The VLDB Journal (2025) 34:61
https://doi.org/10.1007/s00778-025-00940-w

REGULAR PAPER

Route optimization with collective spatial keywords: A skyline-based approach

Jiajia Li[1] · Qiulin An[1] · Yang Song[1] · Xing Xiong[1] · Lei Li[3,4] · Fengmei Jin[2] · Xiaofang Zhou[4]

Received: 30 December 2024 / Revised: 6 June 2025 / Accepted: 11 August 2025 / Published online: 25 August 2025

Abstract

With the development of location-based services, smart cities, and intelligent transportation, route planning has evolved beyond shortest path finding to satisfy user's flexible travel purposes through the Optimal Routes with Collective Spatial Keywords (ORCSK) routing. Because different Points of Interest (POIs) contain different sets of keywords, the user usually needs to visit multiple POIs to fulfill all needs. Moreover, the POIs' stop hardness (time and cost) also influences user experience, but it was ignored by the existing solutions. Therefore, this work proposes to extend the ORCSK problem into Skyline Optimal Routes with Collective Spatial Keyword (Sky-ORCSK) by considering both distance and stop hardness. Specifically, we first propose the IG-Sky algorithm from the spatial keyword search perspective by extending the IG-Tree. Then we propose the DA-Sky algorithm from the path enumeration perspective by extending our previous DA-CSK. Furthermore, five optimization strategies are proposed to improve efficiency by pruning the search space. Extensive experimental evaluations on real-world datasets demonstrate the algorithms' efficacy and reliability, marking a significant step forward in refined route planning for modern urban environments.

Keywords Skyline · Spatial Keyword · Shortest Path · Optimization Strategies

1 Introduction

With the proliferation of GPS-enabled smart devices, location-based social networks, and spatial databases [14, 15], the capabilities of routing on road networks have expanded significantly. Today, routing can cater to a variety of user-specified needs beyond merely identifying the shortest [56–59] or fastest paths [29–32] like multiple travel model [33, 37], multiple optimization dimensions [38–40, 46, 51], multiple diversified results [9, 35, 42, 53, 61], and keyword-aware multi-stop routing [7, 16, 24–27, 41, 49]. Among them, keyword-aware routing is most practical one for users to plan their personalized trips because it finds the shortest route that can satisfy a list of user's needs (keywords) by stopping at multiple Points-of-Interest POIs (each satisfying a subset of keywords).

However, the current problem setting only takes the total distance as the optimization goal but ignores the hardness of stops, which also influences user experiences. For instance, while large shopping malls may satisfy more user needs, parking can sometimes be very difficult, either very expensive or hard to find a spot. Take Figure 1 as an example. Consider a user returning home from work who needs to

✉ Lei Li
thorli@ust.hk

Jiajia Li
lijiajia@sau.edu.cn

Qiulin An
anqiulin0730@163.com

Yang Song
songyang2@stu.sau.edu.cn

Xing Xiong
xingx1015@163.com

Fengmei Jin
fengmei.jin@polyu.edu.hk

Xiaofang Zhou
zxf@cse.ust.hk

1 Shenyang Aerospace University, Shenyang, China

2 Department of Computing, The Hong Kong Polytechnic University, Hong Kong SAR, China

3 DSA & INTR Thrust, The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China

4 Department of CSE, The Hong Kong University of Science and Technology, Hong Kong SAR, China

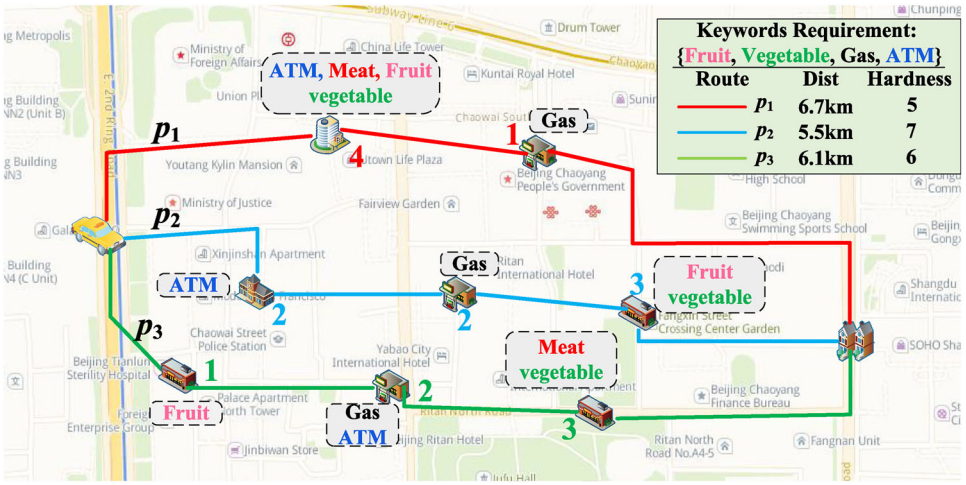

Fig. 1 Alternative Routes for Multi-Keywords Needs

refuel his car, purchase fruits and vegetables, and withdraw some cash. There are three potential routes p_1, p_2, and p_3, that could fulfill the user's needs based on the location of POIs and their associated keywords. Additionally, each POI has a hardness of stops represented by a positive integer. If the user demands two options, then p_2 and p_3 are returned as they are shorter than p_1. However, the total hardness of stops can be a significant inconvenience in urban areas, because users often consider not just the travel distance but also the hardness of stops required to complete their tasks. Therefore, p_1 with a slightly longer distance but its total hardness of stops is lower than both p_2 and p_3. Similarly, although the total hardness of stops for p_2 is the highest, it has the shortest route distance. Therefore, all three paths are meaningful choices for the user. Unfortunately, none of the existing solutions can achieve this.

To incorporate the hardness of stops into consideration, this work proposes the Skyline Optimal Route with Collective Spatial Keyword (Sky-ORCSK) problem, which aims to provide users with a set of non-dominated routes by considering both the distance and the hardness of stops. For the exact Sky-ORCSK solution, one approach is to invoke the KSP [42] (top-k shortest path) algorithm directly and enumerate the paths in the distance-increasing order continuously and test them one by one until the final skyline set is obtained. However, it needs to enumerate a very large value of k paths, while the *KSP* algorithm is relatively slow to run. Therefore, the KSP algorithm cannot be used to obtain the exact solution for the Sky-ORCSK problem. The second potential exact solution is to enumerate all POI combinations in the road network that cover the query keywords, use the TSP (Traveling Salesman) [34] algorithm to generate corresponding routes, and then derive the complete skyline result based on dominance relationships. However, the number of POI combinations grows exponentially with the number of query keywords. Moreover, constructing an exact shortest path for each POI combination has been proven to be an NP-hard problem in [41]. Therefore, designing an efficient exact algorithm goes beyond the scope of this work. As a result, in this paper, we propose two approximate algorithms to solve the Sky-ORCSK problem. Nonetheless, experimental results demonstrate that our proposed method yields close to optimal results in a much shorter time despite these theoretical limitations.

The first method leverages the spatial index IG-Tree [16], originally designed to find an optimal path that meets specific keywords. Although it was not intended to identify multiple paths, its index structure facilitates the rapid identification of relevant POIs. By generating a candidate set of POIs and then paths from this set, we can apply the proposed IG-Sky algorithm to efficiently produce a skyline set of non-dominated paths based on their dominance relationships.

Our second method, which is the primary focus of this paper, introduces the DA-Sky algorithm, a deviation-based skyline path expansion technique. Starting with the shortest path that meets initial query conditions, the algorithm extends from deviation points at POIs along the path, incorporating these POIs into the shortest path tree rooted at the destination. This creates a segmented route from the origin to the deviation point and onward to the destination. Through iterative deviations and prioritization via a distance-based priority queue, we efficiently explore new paths. To streamline this process and minimize unnecessary deviations, we introduce a Collective Keyword-aware Pseudo-Tree (*ckPT*). This structure uses bitmaps to monitor the satisfaction of query keywords across paths and organizes paths in a hierarchical manner, reducing the enumeration space for subsequent paths by focusing on new, unsatisfied keywords. To further improve efficiency, we propose five optimization strategies to prune the search space.

Our study presents several key contributions to the domain of keyword-aware routing as summarized below:

– We formally define the Sky-ORCSK problem to find paths that are more user-friendly by taking the hardness of stops into consideration;
– We introduce two innovative algorithms, IG-Sky and DA-Sky, to effectively answer Sky-ORCSK;
– To optimize the DA-Sky algorithm, we devise and implement several speedup techniques;
– Extensive experiments conducted on real data sets verify that our approaches perform better than the state-of-the-art in terms of quality and efficiency.

This work is an extension of our previous work DA-Prune [26] where we first propose k-ORCSK problem that returns top-k paths without considering the hardness of stops. Therefore, we extend the problem definition to form the skyline version and also extend the deviate algorithm to find the skyline ORCSK paths. Finally, in the experimental section, we also use DA-Prune as a baseline to validate the effectiveness of the algorithm proposed in this paper.

2 Related work

In this section, we categorize the existing solutions to keyword-aware routing and skyline-related routing.

2.1 The keyword-aware routing in road network

The keyword-aware routing algorithms are compared in Table 1 from the following three dimensions: POI order, keyword number, and *KSP* enumeration. Meanwhile, [28] also provides a review of keyword-aware route planning. Our previous work *DA-CSK* [26] can solve the k-ORCSK problem, but did not consider the skyline issue.

2.1.1 Single / multiple keyword in each POI

Firstly, most of the existing works [7, 27, 36, 49, 65] only support a single keyword in each POI. This may be caused by the fact that although the spatial keyword-aware routing problem seems very related to the spatial keyword search problem [54, 62, 63], it is actually more related to the route scheduling problem [29, 58], which regards the POIs as vertices. Because the single keyword POIs are equivalent to vertices and easy to handle, these works mainly focus on the optimality of the results and the influence of the visiting orders, which will be discussed in Section 2.1.2.

To solve the multiple keywords problem, *IG-Tree* [17] combines IR^2-*Tree* [11] and *G-Tree* [64] for efficient spatial keyword search and kNN search. *OptDist* [17] is the approximation of *IG-Tree* that chooses the best candidate POIs for each keyword in terms of source-keyword-destination distance and connects them greedily into a connected path. However, due to the local optimality of the greedy strategy, which overlooks the different combinations of POIs, the resulting path tends to be quite long. Some paths may even point in the opposite direction, away from the destination.

In addition, it can hardly adapt to the *k-ORCSK* problem. If the top-2 path is needed, to ensure that the POI set does not completely overlap with the top-1 path, one way is to remove a POI from the top-1 path and search again. However, it is difficult to decide which one to remove to get the least cost path. Suppose we have m required keywords and the average number of POIs that contain each keyword is n. Then the time complexity for top-1 is $O(nm^2 + m)$. When answering the top-k query, it has to try to remove each POI and search for substitutes for at least k times to obtain the shortest one. The complexity soars up to $O(kn + m^3n + km^2n^2)$.

MRRP [41] reduces visited POIs from the perspective of keyword coverage. Among its four methods, *kRA-MS* has the best performance by pruning and caching strategies. Unfortunately, it is costly to calculate the distance of the potential POIs, and the length of the chosen paths is not short enough due to the greedy choice of locally optimal POI. Moreover, it also cannot answer the top-k query quickly because the algorithm's execution process cannot perform path backtracking efficiently.

2.1.2 Fixed / arbitrary POI visit order

For the fixed order *OSR* query, [49] proposes the following three algorithms: 1) *LORD* is a threshold-based iterative algorithm which prunes the locations that cannot belong to the optimal route; 2) *R-LORD* extends *LORD* with *R-tree*; 3) *PNE* progressively issues *NN* queries on different point types to construct the optimal route. However, they cannot answer *k-ORCSK* directly. *KOSR* [36] solves this problem with two algorithms: *PruningKOSR* uses the dominance relationship to filter temporarily unnecessary routes and *StarKOSR* uses the target-directed strategy to find the optimal feasible routes. Although they only support sequenced keyword query with a single keyword, we can extend the *StarKOSR* to *k-ORCSK*.

For the arbitrary visiting order, *IG-Tree* and *MRRP* naturally support it. Apart from them, *KOR* [7] deals with limited budget and *SFS* [27] utilizes the partial order. *RCOSR* [65] takes the rating score of each POI in the optimal sequenced route into account to find routes that surpasses a rating threshold. These variations are also NP-H by reducing to (*TSP*), so their results are approximate with no bound. Due to different problem natures, they cannot solve our problem directly.

