shortest queries [21], [22], [25] assuming that the graph can't be geared up into the memory or there exist extra constrains.

7.3 Online Minimum Bipartite Matching

The online minimal bipartite matching trouble is ap- plied to a wide variety of applications, e.G., challenge assignment in crowd sourcing and taxi dispatching, and has been drastically studied. Some of the famous algorithms for online minimal bipartite matching encompass Greedy [11], Permutation [11], HST-Greedy [14], and HST-Reassignment [5]. The Greedy matches each new arrival query to its modern nearest neighbor, which is efficient but leads to big competitive ratio. Permutation applies Hungarical Algorithm to compute the correct matching to locate the unmatched provider for the brand new arrival question. Both HST- Greedy and HST-Reassignment are randomized algorithms, where HST refers to hierarchically separated tree built on carrier providers. HST-Greedy reveals the nearest issuer primarily based on tree metric to a question. HST-Reassignment develops a reassignment technique to enhance the local optimal trap as a result of greedy. Both HST-Greedy and HST-Reassignment can obtain a good aggressive ratio while want extra index strictures and preprocessing cost.

8. CONCLUSIONS

In this paper, we present a comprehensive study on correctness-aware kNN queries, which aims at optimizing the system throughput while guaranteeing the correct-ness of the kNN queries on moving objects. We develop an efficient framework GLAD which provide high sys-tem throughput while guaranteeing the query correctness. The GLAD framework includes effective index structures and novel scheduling algorithms to improve the system throughput. We further propose approximate solutions that provide a controllable trade-off between correctness and throughput of kNN queries. Extensive experiments on real- world road network show that our solutions are several times faster than the state-of-the-art. Finally, we propose an effective cost-based dispatching strategy for our model to minimize the overall travelling cost from the moving objects to the query locations.

REFERENCES

[1] https://news.thomasnet.com/fullstory/gps-receiver-provides- 20-hz-update-rate-587490.

[2] https://opendata.cityofnewyork.us/.

[3] https://www.openstreetmap.org/.

[4] Didichuxing. https://gaia.didichuxing.com/.

[5] N. Bansal, N. Buchbinder, A. Gupta, and J. S. Naor. A randomized o (log 2 k)-competitive algorithm for metric bipartite matching. *Algorithmica*, 68(2):390–403, 2014.

[6] H. Bast, S. Funke, and D. Matijevic´. Transit: ultrafast shortest- path queries with linear-time preprocessing. In *9th DIMACS Implementation Challenge – Shortest Path*, 2006.

[7] U. Demiryurek, F. Banaei-Kashani, and C. Shahabi. Efficient continuous nearest neighbor query in spatial networks using euclidean restriction. In *International Symposium on Spatial and Temporal Databases*, pages 25–43. Springer, 2009.

[8] E. W. Dijkstra. A note on two problems in connexion with graphs.

Numerische mathematik, 1(1):269–271, 1959.

[9] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contrac- tion hierarchies: Faster and simpler hierarchical routing in road networks. In *International Workshop on Experimental and Efficient Algorithms*, pages 319–333. Springer, 2008.

[10] D. He, S. Wang, X. Zhou, and R. Cheng. An efficient framework for correctness-aware knn queries on road networks. In *Data Engineering (ICDE), 2019 IEEE 35th International Conference on*, pages 1298–1309. IEEE, 2019.

[11] B. Kalyanasundaram and K. Pruhs. Online weighted matching.

Journal of Algorithms, 14(3):478–488, 1993.

[12] M. Kolahdouzan and C. Shahabi. Voronoi-based k nearest neigh- bor search for spatial network databases. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 840–851. VLDB Endowment, 2004.

[13] S. Luo, B. Kao, G. Li, J. Hu, R. Cheng, and Y. Zheng. Toain: a throughput optimizing adaptive index for answering dynamic k nn queries on road networks. *Proceedings of the VLDB Endowment*, 11(5):594–606, 2018.

[14] A. Meyerson, A. Nanavati, and L. Poplawski. Randomized online algorithms for minimum metric bipartite matching. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algo- rithm*, pages 954–959. Society for Industrial and Applied Mathe- matics, 2006.

[15] S. Nutanong, R. Zhang, E. Tanin, and L. Kulik. The v*-diagram: a query-dependent approach to moving knn queries. *Proceedings of the VLDB Endowment*, 1(1):1095–1106, 2008.

[16] D. Ouyang, L. Qin, L. Chang, X. Lin, Y. Zhang, and Q. Zhu. When hierarchy meets 2-hop-labeling: Efficient shortest distance queries on road networks. In *SIGMOD*, pages 709–724, 2018.

[17] I. Pohl. Bidirectional and heuristic search in path problems. Technical report, 1969.

[18] H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 43–54. ACM, 2008.

[19] B. Shen, Y. Zhao, G. Li, W. Zheng, Y. Qin, B. Yuan, and Y. Rao. V-tree: Efficient knn search on moving objects with road-network constraints. In *Data Engineering (ICDE), 2017 IEEE 33rd Interna- tional Conference on*, pages 609–620. IEEE, 2017.

[20] Y. Tong, J. She, B. Ding, L. Chen, T. Wo, and K. Xu. Online minimum matching in real-time spatial data: experiments and analysis. *Proceedings of the VLDB Endowment*, 9(12):1053–1064, 2016.

[21] S. Wang, W. Lin, Y. Yang, X. Xiao, and S. Zhou. Efficient route planning on public transportation networks: A labelling approach. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 967–982. ACM, 2015.

[22] S. Wang, X. Xiao, Y. Yang, and W. Lin. Effective indexing for approximate constrained shortest path queries on large road net- works. *Proceedings of the VLDB Endowment*, 10(2):61–72, 2016.

[23] R. C.-W. Wong, Y. Tao, A. W.-C. Fu, and X. Xiao. On efficient spatial matching. In *Proceedings of the 33rd international conference on Very large data bases*, pages 579–590. VLDB Endowment, 2007.

[24] 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. *IEEE Transactions on Knowledge and Data Engineering*, 27(8):2175– 2189, 2015.

[25] A. D. Zhu, X. Xiao, S. Wang, and W. Lin. Efficient single-source shortest path and distance queries on large graphs. In SIGKDD, pages 998–1006, 2013.