2.1.3 Top-*k* shortest path enumeration

Lastly, *KSP* finds top-k shortest paths in the distance-increasing order and most of the existing methods [8, 12, 18, 20, 42] have high complexity. We can use it to answer

Table 1 Spatial Keyword-Aware Optimal Route Algorithms

Technique	#kw in POI		Visit order		k		Metric	
	1	≥1	Fixed	Any	1	≥1	Distance	Hardness of stops
KSP [12], FKP [18]						√	√	
IG-Tree [17], MRRP [41]		√		√	√		√	
KOR [7], SFS [27]	√			√	√		√	
LORD [49]	√		√		√		√	
KOSR [36], RCOSR [65]	√		√			√	√	
DA-CSK [26]		√		√		√	√	
Sky-ORCSK (this paper)		√		√		√	√	√

k-ORCSK directly by testing the resulting as one of our baselines, but this blind-enumeration is very time-consuming.

2.2 Skyline Routing in Road Network

2.2.1 Skyline and Constrained Shortest Path

The skyline query [22, 50] identifies the non-dominated points from the multi-dimensional data sets. [19] introduces the skyline problem in road network by replacing the Euclidean distance with network distance along a pre-defined route, [10] replaces this route with multiple reference points, and [13, 43] increases the number of edge weights from single value to multiple ones. Nevertheless, these problems have nothing to do with routing; they only utilize network distance, which is essentially running several single-valued routing once on each dimension.

The real skyline routing problem returns a set of routes between the same origin-destination pair whose accumulated weights on a set of dimensions cannot dominate each other. [23] is the first one to provide a search algorithm with path expansion. However, this problem has a very high complexity and can hardly be used in real life. Therefore, its relaxed version *Constrained Shortest Path (CSP)* [38–40, 51, 52, 55] is more widely studied by only optimizing on dimension while the setting thresholds on other dimensions. Apart from the accumulated weights, edge labels are also considered as criteria to satisfy [57, 59, 60]. Nevertheless, none of the existing methods can handle the hardness of stops for POI, which is not a pre-defined weight on edge or vertex but can only be identified along routing, so they cannot solve our problem.

2.2.2 Keyword-Aware Skyline Path

Skyline Trips of Multiple POIs Categories (STMPC) [6] extends the keyword-aware routing by assigning each POI a cost such that this cost and distance can form a skyline. *Keyword-Aware Skyline Routes (KSR)* [48] is the only work that takes keyword cover and stopping time as skyline. However, its exhausted enumeration nature restricts it within indoor venues with hundreds of vertices, but it cannot scale to large road networks (Table 2).

Table 2 Important Notations

Notation	Definition
$sp(v_i, v_j)$, $\delta(v_i, v_j)$	Shortest path from v_i to v_j and its cost
$K(\cdot)$	Collective Keyword type set of vertices or a path
$O(kw_i)$	POI set containing the keyword kw_i
Q_{kw}	User-specified keywords set of query Q
$ReqP$	Set of POIs containing the required keyword in Q_{kw}
$c(v_i)$	The prefixed path from s to v_i
$w(p)$	The distance cost of path p
$h(v_p)$	The hardness of stops of POI v_p
$S(p)$	The total hardness of stops on path p
S_p	Skyline path set

3 Preliminaries

In this section, we define the problem and the corresponding concepts formally. Consider a road network represented as a graph $G(V, E)$, where V is the set of vertices and $E \subseteq V \times V$ represents the set of edges. Each edge $(u, v) \in E$ is assigned a non-negative weight $w(u, v)$ indicating the distance cost from u to v. A path p is a sequence of vertices $\langle v_0, \ldots, v_n \rangle$ where $(v_i, v_{i+1}) \in E$, $\forall i \in [0, n-1]$. The distance cost of p is denoted by $\omega(p) = \sum_{i=0}^{n-1} w(v_i, v_{i+1})$. If $v_i \in V$ is associated with some keywords (≥ 1) such as *ATM, fruit, meat, gas*, and etc, then it is regarded as a POI. $K(\{v_i\}) = \{kw_1, \ldots, kw_l\}$ represents the collective set of keywords associated with POI v_i. Because we have to stop when visiting a POI, then for each POI v_p, we introduce a *hardness of stops* $h(v_p)$ to capture the time or cost it takes to

park. The collective keyword type set $K(p)$ of a path p contains all keywords from the POIs it traversed. In addition, for efficient keyword-based querying, an inverted list $O(kw_i)$ is maintained for each keyword kw_i, storing all vertices that contain kw_i.

In this paper, we introduce the Sky-ORCSK query, which evaluates routes based on two dimensions: the route distance cost and the total hardness of stops. To fully comprehend the Sky-ORCSK query, we introduce the following two important concepts.

Definition 1 (Total hardness of stops) Given a path p that satisfies the user specified query keywords Q_{kw}, the total hardness of stops of p is denoted as $S(p)$. $S(p)$ can calculated by the sum of the hardness of stops for all POIs cover the query keywords along path p.

Definition 2 (Dominance relationship) Given two paths p_1 and p_2, p_1 is said to dominate p_2 ($p_1 \succ p_2$) if one of the following conditions is satisfied:

1) $\omega(p_1) < \omega(p_2)$ and $S(p_1) \leq S(p_2)$;
2) $\omega(p_1) \leq \omega(p_2)$ and $S(p_1) < S(p_2)$.

The dominance relationship is transitive; that is, if $p_1 \succ p_2$ and $p_2 \succ p_3$, then it logically follows that $p_1 \succ p_3$.Now we are ready to define our problem:

Definition 3 (Skyline routes with collective spatial keywords (Sky-ORCSK)) Given a road network $G(V, E)$, a starting vertex s, a destination t, and a set of query keywords $Q_{kw} \subseteq K(V)$, let P be all possible valid paths from s to t such that each one covers Q_{kw}, ORCSK aims to identify a path $p \in P$ and the corresponding set of POIs, denoted by R_p, such that $\omega(p) \leq \omega(p')$ for all $p' \in P \setminus \{p\}$, while Sky-ORCSK returns a subset of P, denoted as S_p. Each path in S_p covers all query keywords Q_{kw} and satisfies the following conditions:

1) $\forall p' \notin S_p, \exists p \in S_p \; s.t. \; p \succ p'$;
2) $\forall p \in S_p, \nexists p' \in S_p \; s.t. \; p' \succ p$. Note that, for each p_i in S_p, the stops number $S(p_i) = |R_{p_i}|$.

Given a query $Q(v_1, v_{15}, \{kw_1, kw_2, kw_3\})$ in Figure 2, each POI contains one or multiple keywords and the hardness of stops for each POI is showed in red. For this query, there paths are returned, denoted as $S_p\{kw_1, kw_2, kw_3\} = \{p_1, p_2, p_3\}$:

1) $p_1 = \langle v_1, \boldsymbol{v_4}, \boldsymbol{v_5}, \boldsymbol{v_9}, v_{13}, v_{15}\rangle$, with $\omega(p_1) = 19$ (shortest) and $S(p_1) = 3 + 2 + 1 = 6$ (Hardest to stop);
2) $p_2 = \langle v_1, \boldsymbol{v_2}, \boldsymbol{v_7}, v_8, v_{14}, v_{15}\rangle$, with $\omega(p_2) = 22$ (longer) and $S(p_1) = 2 + 3 = 5$ (Moderate hardness);

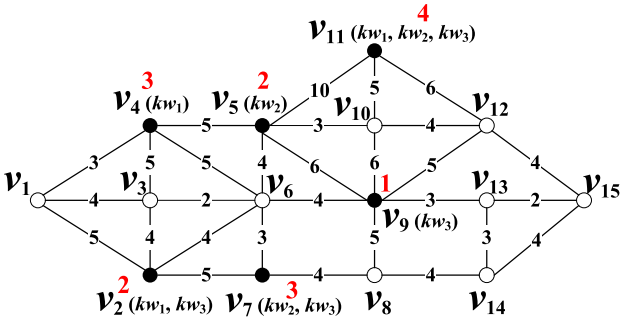

Fig. 2 Example of ORCSK and Sky-ORCSK Queries

3) $p_3 = \langle v_1, v_4, v_5, v_{10}, \boldsymbol{v_{11}}, v_{12}, v_{15}\rangle$, with $\omega(p_3) = 26$ (longest) and $S(p_3) = 4$ (Easiest to stop).

Users can select from these results based on their individual preferences, balancing between the distance of the path and the total hardness of stops.

4 SKY-ORCSK algorithms

As the Sky-ORCSK problem involves both spatial keyword searching and path finding, we propose two algorithms from these two perspectives.

4.1 IG-Sky algorithm

This algorithm chooses the best candidate POIs for each keyword in terms of source-keyword-destination distance by utilizing a spatial index and connecting them greedily into a connected path. The IG-Tree [17] index was originally designed to solve ORCSK, which combines IR^2-Tree [11] and G-Tree [64] for efficient spatial keyword search and kNN search, and we extend it to solve Sky-ORCSK. Firstly, it also uses METIS [21] to partition and organize the graph into a hierarchical structure. The keywords are accumulated within different levels of partitions, and distances between partition boundaries are also precomputed. During the query process, IG-Tree searches for the next POI greedily. First, it sorts the POIs by the shortest distance from the starting point. Then, it connects the nearest POI that contains the unsatisfied keywords and uses it as the next starting point to search for the remaining unsatisfied keywords iteratively until all keywords are covered. However, a notable limitation of IG-Tree is its ability to query only a single optimal path in terms of distance, whereas the Sky-ORCSK problem necessitates planning multiple distinct paths. Although IG-Tree cannot directly solve the Sky-ORCSK problem, its index can still be effectively used to identify suitable POI candidate sets. From these sets, multiple candidate paths can be generated using

a novel path generation method. Based on this conceptual framework, we propose the IG-Sky algorithm as follows.

Given a query's origin s and destination t, a set of query keywords, the IG-Sky algorithm first determines their *Lowest Common Ancestor (LCA)* in the IG-Tree, denoted as LCA(s, t). The inverted list, constructed during the index's construction, stores which keywords are present in the partition corresponding to the LCA(s, t) node. If the POIs in this partition do not encompass all query keywords, the search ascends to the parent node of LCA(s, t) and tests the keywords iteratively until finding a partition node that contains all query keywords. After that, all leaf nodes under this node are saved, excluding those that lack the query keywords, thereby yielding a candidate set of POIs containing the necessary keywords within the partition. Subsequent path generation begins with the establishment of a shortest-distance path, following a greedy approach. This initial path serves as a basis for further iterations, wherein POIs from the candidate set, sharing the same keywords, are used to replace nodes in the path. This replacement occurs in various combinations, considering the replacement of one, two, or more POIs, ultimately leading to a set of multiple alternative paths.

It is worth noting that the candidate set only specifies the POIs to be visited, not their visiting sequence. Since the order of POIs post-replacement may not yield the shortest path, a reconnection of POIs is necessary. This reconnection also adopts a greedy strategy, beginning with the nearest POI to s and proceeding to iteratively connect the closest remaining POIs, concluding with a connection to t. Through this methodology, a candidate set encompassing multiple paths that satisfy the query keywords is obtained. Finally, the dominance relationship between these paths is applied to filter and determine the final path results. While generating the paths, IG-Sky entails numerous shortest-distance calculations, which is time-consuming. To speed up these calculations, the SOTA distance index H2H [44] is integrated.

Figure 3 shows the IG-Tree built for the road network in Figure 2 and an example of processing the Sky-ORCSK query $Q(v_1, v_{12}, \{kw_1, kw_2\})$. The LCA$(v_1, v_{12})$ is identified as G_1 first, which covers all the required keywords types kw_1 and kw_2. Then the POI candidate set is obtained as $\{v_4, v_5, v_{11}\}$ who are contained under G_1. Based on these candidate POIs, a valid path from v_1 to v_{12} is generated as the original path, i.e., $p_1 = \langle v_1, \boldsymbol{v_4}, \boldsymbol{v_5}, v_{10}, v_{12}\rangle$ with the stay POIs $\{v_4, v_5\}$. Since v_{11} satisfies both v_1 to v_{12}, it can replace v_4 and v_5 and a new path can be obtained, i.e., $p_2 = \langle v_1, v_4, v_5, v_{10}, \boldsymbol{v_{11}}, v_{12}\rangle$. Then, according to the dominance relationship between the routes, the result set S_p, including p_1 and p_2 is found and returned.

Complexity Analysis. The time complexity of the IG-Sky algorithm is mainly in the node replacement process, assuming that the number of POI nodes in the base path is p. The time complexity to cover all the replacement cases is any

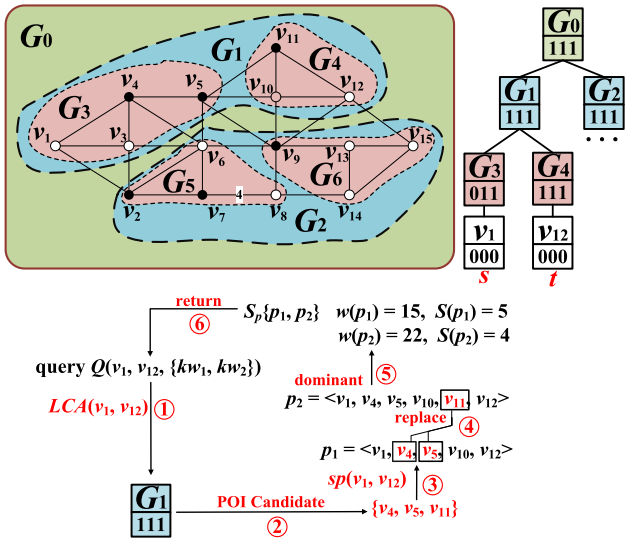

Fig. 3 IG-tree Index and Sky-ORCSK Query

number of combinations of p, i.e., $Q(2^p)$. For each combination, POIs with the same keywords need to be replaced, and in the worst case, POIs with all combinations of keywords need to be replaced, i.e., the total number of POI candidate sets $|P_{cand}|$. Therefore, the IG-Sky algorithm has a time complexity of $O(|P_{cand}|2^p)$ in the worst case.

4.2 DA-Sky Algorithm

Although the IG-Sky algorithm can obtain query results quickly, its resulting paths are often quite long. The experimental results shows that the paths generated by IG-Sky are often two to three times longer than $\delta(s, t)$. This is because IG-Sky still adopts a greedy strategy to construct the path after obtaining the candidate set, which leads to the persistence of the local optimality problem. Therefore, to improve the quality of the results, in this section, we propose the DA-Sky algorithm, which expands the road network edge by edge based on the concept of path deviation.

Firstly, the vertices along $sp(s, t)$ represent potential deviation points for detours to satisfy additional keywords. Thus, we iteratively explore each deviation vertex, seeking the nearest POI that contains unsatisfied query keywords. Upon identifying such a POI, it becomes a new source vertex, from which we compute the shortest path to t. This process also defines a second level of deviation nodes for further exploration. This recursive exploration of the road network continues until all query keywords are covered. Subsequently, the corresponding paths are added to the result set. This result set is continuously updated to reflect the dominance relationship among the paths, ensuring that the final set comprises only the non-dominated paths that optimally satisfy the query criteria.

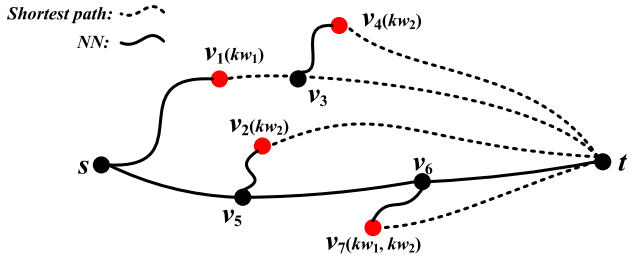

Fig. 4 Example for General Idea of *DA-Sky*

Figure 4 shows a general ides of DA-Sky, given a query $Q(s, t, \{kw_1, kw_2\})$, the first step involves retrieving $sp(s, t)$ and identifying the first level of deviation nodes: s, v_5, and v_6. For each deviation node, we search its nearest POIs w.r.t the unsatisfied query keywords. For example, the subpath $\langle s, v_1\rangle$ covers kw_1, and we then proceed to find the shortest path from v_1 to t, encountering the next POI v_4 (containing kw_2) at the second level of deviation node v_3. Consequently, the path $\langle s, v_1, v_3, v_4, t\rangle$ emerges as a candidate. This process is repeated for deviation nodes v_5 and v_6 to generate additional candidate paths. Each time a valid path is found, it is added to the result set. This result set is continually updated to reflect the dominance relationships among the paths, thereby ensuring that the final set comprises only the optimal paths fulfilling the query criteria.

However, this strategy presents significant challenges in implementation. One of the primary concerns is the lack of a definitive order for the deviated paths, which compromises the ability to systematically control the exploration process. Additionally, there is no inherent quality assurance for the paths generated. Compounding these issues is the immense search space, which leads to an excessive number of path retrievals and, consequently, this process places a considerable computational burden. To mitigate these challenges, we propose utilizing a path enumeration framework to answer the Sky-ORCSK query. By applying deviation principles within this framework, we aim to significantly enhance the efficiency of the search process, ensuring more controlled and quality-assured path discovery.

4.2.1 Enumeration Framework and Limitations

The path enumeration procedure is outlined in Algorithm 1. We initiate the process by identifying $sp(s, t)$, denoted as p_0, and enqueueing it into a priority queue PQ. Each path in PQ also records the last vertex of its prefix (initially s for p_0). The algorithm iteratively retrieves the top path p_{top} from PQ (Line 4) and updates the result set S_p (Lines 6-17) if it covers Q_{kw}. The update mechanism operates as follows: If S_p is empty, the new path is directly added to the set. Otherwise, it compares the new path with the tail path in S_p (i.e., the most recently added path). If the tail path dominates the new path, the new path is discarded, indicating its inferiority. Conversely, if the new path dominates the tail path, the tail path is removed from S_p, and this comparison process continues with the next tail path in S_p until no dominance is established. The new path is then added to S_p. If p_{top} does not satisfy all keywords, the algorithm proceeds to expand p_{top} to generate new candidate paths. Specifically, it finds the shortest new path from the last prefix vertex v of p_{top} to t, excluding vertices in the prefix and the subsequent edge of v. A new candidate path p' is formed by concatenating this new segment with the prefix. This new path p' is then inserted into PQ with its last prefix vertex updated to v. The process iterates, advancing the prefix to the next vertex along p_{top} and repeating until v reaches t. The algorithm terminates when PQ is empty, indicating that all paths have completed their deviations.

To enhance the efficiency of our algorithm, we propose several strategies in the following sections.

Algorithm 1: Path Enumeration-Based Sky-ORCSK

Input: $G(V, E)$, $Q(s, t, Q_{kw})$
Output: The Skyline paths set S_p

```
1  p_0 ← ShortestPath(s,t), Prefix[p_0] ← {s}, S_p ← ϕ;
2  PQ.insert(p_0, δ(p_0));
3  while PQ is not empty do
4      p_top ← PQ.pop();
5      if K(p_top) = Q_kw then
6          if S_p = ϕ then
7              S_p ← S_p ∪ p_top;
8          else
9              p_tmp ← S_p.back();
10             if p_tmp ≻ p_top then
11                 continue;
12             while p_top ≻ p_tmp do
13                 S_p.pop_back();
14                 if S_p = ϕ then
15                     break;
16                 p_tmp ← S_p.back();
17             S_p ← S_p ∪ p_top;
18     p_pre ← Prefix[p_top];
19     for v ∈ p_top/p_pre do
20         //The search avoids p_pre and the next edge (v,u) ∈ p_top;
21         p' ← Prefix[p_top] ⊕ ShortestPath(v,t);
22         PQ.insert(p', δ(p'));
23         p_pre ← p_pre ∪ {v}, Prefix[p'] ← p_pre;
24 return R_p;
```

4.2.2 Collective Keyword-Aware Pseudo Tree ckPT

As paths generated in Algorithm 1 derive from one another, they exhibit relational dependencies, particularly in terms of shared prefixes. To efficiently represent these relationships, we organize the paths within a tree structure. This structure not only stores the path relations compactly but also facil-

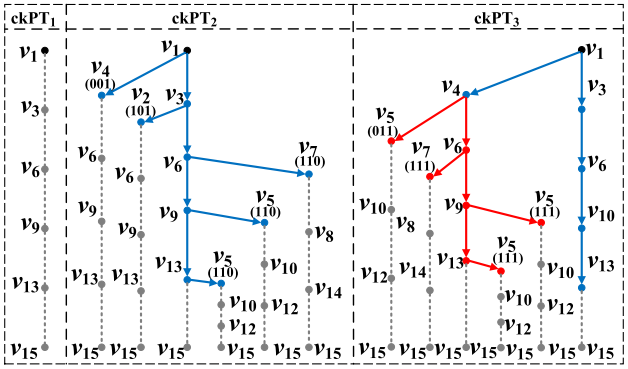

Fig. 5 *ckPT*s for $Q(v_1, v_{15}, \{kw_1, kw_2, kw_3\})$

itates the sharing of satisfied keyword information among common prefixes. To realize this concept, we introduce the *ckPT*, specifically designed to organize path relations and keyword satisfaction information efficiently.

ckPT Structure: To facilitate a keyword-oriented enumeration paradigm, the search process is directed towards unsatisfied keywords by initiating a nearest neighbor (*NN*) query from deviation points. The *ckPT* is employed to store concatenated paths comprising the prefix paths and their corresponding *NN* results. Specifically, the root of the *ckPT* is s, and the nodes within the *ckPT* represent vertices on the candidate partial paths. The leaf nodes are the *NN* results obtained in the previous step, with each capable of satisfying at least one query keyword. We denote the prefixed path from s to a vertex v_i as $c(v_i)$. To expedite the process of examining unsatisfied keywords along a path, each node in the *ckPT* is equipped with a bitmap. This bitmap captures the keyword satisfaction status, thereby allowing for rapid checking of keyword satisfaction for two partial paths using bitwise operations. It is important to note that a single vertex may appear multiple times within the *ckPT*, as it can be the nearest neighbor for different deviation nodes and thus part of distinct paths.

ckPT Construction: When a partial path $c(v_m)$ is retrieved from the priority queue PQ, the algorithm locates the corresponding path and its endpoint v_m within the *ckPT*, where v_m is a leaf node. Subsequently, v_m undergoes expansion, with each point along the shortest path $sp(v_m, t)$ considered as a new deviation point. For each deviation, the nearest required POI is identified. The segments from v_m to these deviations, and from the deviations to their respective nearest POIs, are then appended to the *ckPT* as subsequent paths branching from v_m. Unique path IDs are assigned to these diverging paths, each terminating at the identified POIs.

Given the query $Q(v_1, v_{15}, \{kw_1, kw_2, kw_3\})$. Figure 5 shows the *ckPT* in the context of Figure 1. Nodes and paths are represented as colored circles and solid lines, respectively, while those on the shortest path are depicted in grey and dashed lines. The pseudo tree is initiated with source vertex v_1 as the root, designated as $ckPT_1$. $ckPT_2$ is derived by expanding $c(v_1)$ in $ckPT_1$, considering v_1 as the sole leaf node. This expansion produces deviation nodes v_1, v_3, v_6, v_9, v_{13}, and identifies the nearest required POIs to these nodes as v_4, v_2, v_7, v_5. Consequently, subpaths $\langle v_1, v_4\rangle$, $\langle v_1, v_3, v_2\rangle$, $\langle v_1, v_3, v_6, v_7\rangle$, $\langle v_1, v_3, v_6, v_9, v_5\rangle$, and $\langle v_1, v_3, v_6, v_9, v_{13}, v_5\rangle$ are appended to v_1. Additionally, bitmaps indicating the covered keywords in the paths are stored; for example, the path from v_1 to v_7 includes kw_2 and kw_3, thus $c(v_7).\text{bit} = 110$. Similarly, $ckPT_3$ is formulated by selecting and expanding a leaf node in $ckPT_2$. Due to space constraints, only a partial view of $ckPT_3$ with $c(v_4)$ as its root is displayed in the figure (marked in red).

4.2.3 Bitmap Partial Shortest Path Tree SPT$_b$

During the expansion phase, calculating the shortest paths from various leaf nodes to t is necessary to identify potential deviations. To circumvent the inefficiency of repeated graph traversal, we construct a reverse shortest path tree rooted at t in advance. Furthermore, a complete path from s to t can be efficiently obtained at any point by concatenating the *ckPT* with the reverse shortest path tree. This approach significantly reduces the need for expensive graph search operations, thereby enhancing the overall efficiency of the path-finding process.

It is noteworthy that the leaf nodes of the *ckPT* are always capable of providing at least one query keyword. Therefore, it suffices to store only the shortest paths from the required POIs to t. This approach necessitates maintaining only a partial shortest path tree, denoted as SPT_b. To construct $textit_b$, we first determine the set of required POIs, $ReqP$, using the inverted index for each type of query keyword. Subsequently, *Dijkstra*'s algorithm initiates a search from t and continues until all POIs in $ReqP$ are included. During this process, a bitmap is employed to effectively organize the keywords, allowing for the easy accumulation of satisfied keywords for each path. These bitmaps are then incorporated into SPT_b.

Figure 6 depicts an instance of SPT_b. The set $ReqP$ consists of POIs containing kw_1 and kw_2, thus $ReqP = \{v_2, v_4, v_5, v_7, v_{11}\}$. The network is expanded using *Dijkstra* until all POIs in $ReqP$ are encountered. During this expansion, bitmaps representing the keywords are embedded into SPT_b. For instance, the bitmap for v_2 is $bit(v_2) = 101$, reflecting that the union of keywords at v_2 and v_1 is $\{kw_1, kw_3\}$. Similarly, $bit(v_5) = 011$, where its parent node's bitmap is 001 and the keyword set for v_5 itself $K(v_5)$ is $\{kw_2\}$.

The space complexity of bitmaps: The space complexity of the bitmap mainly depends on the number of query keywords $|Q_{kw}|$ and the number of nodes in the SPT_b. For each node in the SPT_b, the bitmap size is $O(|Q_{kw}|/8)$ bytes.

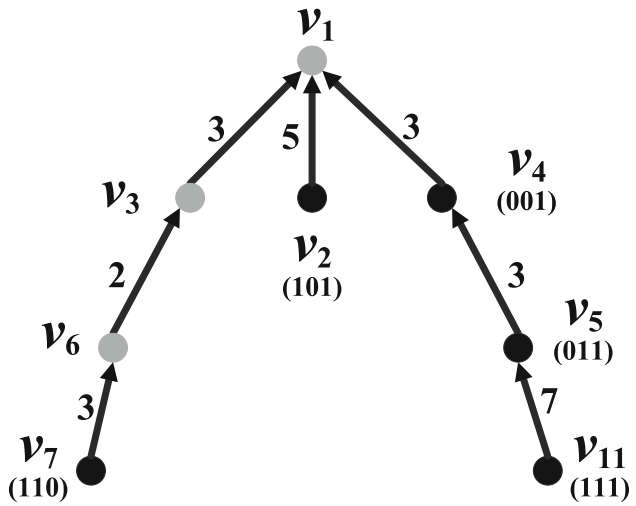

Fig. 6 SPT$_b$ rooted at v_1 for a query $Q(*, v_1, \{kw_1, kw_2\})$ of Figure 2.

Therefore, in the worst case, the total space complexity of SPT_b is $O(|V| \cdot |Q_{kw}|/8)$ bytes, where $|V|$ is the number of vertices of map. This indicate that despite the potential for a large number of keywords, the size of the bitmap remains limited to the number of keywords used in the query. As such, the bitmap's memory footprint is minimal.

4.2.4 Best First Path Expansion

The plethora of deviations from leaf nodes in the *ckPT* to the destination engenders an exponential expansion of the *ckPT*. Expanding partial paths within the *ckPT* solely based on distance criteria can inadvertently introduce numerous non-contributory partial paths, thus inflating both spatial and temporal overheads. To mitigate this, we incorporate unsatisfied keywords into a heuristic distance measure and employ a best-first search paradigm. This strategy aims to prioritize the discovery of valid paths, minimizing unnecessary expansions.

An effective heuristic should not only be computationally efficient but also approximate the actual costs closely, guiding the expansion towards the most promising directions. For the Sky-ORCSK problem, path quality is evaluated based on two dimensions: the path distance $w(p)$ and the total hardness of stops $S(p)$. It follows that paths characterized by lower $w(p)$ and $S(p)$ values are more competitive. Accordingly, the heuristic is designed to steer path expansion towards POIs that satisfy multiple query keywords, thereby optimizing both the distance and the comprehensiveness of keyword coverage.

To enhance efficiency, we streamline the calculation by concentrating on POIs that encompass multiple query keywords, temporarily setting aside the intricate keyword combinations. We define "*difficulty*" as the maximum lower bound of the detour cost associated with reaching these POIs.

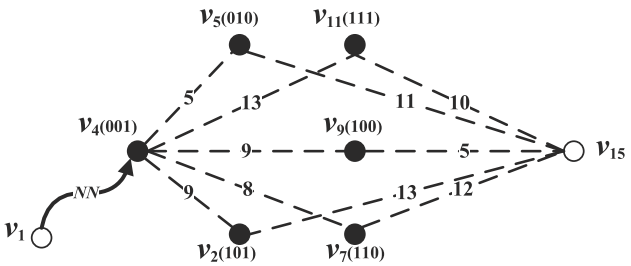

Fig. 7 Calculation of max_{LB} for v_4.

Specifically, for each partial path $c(v_i)$, we first identify the set of unsatisfied keywords $Q'_{kw} = Q_{kw} \backslash c(v_i).\text{bit}$. For every keyword $kw_m \in Q'_{kw}$, we utilize the reverse index to ascertain the POI candidate set (C_{kw_m}) that includes this keyword along with at least one other query keyword. We then calculate the minimum detour cost to reach any POI in C_{kw_m}, and the highest one is selected as the lower bound cost for satisfying the remaining keywords. Consequently, the heuristic cost for any valid path prefixed by $c(v_i)$ is determined by this lower bound cost, offering a simplified yet effective metric for guiding the search towards POIs that fulfill multiple query keywords. To sum up, the heuristic cost of the valid path $lb(c(v_i))$ prefixed by $c(v_i)$ is calculated by:

$$lb(c(v_i)) = \omega(c(v_i)) + max_{LB} = \omega(c(v_i)) + \max_{\forall kw_m \in Q'_{kw}} \left(\min_{\forall v_j \in C_{kw_m}} (\delta(v_i, v_j) + \delta(v_j, t)) \right) \quad (1)$$

In Figure 7, $Q_{kw} = \{kw_1, kw_2, kw_3\}$ and the keywords covered by the partial path $K(c(v_4)) = \{kw_1\}$, thus $Q'_{kw} = \{kw_2, kw_3\}$. The detour cost for reaching each POI v_j, $\delta(v_4, v_j) + \delta(v_j, t)$ can be efficiently retrieved using any path index. Upon evaluating the sum distances, for kw_2, its $C_{kw_2} = \{v_5, v_7, v_{11}\}$ with a minimum detour cost of 16. Similarly, for kw_3, $C_{kw_3} = \{v_2, v_7, v_9, v_{11}\}$, among which v_9 presents the smallest detour cost of 14. Therefore, the maximum of these detour costs, denoted as $max_{LB} = max\{16, 14\} = 16$, is attributed to v_5.

Heuristic Value Calculation Improvement Strategy. The determination of heuristic values necessitates calculating numerous shortest distances, a process that can be computationally intensive, particularly with a large set of unsatisfied keywords. To mitigate this, we leverage a path index to efficiently retrieve the shortest distances. Furthermore, we implement the following pruning techniques to further reduce computational overhead: 1) The detour cost for each POI is computed only once, eliminating the need for redundant calculations. 2) If the detour cost for a POI containing keyword kw_b is less than the current max_{LB}, it implies that the minimum detour cost for kw_b will not influence the update of

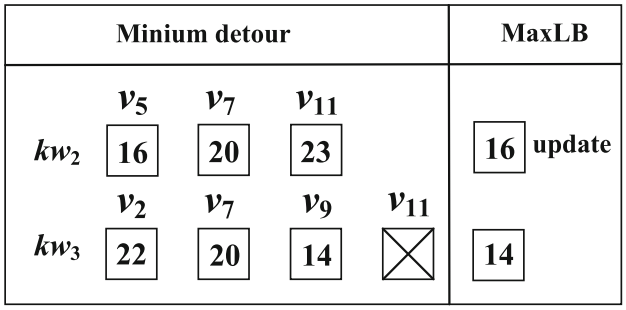

Fig. 8 Efficient max_{LB} Calculation of Figure 7

max_{LB}. Consequently, there is no necessity to compute the detour costs for other POIs that also contain kw_b.

The refined approach to calculating max_{LB} as depicted in Figure 8 enhances efficiency. Initially, the minimum detour cost associated with kw_2 is determined by evaluating the detour distances for v_5, v_7, and v_{11}, yielding a value of 16. This value is subsequently utilized to update max_{LB}. In the case of kw_3, upon computing the detour distance for v_9 and finding its minimum detour cost to be 14, which is less than the current max_{LB}, so the detour costs of other POIs (e.g., v_{11}) containing kw_3 do not need to calculate.

4.2.5 Deviation Algorithm (*DA-Sky*)

Now, we are poised to delineate our deviation-based algorithm, *DA-Sky*, incorporating the previously introduced components, with comprehensive details presented in Algorithm 2. The initial step involves constructing SPT_b rooted at t. Subsequently, s is integrated into *ckPT* as the root, and the prefix path $c(s)$ is enqueued into PQ after the computation of its max_{LB} (Lines 1-4).

PQ orchestrates the management of candidate partial paths that have been expanded, utilizing tuples of the form $\langle pathID, lb(c(v_i))\rangle$, where $pathID$ signifies the ID of the current partial path $c(v_i)$ linked to *ckPT* (for simplicity, $c(v_i)$ on *ckPT* is utilized in lieu of $pathID$), and $lb(c(v_i))$ represents the lower bound cost for paths prefixed with $c(v_i)$, as determined by Equation 1. PQ arranges partial paths based on $lb(c(v_i))$ and sequentially retrieves the path with the minimal aggregate cost during each iteration. Newly generated partial paths are likewise orderly inserted into PQ.

For each partial path $c(v_m)$ retrieved from PQ, the algorithm checks its validity. If the path is deemed valid, the result set S_p is updated accordingly (Lines 8-9), mirroring the update mechanism detailed in Algorithm 1 (Lines 6-17). Conversely, if the path is not valid, the algorithm proceeds to identify all vertices on the shortest path from v_m to t, treating these vertices as new deviation points (Line 10). Notably, the *IER* algorithm [45] is employed within the *FindNN* function,

Algorithm 2: *DA-Sky*(G, s, t, Q_{kw})

Input: $G(V, E)$, $Q\{s, t, Q_{kw}\}$
Output: The Skyline paths set S_p

1 Construct SPT_b rooted at t, Initialize PQ;
2 $S_p \leftarrow \emptyset$, $c(s) \leftarrow s$, $ckPT_1 \leftarrow \{c(s)\}$;
3 $max_{LB} \leftarrow CalMaxLB(c(s), \emptyset)$;
4 $PQ.insert(\langle c(s), \omega(c(s)), max_{LB}\rangle)$;
5 **while** PQ *is not empty* **do**
6 $c(v_m) \leftarrow PQ.extract\_Min()$;
7 $Bit \leftarrow c(v_m)$.bit in SPT_b;
8 **if** $K(c(v_m)) \vee Bit = Q_{kw}$ **then**
9 Same as lines 5-16 of Algorithm1
10 $C \leftarrow$ Get all vertices on the shortest path from v_m to t utilizing SPT_b **for** *each* v_i *in* C **do**
11 $c(v_i) \leftarrow c(v_m) \oplus v_i$;
12 $NN_{v_i} \leftarrow FindNN(v_i, K(c(v_i)))$;
13 $P_a \leftarrow c(v_i) \oplus sp(v_i, NN_{v_i})$;
14 $P_a \leftarrow PathRefine(P_a)$;
15 $cost \leftarrow w(P_a)$;
16 $ckPT \leftarrow ckPT \oplus P_a$;
17 $K(P_a) \leftarrow K(c(v_i)) \vee K(\{NN_{v_i}\})$;
18 $max_{LB} \leftarrow CalMaxLB(P_a, K(P_a))$;
19 $PQ.insert(\langle P_a, cost + max_{LB}\rangle)$;
20 return S_p;

in conjunction with the state-of-the-art *H2H* technique [44], to enhance the efficiency of *NN* queries.

For each deviation identified, the *NN* POI is determined and utilized to formulate a new partial path. This process involves refining the new path, eliminating superfluous nodes to guarantee the minimal path distance before integrating it into *ckPT* (Lines 11-16). Concurrently, a bitmap representing the collective keywords is recorded at the corresponding node (Line 17). Subsequently, the maximum lower bound of the detour cost for each new partial path is calculated. This value, along with the path's own cost, is then enqueued into PQ for future consideration (Lines 18-19).

Complexity Analysis. Since we use *H2H* for shortest distance computation, we regard the *FindNN* process as the minimum calculation unit in our paper. In the following, 1) d denotes the average number of deviation nodes for each candidate partial path in *ckPT*. It is bounded by the edge number $|E|$. 2) f denotes the average inverted index size of each type keyword, which is at most $|V|$. 3) g denotes the number of remaining keywords Q'_{kw}, so it is bounded by Q_{kw}. 4) p denotes the number of remaining undeviated candidate paths, which is the same as the number of candidate paths that have been enumerated. 5) h_{max} denotes the maximum hardness of each POI. Firstly, *FindNN* and *CalMaxLB* run f times. In each iteration of *CalMaxLB*, we compute the lower bound of a deviated route twice, so it takes $O(2f \cdot g)$ time. Thus, the time complexity of processing a candidate path is $O(d(1 + 2f \cdot g))$. When considering the hardness, it has $|Q_{kw}| \cdot h_{max}$ skyline solutions, so the over complexity is $O(d \cdot f \cdot g \cdot |Q_{kw}| \cdot h_{max})$. Since our priority queue PQ is

structured with a binary heap, and the algorithm terminates when the remaining candidate paths in the queue PQ are all expanded. So we can conclude that the worst time complexity of *DA-Sky* is $O(|Q_{kw}|\log(d \cdot f \cdot g \cdot p \cdot |Q_{kw}| \cdot h_{max})) = O(|Q_{kw}|\log(|V| \cdot |E| \cdot |Q_{kw}| \cdot p \cdot h_{max}))$.

5 Optimization strategies

While the *ckPT* and optimal path expansion mechanisms in DA-Sky proficiently navigate the path towards completion, they entail an exhaustive search across nearly all possible paths, relying solely on the dominance relationship to check paths without any pruning until the expansion of the queue concludes. To address this problem, in this section, we propose five pruning and optimization strategies. These strategies preemptively filter out partial paths that are unlikely to contribute to the final result set. By circumventing non-essential path expansions, these pruning techniques effectively reduce the queue size, not only lowering the algorithm's memory overhead but also expediting the query process.

5.1 Upper bound pruning strategy

According to the dominance relationship definition provided in Definition 2, when one dimension of a path attains its minimum value, the corresponding value in the other dimension becomes the upper bound for that dimension. Therefore, this provides us with an opportunity to prune paths in advance. For instance, when a path p_i's distance reaches its minimum value, denoted as $w_{min}(p_i)$, the total hardness of stops $S(p_i)$ is established as S_{ub}. This rationale stems from the understanding that other paths cannot surpass p_i in terms of distance. Thus, only paths with total hardness of stops less than S_{ub} avoid being dominated, positioning $w_{min}(p_i)$ as an effective upper boundary.

As paths with the total hardness of stops larger than this boundary are subject to discard, Algorithm 2 ensures that paths are expanded from the queue based on ascending distance order. Consequently, valid paths are generated sequentially from the shortest to the longest distance. The total hardness of stops associated with each valid path generated during expansion becomes S_{ub} for subsequent paths. Once a valid path is incorporated into the result set Sp after passing through dominance filtering, its total hardness of stops sets S_{ub} for paths in future expansions. Hence, paths exceeding this stop hardness threshold in subsequent expansions are directly pruned, preventing their entry into PQ.

Strategy 1 ***(Upper-Bound Pruning Strategy)*** *Let S_{ub} denote the upper bound of the total hardness of stops. If $S(c(v_i)) > S_{ub}$, then the partial path $c(v_i)$ can be safely discarded without being enqueued into the priority queue PQ.*

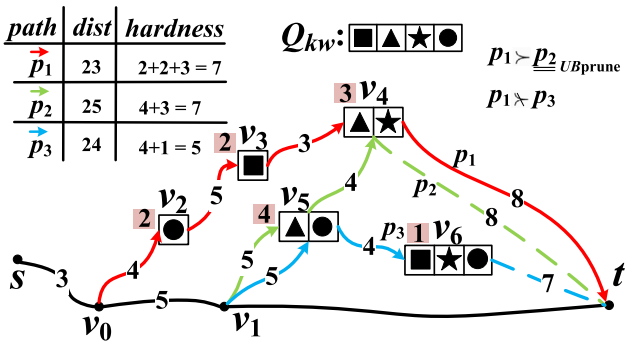

Fig. 9 Example of the Upper Bound Pruning

In pursuit of identifying query results at the earliest, the upper bound represented by S_{ub} is continuously refined. This refinement process involves updating S_{ub} using the path with the lowest total hardness of stops that has already been merged into the Skyline result set. As the algorithm progresses and path costs incrementally increase, S_{ub} correspondingly decreases with each addition to the skyline result set. This dynamic adjustment not only facilitates the early detection of optimal paths but also increases the efficiency of the pruning process. The reduction in S_{ub} enhances the likelihood of pruning partial paths, thereby streamlining the search process and expediting the attainment of query results.

Figure 9 illustrates an example of upper bound pruning. Given $Q(s, t, Q_{kw})$, the DA-Sky algorithm initially identifies $sp(s, t)$, subsequently initiating deviations based on the nodes along this path. $p_1 = \langle s, v_0, v_2, v_3, v_4, t\rangle$ represents a valid path derived from deviation point v_0, which has been incorporated into the result set S_p. This path traverses nodes v_2, v_3, and v_4, therefore the total hardness of stops of this path is 7, thereby setting S_{ub} to 7. The figure also delineates two partial paths, $p_2 = \langle s, v_0, v_1, v_5, v_4, t\rangle$ and $p_3 = \langle s, v_0, v_1, v_5, v_6, t\rangle$, emerging from deviation point v_1. Path p_2, having encountered two POI points, v_4 and v_5, satisfies three out of four query keywords, rendering its total hardness of stops is also equal to 7. In accordance with the pruning strategy 1, p_2 is subject to pruning. Conversely, path p_3, which passes through nodes v_5 and v_6 and fulfills all query keywords, its total hardness of stops is 5 and lower than S_{ub}, therefore path p_3 is preserved. Upon the completion and potential addition of path p_3 to the result set S_p, S_{ub} would be updated to 5, reflecting the dynamic refinement of the upper bound in response to evolving query results.

5.2 Nearest neighbor (NN) pruning strategy

During the deviation phase, the DA-Sky algorithm searches for POIs that satisfy the query keywords along the shortest

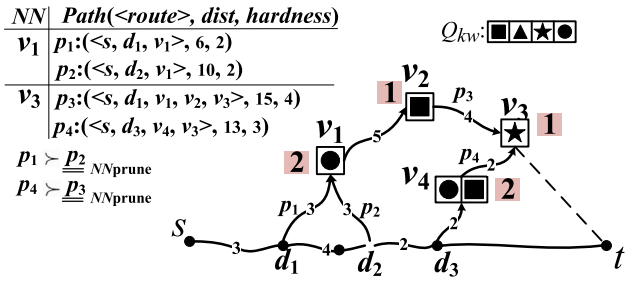

Fig. 10 Example of the NN Pruning Strategy

path. It identifies the nearest POI and then connects this NN node directly to t via the SPT_b, thereby generating a new temporary path. This procedure does not preclude multiple nodes from deviating to the same POI, resulting in the formation of several distinct temporary paths that are subsequently enqueued in PQ. However, as the segments of these paths from NN to t are identical, with only the prefix paths from s to NN differing, it is the prefix paths that ultimately determine the dominance hierarchy among these paths. Consequently, it suffices to retain only the optimal prefix path, while the others may be pruned without being added to PQ. This approach effectively minimizes PQ insertion operations and reduces the queue size, thereby curtailing unnecessary path expansions and enhancing overall algorithm efficiency.

Identifying all deviation points extending to the same NN directly is impractical. Therefore, we dynamically update the minimum cost for each NN throughout the extension process. If extending a new path to the same NN incurs a cost greater than the currently recorded cost for that NN, the path is promptly pruned. The minimum cost associated with an NN is revised downward whenever a new, less expensive path is identified. To efficiently manage the cost records for each NN, a hash table, denoted as $H.NN(v_i)$, is established. This table records the current path distance $H(v_i)$.dist and the total hardness of stops $H(v_i)$.stop for each NN v_i, thus streamlining the extension process by ensuring only the most cost-effective paths are pursued.

Strategy 2 ***(NN Pruning Strategy)*** *Suppose path p_0 is the new path obtained via the deviation to the nearest POI v_a from node v_0. If the conditions $w(p_0) > H(v_a).dist$ and $S(p_0) \geq H(v_a).stop$ or $w(p_0) \geq H(v_a).dist$ and $S(p_0) > H(v_a).stop$ are met, then p_0 will be pruned off.*

As illustrated in Figure 10, deviation points d_1 and d_2 both lead to node v_1, resulting in the generation of two distinct partial paths p_1 and p_2, respectively. Despite the total hardness of them are same, $\omega(p_1)$ is less than $\omega(p_2)$, allowing p_1 to dominate p_2 and resulting in the pruning of p_2. Since p_1 is generated first, the value of $H(v_1)$ remains as the cost of path p_1. In a similar vein, two paths lead to node v_3. Assuming path p_3 is generated initially, the cost associated with $H(v_3)$ corresponds to that of p_3. Subsequently, when path p_4 is generated with both a lower distance cost and lower total hardness of stops than those recorded by $H(v_3)$, p_3 is pruned, prompting an update of $H(v_3)$ to reflect the cost of path p_4.

5.3 Deviate pruning strategy

When DA-Sky expands a path, it takes all the nodes on this path as deviation nodes and generates new paths by expanding outwards from the nearest POI. However, when the deviation points close to the end of the deviation path are deviating, they will go around to a farther distance to search for POIs because there may not be the required POI points in the direction of the approaching the end point, so that the generated paths are costly in terms of distance, and do not occupy the advantage of becoming the result. Meanwhile, for the forward deviation points, since they are farther away from the end of the path, they will always find POIs in the direction of the convergence of ends, and the generated path distance is shorter and more advantageous. As shown in the structure of $ckPT$ in Figure 5, in $ckPT_2$, the deviation points v_9, and v_{13} in the deviation path $\langle v_1, v_3, v_6, v_9, v_{13}, v_{15}\rangle$ close to the end point v_{15} are both deviated to v_5. And it can be seen in figure 2 that v_9 and v_{13} are deviated in the opposite direction towards v_1 to v_{15}, and the path distance is inevitably larger. While the previous deviation point of v_1 deviates to v_4, and further deviation shown in $ckPT_3$, v_4 will likewise deviate to v_5. And as can be seen in figure 2, this part of the path of $\langle v_1, v_4, v_5\rangle$ is in the general direction of v_1 to v_{15}, so the generated path is more advantageous. It can be seen that the path generated by the deviation point near the end of the path is meaningless: continuously expanding the path cannot find a result, and inserting it in the queue will affect the path that may become the result to be selected preferentially.

To refine the selection of deviation points, particularly near the end of the deviation path. we introduce parameters α and β as constraints. The deviation trimming process is activated when the proportion of nodes that have completed deviation relative to the total number of nodes in the path falls below α. During pruning, nodes that have not yet deviated are considered for deviation. If the relative position of a deviation node within the path (excluding already deviated nodes) exceeds β, the nodes that follow will not undergo deviation. The introduction of α ensures the logical implementation of deviation pruning. Specifically, when a path has been extensively extended, leaving only a small fraction of nodes eligible for further expansion and one or two keywords yet to be satisfied, indiscriminate application of deviation pruning could inadvertently eliminate paths on the verge of qualifying as results. Experimental evaluations suggest optimal settings of $\alpha = 1/3$ and $\beta = 1/2$, indicating that deviation should be limited to the initial half of the

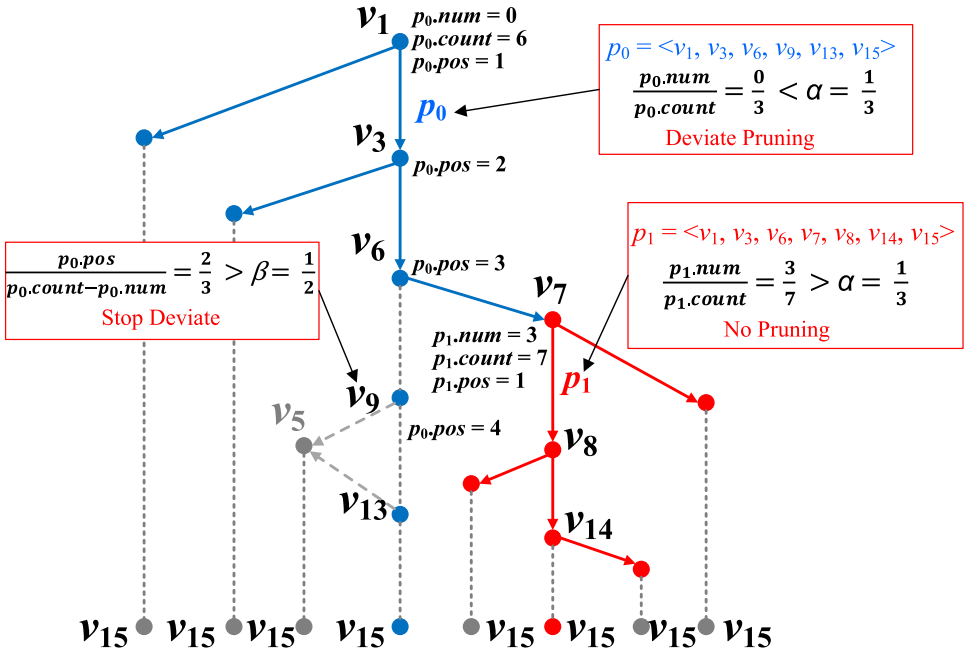

Fig. 11 Example of the Deviation Pruning

remaining nodes when over two-thirds of a path has not been expanded.

Strategy 3 ***(Deviate Pruning Strategy)*** *Consider p_0 as a path recently retrieved from PQ, where $p_0.num$ denotes the number of nodes in p_0 that have already undergone deviation, $p_0.count$ represents the total number of nodes within p_0, and $p_0.pos$ indicates the position of the next node to deviate (starting from the first unexpanded node). The criteria for halting the continuous deviation of p_0 are as follows: if the ratio $p_0.num/p_0.count$ is less than α, and concurrently, $p_0.pos/(p_0.count - p_0.num)$ exceeds β.*

Figure 11 shows the deviation pruning example of $Q(v_1, v_{15}, \{kw_1, kw_2, kw_3\}$. For $p_0 = \langle v_1, v_3, v_6, v_9, v_{13}, v_{15}\rangle$, its total number of nodes is 6, so $p_0.count = 6$. Since it is the first expansion, all the deviation points are unexpanded, thus $p_0.num = 0$ and $p_0.num/p0.count = 0 < \alpha = 1/3$. The deviation pruning is performed here. For v_9, it is the fourth deviation point in p_0, so $p_0.pos = 4$ for v_9, and $p_0.pos/(p_0.count - p_0.num) = 2/3 > \beta = 1/2$, so all deviation points after v_9 will not be deviated. Similarly, for $p_1 = \langle v_1, v_3, v_6, v_7, v_8, v_{14}, v_{15}\rangle$, v_1, v_3, v_6 have complete the deviation, so $p_1.num = 3$. And $p_1.count = 7$, $p_1.num/p_1.count = 3/7 > \alpha$, so the deviation pruning will not be performed.

5.4 Range Optimization Strategy

Both the NN calculation and the heuristic value determination processes necessitate considering the global distribution of required POIs, thereby implicating a vast search space. Nonetheless, POIs situated significantly far from both s and t are unlikely candidates for deviation and thus need not be included in the evaluation. To refine our approach and expedite both NN and heuristic value computations, we propose restricting our search to only those POIs that fall within a predefined proximity to s and t, effectively considering them as potential candidates. This strategy not only accelerates the calculations but also diminishes the size of the candidate path sets, thereby streamlining the path-finding process.

Strategy 4 ***(Range Optimization Strategy)*** *Given a query $Q(s, t, Q_{kw})$, γ denotes the search range parameter and $R = \gamma \times \delta(s, t)$ is the radius of the search range. Let $O'(kw_i)$ denote the set of POIs containing kw_i within the search range, that is $\forall v_j \in O'(kw_i)$, $\delta(s, v_j) + \delta(v_j, t) < R$. If all the required POIs can be found within the range, those outside the range will not be considered. Otherwise, the range will be epanded by increasing γ by 1 until all can be found. That is, if $\exists kw_i \in Q_{kw}$, $O'(kw_i) = \emptyset$, $\gamma = \gamma + 1$.*

In Figure 12, given the $Q_{kw} = \{kw_1, kw_2, kw_3\}$, and initially, the search range is a gray ellipse calculated based on the initial γ. Within this range, $O'(kw_1) = \{v_1, v_2\}$, $O'(kw_3) = \{v_2, v_3\}$, while there are no POIs containing $kw2$, so $O'(kw_2) = \emptyset$. To this end, by increasing γ by 1, the search range is expanded to the red ellipse. Now, $O'(kw_2) = \{v_4, v_5\}$, and v_5 will also be updated to both $O'(kw_1)$ and $O'(kw_3)$. In addition, since v_6 is outside the big red ellipse, it will not be considered for expansion.

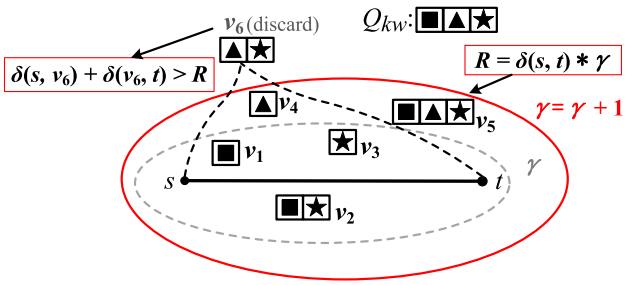

Fig. 12 Example of the Range Optimization

5.5 Lower Bound Optimization Strategy

According to Definition 3, the Sky-ORCSK query takes into account two dimensions, route distance and the total hardness of stops. While in PQ, the sub-paths are arranged in ascending order based on the distance, and the sub-paths with the smallest distance are prioritized for expansion. This expansion cannot be arbitrarily halted, as a path with a longer distance may have the lower total hardness of stops and thus be included in the skyline results. However, expanding all the paths in PQ is time-consuming. Assuming that the lower bound on the total hardness of stops of all resulting paths is S_{lb}, the expansion process can be safely halted if the total hardness of stops of the popped valid path p_i is equal to S_{lb}. This is because the total hardness of stops for each unexpanded path in PQ is guaranteed to be no less than that of p_i, and its distance is longer than p_i's, so it cannot be the result.

S_{lb} can be determined greedily with the help of the inverted index. Firstly, the required POIs can be found by checking the inverted index of each keyword in the given Q_{kw}. Then, in each iteration, we greedily select POIs with the smallest hardness of stop for unsatisfied query keywords, one can identify the combination of POIs that satisfies all keywords in Q_{kw} with the smallest total hardness of stops, thus obtaining S_{lb}.

Strategy 5 ***(Lower Bound Optimization Strategy)*** *Let S_{lb} represent the lower bound of the number of stops for the given Q_{kw}. If the popped path p_i is a valid path, and its total hardness of stops $S(p_i)$ is equal to S_{lb}, the expansion will be halted.*

5.6 *DA-SkyPrune* Algorithm with Optimizations

The algorithm *DA-SkyPrune* employs the foundational paradigm of *DA-Sky*, while seamlessly integrating several optimization strategies previously discussed, as delineated in Algorithm 3. For simplicity, the same process as Algorithm 2 is not re-listed, but is referenced.

First, we initialize S_{ub} and the hash table H, and calculate S_{lb} based on the Strategy 5. Following Strategy 4, required

Algorithm 3: *DA-SkyPrune*(G, s, t, Q_{kw})

Input: $G(V, E)$, $Q\{s, t, Q_{kw}\}$
Output: The Skyline paths set S_p

1 $S_{ub} \leftarrow 0$, $S_{lb} \leftarrow$ ComputeLower(), $H \leftarrow \phi$;
2 Select those POIs in the γ range as candidates;
// Strategy 4
3 Same as lines 1-4 of Algorithm 2;
4 **while** $PQ \neq \phi$ **do**
5 Same as lines 8-9 of Algorithm 2;
6 **if** $S_p.back().stop = S_{lb}$ **then**
7 break; // Strategy 5
8 $S_{ub} \leftarrow S_p.back().stop$;
9 C $\leftarrow$ Get all vertices on the shortest path from v_m to t;
10 Size $\leftarrow v_m.size$ + C.count;
11 **for** *each v_i in C(from the first)* **do**
12 **if** $v_m.size/Size < \alpha$ **then**
13 **if** $v_i.pos/C.count > \beta$ **then**
14 break; // Strategy 3
15 Same as lines 11-15 of Algorithm 2;
16 **if** $S(P_a) \geq S_{ub}$ **then**
17 continue; // Strategy 1
18 **if** $H(NN_{v_i})$ *is not exist* **then**
19 $H(NN_{v_i}).stop \leftarrow S(P_a)$;
20 $H(NN_{v_i}).dist \leftarrow w(P_a)$;
21 **else**
22 **if** $H(NN_{v_i}) \succ P_a$ **then**
23 continue; // Strategy 2
24 **if** $P_a \succ H(v_i)$ **then**
25 $H(NN_{v_i}).stop \leftarrow S(P_a)$;
26 $H(NN_{v_i}).dist \leftarrow w(P_a)$;
27 Same as lines 27-30 of Algorithm 2;
28 **return** S_p;

POIs within the search range R are figured out to serve as candidates (Line 2). Strategy 5 facilitates early termination of the algorithm. If the total hardness of stops for a path newly added to S_p equals S_{lb}, the algorithm may cease prematurely, halting further expansion of PQ (Lines 6-7). Per Strategy 3 guidelines, the deviation process for a path can be halted ahead of schedule based on the limits of parameters α and β (Lines 12-14). According to Strategy 1, if the total hardness of stops in the extended path P_a exceeds S_{ub}, P_a is directly pruned (Lines 15-16) and not enqueued into PQ (Line 17). Within the context of the hash table H, if a path P_a passes an NN_{v_i} not already present in H, the path's total hardness of stops and distance are directly recorded under $H(NN_{v_i})$ (Lines 18-20). Should NN_{v_i} already exist within H, and if $H(NN_{v_i})$'s associated total hardness of stops and path distance dominate P_a, P_a is pruned according to Strategy 2 (Lines 22-23). Conversely, if P_a can dominate the existing record for NN_{v_i} in H, $H(NN_{v_i})$ is updated (Lines 24-26).

Table 3 illustrates the evolution of the priority queue PQ for the query $Q(v_1, v_{15}, \{kw_1, kw_2, kw_3\})$ in the road network of Figure 1. Each partial path is assigned a unique

Table 3 $Q(v_1, v_{15}, \{kw_1, kw_2, kw_3\})$ in PQ

Step	Paths ($\langle route\rangle$ (lb), $binary$,$stops$)
1	① $(\langle s\rangle(15), 000, 0)$
2	②$(\langle s, v_4\rangle(19), 001, 3)$,
	③$(\langle s, v_3, v_2\rangle(25), 101, 2) \Rightarrow (\langle s, v_2\rangle(22), 101, 2)$
	④$(\langle s, v_3, v_6, v_7\rangle(27), 011, 3)$
3	⑤$(\langle s, v_4, v_5\rangle(19), 011, 5)$,③,
	⑥$(\langle s, v_4, v_6, v_7\rangle(23), 111, 6)$, ④
4	⑦$(\langle s, v_4, v_5, v_9\rangle(19), 111, 6)$, ③,
	⑧$(\langle s, v_4, v_5, v_{10}, v_{11}\rangle(26), 111, 9)$,④,
	~~⑨ $(\langle s, v_4, v_5, v_{10}, v_{12}, v_9\rangle(25), 111, 6)$~~
5	⑦→ $(\langle s, v_4, v_5, v_9, v_{13}, t\rangle(19), 111, 6)$, ③, ⑥, ⑧, ④
6	⑩$(\langle s, v_2, v_7\rangle(22), 111, 5)$, ⑥, ⑧, ④
	~~⑪ $(\langle s, v_2, v_6, v_7\rangle(24), 111, 5)$~~
7	⑩→ $(\langle s, v_2, v_7, v_8, v_{14}, t\rangle(22), 111, 5)$, ⑥, ⑧, ④
8	~~⑥$(\langle s, v_4, v_6, v_7\rangle(23), 111, 6)$~~ ⑧, ④
9	⑧→ $(\langle s, v_4, v_5, v_{10}, v_{11}, t\rangle(26), 111, 4)$,④
10	~~④$(\langle s, v_3, v_6, v_7\rangle(27), 011, 1)$~~

identifier. At step 1, the shortest path from s to t is enqueued into PQ, and then it is dequeued for deviation. According to Strategy 3, only nodes in the first half are considered for deviation, resulting in the generation of three new partial paths (step 2). Note that p_3 is refined by excluding v_3, and then updated. Next, p_2 is dequeued and generates p_5 and p_6, and then p_5 is dequeued and generates p_7, p_8, and p_9. p_9 is pruned due to NN pruning, as it and p_7 both deviate towards node v_9, but path 7, being generated first, establishes the cost record for $NN(v_9)$. When p_9, later generated, fails to offer a competitive cost, it is pruned. Continuing to step 5, p_7 is moved to the result set S_p. Step 6 involves the dequeuing of p_3 and the generation of p_{10} and p_{11}, both deviating towards v_7, with p_{11} being pruned as per the NN pruning strategy. p_{10} is then dequeued and added to S_p at step 7, prompting an update to S_{ub}. By step 8, p_6 is pruned due to exceeding S_{ub}, and at step 9, p_8 is added to S_p, having found the path with the minimum number of stops. Leveraging the lower bound optimization strategy, further expansion of PQ is halted, and the remaining p_4 in the queue is disregarded.

6 Experimental evaluation

In this section, we evaluate our proposed algorithm by comparing it with several state-of-the-art solutions. We first introduce the details of the experimental setting, followed by the performance comparison.

6.1 Experimental setup

Datasets: We selected four real-world road networks from [1, 2] and their corresponding POI data and the average number of keywords with POIs from [2, 3], as summarized in Table 4. Among these, the medium-sized New York (NY) map was chosen as the default map for the experiments. The experimental parameters, along with their default settings (highlighted in bold), are provided in Table 5. Specifically, POI density is defined as the ratio between the number of POIs and the number of nodes. The number of keyword types in POIs is proportional to the number of POIs. The query OD distance, Q_{dist}, is segmented based on percentages of the maximum diameter of the road network. For each POI, the hardness of the stop range is from 1 to h_{max}. Based on parking evaluations from real-world scenarios [47], h_{max} is assigned to each POI following a normal distribution. When the number of POIs is varied through other parameters, such as varying POI density, the locations of each POI are first generated using a uniform distribution and then mapped to their nearest vertex.

Sky-ORCSK Query Workload: For the *NY* and *BJ* road networks, the query workloads are derived from real trajectory data [4, 5]. Specifically, we randomly select 1,000 trajectories and use their starting and ending points as the Sky-ORCSK query origins and destinations, respectively. For query keywords, we assign them to the nearest POIs based on the dwell time at trajectory points. For the *CAL* and *COL* road networks, due to the lack of publicly available datasets, we reuse the query data provided in our previous *ORCSK* [26]. Both query locations and query keywords in *CAL* and *COL* are generated randomly.

Evaluation Criteria: The performance is evaluated based on query time, distance, and dominance count, which represent the query's response time, the total distance of the resulting paths, and the number of paths where two algorithms dominate each other, respectively. If a query exceeds a runtime of one hour or encounters memory limitations, its query time is recorded as *INF*. All reported results are averaged over 1,000 queries.

Methods: We selected the following six algorithms to compare their query efficiency and result quality. There are as follows:

- **IG-Enum-Sky**: After obtaining the candidate set by IG-Tree, it enumerates all possible POI combinations, constructs paths using a greedy strategy, and finally builds the skyline set based on dominance relationships.
- **IG-Sky**: It uses an efficient POI replacement strategy to explore a range of POI combinations for generating skyline paths and computes the shortest distance using Dijkstra's algorithm.

Table 4 Real World Maps

Maps	|V|	|E|	POI	$\|K(V)\|$	Average Number of Keyword	Total Available Keywords
CAL	21K	43K	12,197	63	7	87,379
NY	**264K**	**733K**	**20,984**	427	10	213,840
BJ	296K	774K	25,187	511	8	227,496
COL	435K	1,057K	32,179	548	9	301,611

Table 5 Experiment Parameters

Parameter	Value
$\|Q_{kw}\|$	2, 4, **6**, 8, 10
Density	5%, **10%**, 20%, 30%, 50%
$\|K(V)\|$	**3%**, 6%, 9%, 12%, 15%
Q_{Dist}	0-10%, **10-20%**, 20-30%, 30-40%, 40-50%
Distribution	uniform, gauss, Zipfian
h_{max}	**5**, 6, 7, 8, 9, 10
α	1/5, 1/4, **1/3**, 1/2, 2/3
β	1/5, 1/4, 1/3, **1/2**, 2/3
γ	2, **3**, 4, 5

- **IG*-Sky**: which leverages the H2H index [44] to accelerate shortest distance calculations for IG-Sky.
- **DA-Prune** [26]: The original versions of DA-Sky and DA-SkyPrune iteratively construct top-k shortest paths that satisfy the query keywords, until the complete set of skyline paths is covered.
- **DA-Sky**: Path deviation based method in Algorithms 2.
- **DA-SkyPrune**: Path deviation based method with five optimization strategies in Algorithms 3.

Next, to assess the approximation quality of our two algorithms relative to the exact solution. We select the following exact algorithm for answer Sky-ORCSK.

- **KSP** [42]: Blindly constructing top-k shortest paths without considering query keywords, until the current top-k paths cover the entire set of skyline paths. Although KSP can find exact solutions on small road networks, it still suffers from very long query times.

Implementation Details: The algorithms under study were implemented in C++ and compiled using GNN GCC with full optimization flags enabled. All experiments were conducted on an Intel(R) Xeon(R) W-2245 CPU @ 3.2GHz with 250GB of RAM, running Linux (Ubuntu 18.04 LTS, 64-bit) to ensure a robust and consistent testing environment.

6.2 Performance Comparison

We compare the performance of the algorithms by varying parameters as listed in Table 5.

Effect of $|Q_{kw}|$: Figure 13 illustrates the impact of the number of query keywords $|Q_{kw}|$ on algorithm performance across three maps. In terms of query time, IG-Enum-Sky is the worst, followed by IG-Sky, DA-Prune and DA-Sky, while our DA-SkyPrune and IG*-Sky are the best. IG-Enum-Sky constructs the skyline set by exhaustively enumerating all POI combinations, and is therefore most affected by the value of Q_{kw}. In contrast, IG-Sky reduces query time effectively through a POI replacement strategy, but its performance is still limited by the overhead of Dijkstra's algorithm. DA-Prune blindly explores top-k paths without considering skyline constraints, which leads to retrieving more paths than necessary to construct the skyline set, resulting in higher query time. DA-Sky improves upon DA-Prune by incorporating skyline constraints to terminate the blind exploration of paths earlier, thereby improving query efficiency. Our IG*-Sky significantly improves IG-based methods by integrating the H2H index. Specifically, IG*-Sky is 2 to 3 orders of magnitude faster on average than IG-Enum-Sky and IG-Sky, and this advantage becomes even more pronounced with larger road networks. For DA-SkyPrune, it clearly outperforms DA-Prune and DA-Sky by approximately 1 to 2 orders of magnitude. Between the IG*-Sky and DA-SkyPrune, IG*-Sky is slightly faster than DA-SkyPrune. This indicates that, in terms of query efficiency, candidate-set-based methods outperform path-expansion-based methods.

In terms of result quality, since IG-Sky and IG*-Sky differ only in their shortest path computation methods, yielding identical result paths, the path distances for IG-Sky are not separately listed in Figure 13(b),(e),(h) and (f). Specifically, DA-based algorithms are on average 20–30% shorter than IG-based algorithms. This advantage becomes more pronounced with larger maps and higher keyword counts. The disparity arises because IG-based methods, using the IG-tree, efficiently identify keyword combinations but disregard the spatial and orientation relationships among POIs. When adjacent POIs are not aligned along the shortest path from s to t, the detour distance increases. Conversely, DA-based methods employ a deviation method based on points

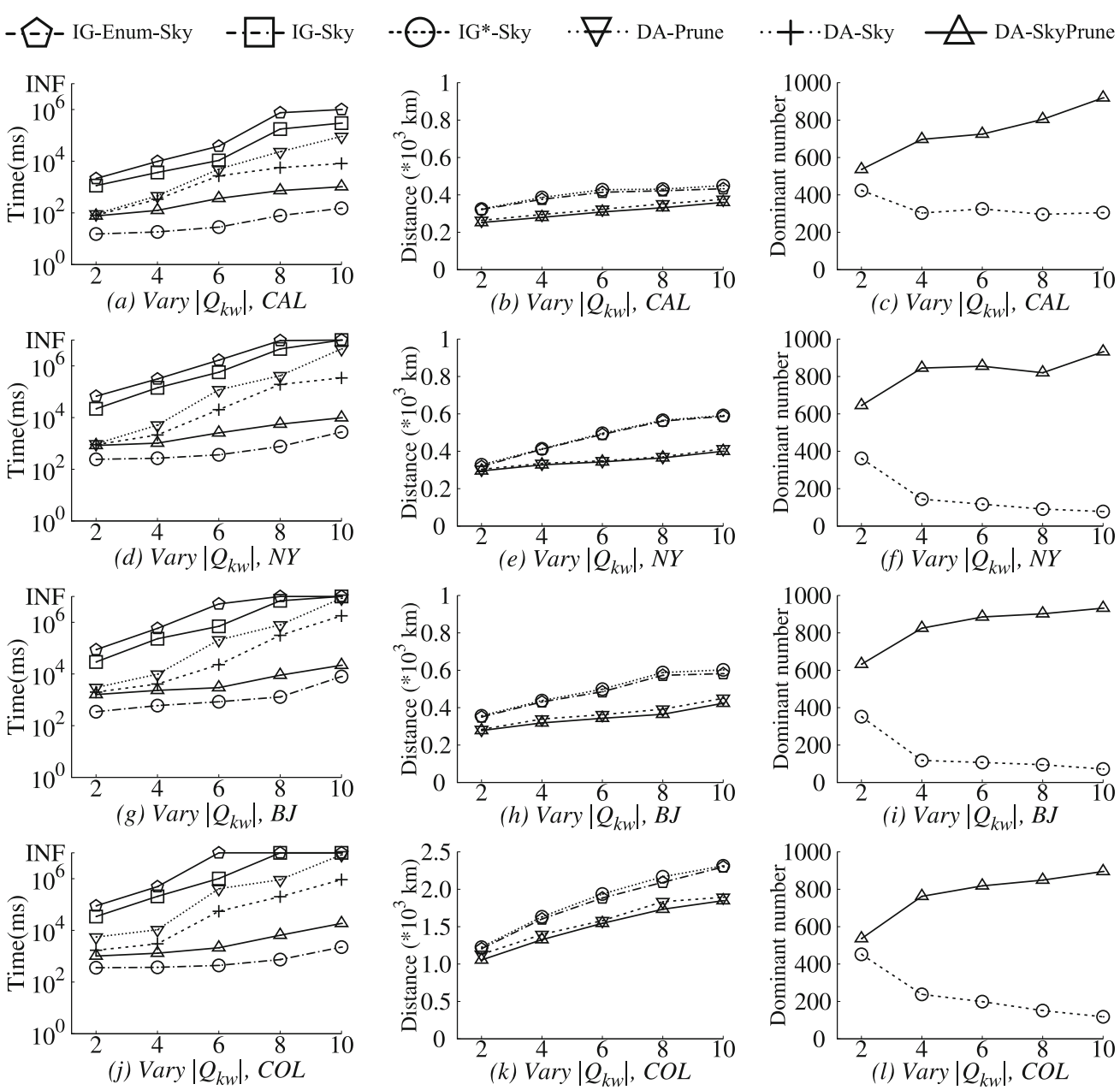

Fig. 13 Performance V.S. Q_{kw} and *Maps*

along the shortest path, consistently using path distance as a heuristic to optimize results. A closer look shows that IG-Enum-Sky performs slightly better than IG*-Sky, as it enumerates all possible combinations. However, this advantage is very minimal, and overall, IG*-Sky is the better choice for IG-based methods. For DA-based methods, DA-SkyPrune slightly outperforms DA-Prune. This is because DA-Prune blindly constructs the skyline by considering only the distance dimension, resulting in longer paths. In contrast, DA-SkyPrune continuously considers skyline constraints during expansion, which effectively improves result quality.

Finally, by analyzing the mutual dominance between the result sets of IG*-Sky and DA-SkyPrune, we observe that as the number of query keywords increases, DA-SkyPrune dominates most of the queries. This indicates that DA-SkyPrune has a greater advantage when handling a larger number of query keywords.

In summary, our algorithms IG*-Sky and DA-SkyPrune significantly improve the query efficiency of existing meth-

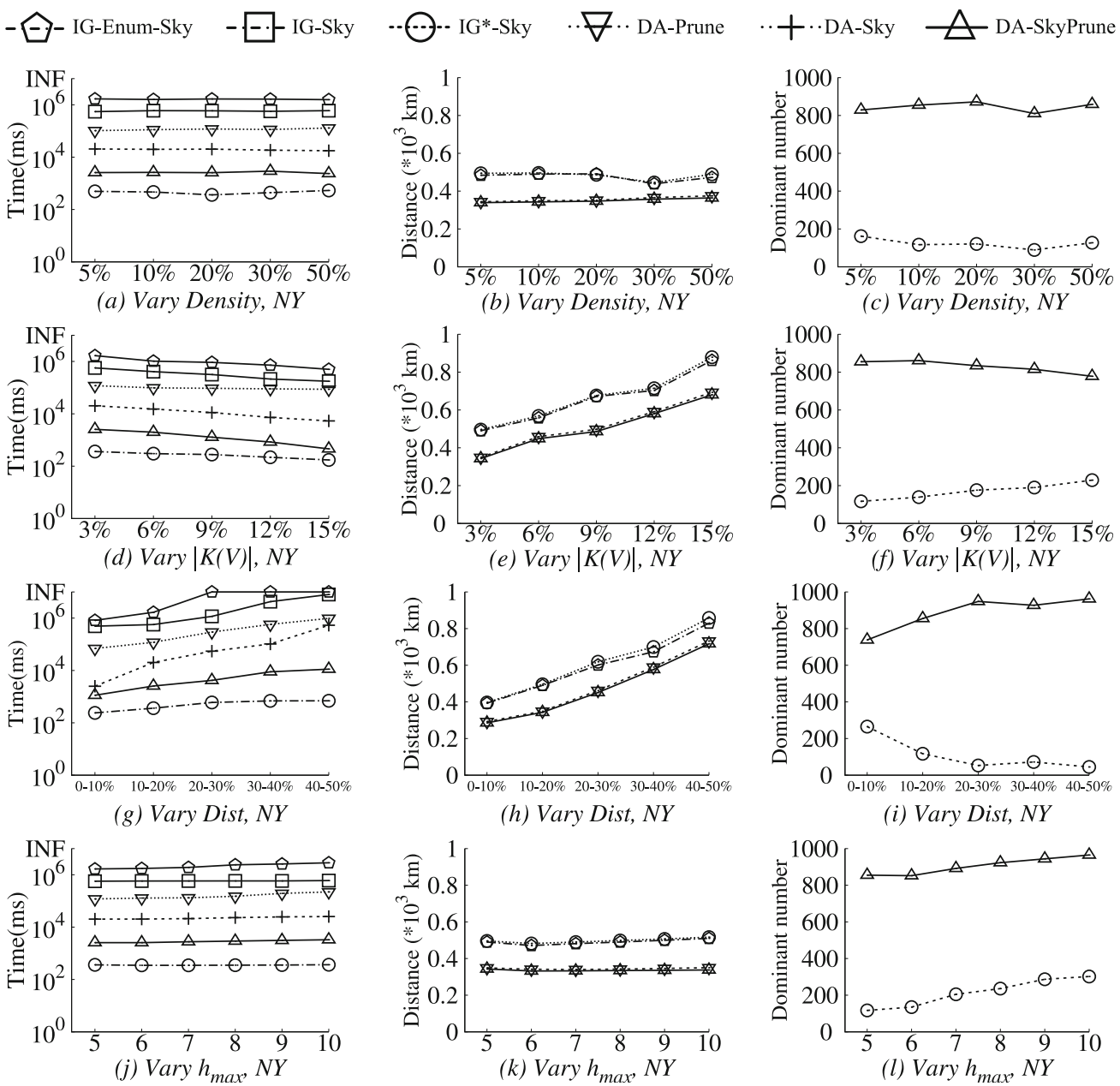

Fig. 14 Performance V.S. *Density*, $|K(V)|$, *Dist* and h_{max}

ods, while achieving a better balance between query time and result quality.

Effect of Density: Figure 14(a)-(c) illustrate the impact of POI density on performance under the NY map. The results indicate that performance is not significantly affected by POI density because the number of keyword types is proportional to the number of POIs. Consequently, the keyword density remains constant, resulting in minimal impact on search efficiency. This observation shows that, compared to POI density, keyword density plays a more critical role in determining algorithm performance.

Effect of $|K(V)|$: Figure 14(d)-(f) illustrates the impact of the number of union keyword types in G on performance. The influence of an increasing variety of keyword types on query performance is twofold: while query times decrease, path lengths increase. As $|K(V)|$ grows, the number of POIs associated with each individual keyword type decreases, given that the total number of POIs remains fixed. This reduction in per-keyword POI count lowers the variety of feasible POI combinations, leading to fewer candidate paths and, consequently, shorter query times. However, the decreased frequency and greater dispersion of POIs associated with

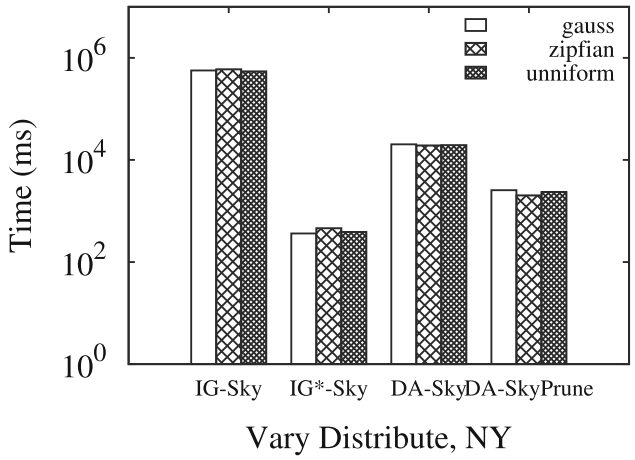

Fig. 15 POI distribution

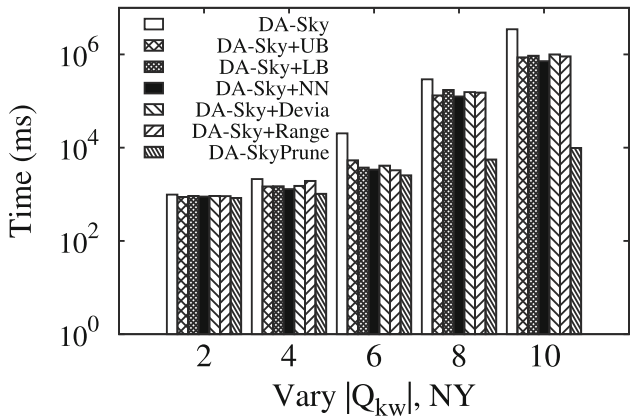

Fig. 16 Strategy Improve

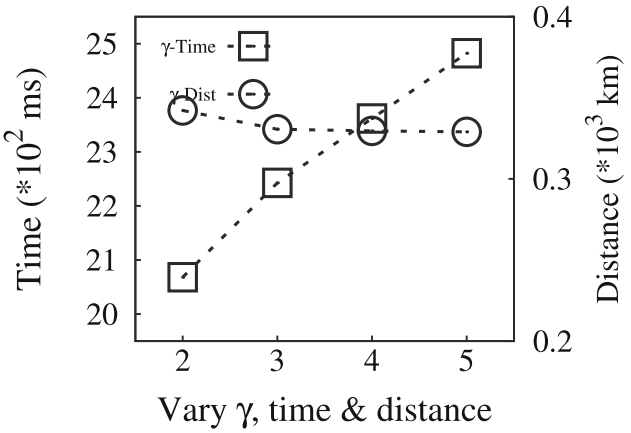

Fig. 17 Strategy Improve

each keyword result in increased distances between them, leading to longer overall path lengths.

Effect of Q_{Dist}: Figure 14(g)-(i) illustrates the impact of the distance between s and d on performance, with the horizontal axis representing the ratio of the distance to the map diameter. The query time of all algorithms increases as the distance grows, due to the expansion of the search area and the corresponding increase in computation. IG*-Sky and DA-SkyPrune exhibit slower increases in query time. This is attributed to the H2H index, which enhances the shortest distance computation efficiency in IG*-Sky, and the multiple pruning strategies in DA-SkyPrune, which improve the efficiency of identifying potential points of interest, allowing both algorithms to maintain strong performance even at long distances. Although the distance gap between the results returned by the two algorithms does not widen with increasing distance, DA-SkyPrune dominates more results compared to IG*-Sky, demonstrating a clearer advantage in result quality at greater distances.

Effect of h_{max}: Figure 14(j)-(l) illustrates the impact of the maximum hardness of stop h_{max} on performance, with the horizontal axis representing the maximum hardness of stop for POIs. As h_{max} increases, the query time of all algorithms gradually increases as well. This also leads to a gradual increase in the distance of result paths. This is because a larger h_{max} enlarged the skyline solution space, resulting in more paths potentially being included in the result set. However, the increase in query time is negligible, indicating that both IG-Sky and DA-SkyPrune can efficiently handle larger h_{max} and exhibit good scalability.

Effect of POI Distribution: Figure 15 presents the results of our investigation into the impact of various POI distributions on the runtime of the algorithms on the NY map. Specifically, we tested POI points fixed according to Gaussian, Zipfian, and uniform distributions. The experiments revealed a modest increase in query times for IG-Sky and IG*-Sky under the Zipfian distribution, while DA-Sky and DA-SkyPrune exhibited a slight deceleration under the Gaussian distribution. Despite these minor variations, all algorithms demonstrated negligible differences across the different distributions. This result indicates the high stability of the algorithms, confirming their robustness and applicability to diverse map distributions while maintaining consistent query performance.

Effect of Each Optimization Strategies: The efficiency of various pruning strategies was evaluated in the context of different keyword quantities, using the default parameter settings. Figure 16 presents the results of strategy improve. Starting from the baseline DA-Sky algorithm, we incrementally integrated boundary pruning (both lower bound (LB) and upper bound (UB)), nearest neighbor pruning (NN), deviation pruning (Devia), and range pruning (Range). Generally, the efficacy of pruning strategies becomes more pronounced as the number of query keywords increases, since a larger pool of candidate POIs amplifies the potential for pruning. Among these strategies, LB pruning demonstrated relatively modest improvements. This is because, by the time the number of stops approaches the lower bound, a significant portion

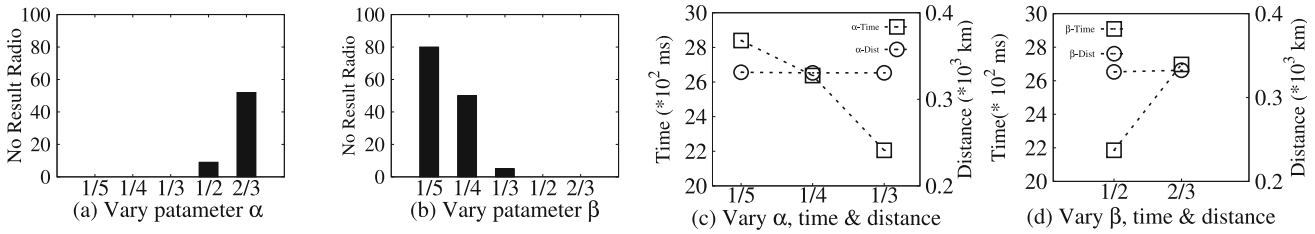

Fig. 18 Change of parameter α and β, NY.

of paths in the queue have already been expanded, leaving only a small fraction subject to pruning. Nevertheless, with ten keywords, LB pruning still improved performance by approximately 30%. NN pruning emerged as the most effective strategy, leveraging the cost records of NN nodes to preemptively select the optimal path for extension towards the same NN. This approach eliminates the need for further iterations, significantly reducing the number of paths. Particularly for queries with numerous keywords, NN pruning was observed to double query time efficiency. The efficacy of Range and Devia pruning was comparable, offering a 40–50% performance boost as the keyword count increased. This similarity in efficiency arises from their analogous pruning principles: Range pruning restricts the candidate POI pool by limiting the search range, while Devia pruning reduces the number of nodes subject to deviation. Together, they aim to curtail futile expansions, diminish redundant path generation, and reduce queue sizes. The UB pruning strategy also demonstrated notable effectiveness, yielding a 30–40% improvement as the number of query keywords increased. When all pruning strategies are applied concurrently, the DA-SkyPrune algorithm achieves nearly an order of magnitude faster performance compared to DA-Sky, with the disparity growing in scenarios involving a higher number of query keywords.

Parameter γ: Figure 17 shows the impact of varying the range optimization parameter γ on the query results. As γ increases, the query time increases significantly. This is because a larger γ results in a larger search space, thereby including more POIs in the computation. Conversely, the quality of the results improves with a broader search range. However, this trend becomes less pronounced when $\gamma > 3$. This indicates that the optimal results can be achieved within a search space of 3 times $\delta(s, t)$, and extending the range beyond this point provides no significant benefit while incurring unnecessary time costs.

Parameters α **and** β: The impact of parameters α and β on query time and result quality is illustrated in Figure 18. As shown in Figures 18(a)-(b), setting α and β outside their optimal ranges can result in the absence of any results. This issue arises when the pruning conditions are too lenient, causing the premature elimination of all candidate paths and, consequently, the failure to identify any solutions. The analysis suggests that α should not exceed $1/3$ and β should not be less than $1/2$ to avoid such outcomes. Further investigation into the correlation between these parameters and algorithm performance, depicted in Figures 18(c)-(d), shows that the average path length remains relatively stable across various parameter values. However, when α is set to $1/2$ and β to $1/3$, the query time significantly decreases, achieving a twofold improvement compared to other parameter settings. Based on these findings, we establish $\alpha = 1/2$ and $\beta = 1/3$ as the optimal parameter values, balancing efficiency with the preservation of result quality.

Skyline Set Approximation Quality: Finally, we evaluate the approximation quality of each algorithm with respect to the exact solution in Table 6. The experiment is only conducted in the smallest CAL network because it is the only one in which the exact KSP solution can run, and we need its exact result as the ground truth to evaluate the quality of the proposed methods. Even so, *KSP* still fails to return the exact results when Q_{kw} is larger than 6, which proves the necessity of the efficient approximate solutions. Therefore, for the last rows of the hard queries, we cannot provide the quality results, but it can still demonstrate the high efficiency of the proposed methods.

Secondly, the Top-k column shows the number of paths tested by different algorithms to construct the skyline set when varying Q_{kw}. Because KSP and DA-Prune construct the skyline set by generating top-k paths, we record the value of k required to obtain the complete skyline set. In contrast, IG*-Sky and DA-SkyPrune incrementally construct the skyline set based on dominance relationships. Therefore, we record the total number of feasible paths considered during Dominance Checking (DC). As Q_{kw} increases, the number of top-k paths that KSP and DA-Prune need to retrieve in order to construct the skyline set grows rapidly. Notably, when $Q_{kw} = 8$ and 10, KSP retrieved more than 10,000 paths but failed to obtain the complete skyline set within one hour, rendering its query invalid. This clearly validates that KSP is unable to answer Sky-ORCSK queries on large-scale road networks. Similarly, the DA-Prune algorithm also needs to retrieve several thousand paths to construct an approximate skyline set. However, since DA-Prune takes keywords into account, the number of paths it retrieves is significantly lower than that of KSP. For IG*-Sky, because it constructs candidate

Table 6 The Number of Paths Required by Constructing the Skyline Set

Q_{kw}	Methods															
	KSP			DA-Prune				IG*-Sky				DA-SkyPrune				
	Top-k	Skyline	Time (ms)	Top-k	Skyline	Time (ms)	AR	Recall	DC	Skyline	Time (ms)	AR	Recall	DC	Skyline	Time (ms)	AR	Recall
2	632	3	11348	7	3	104.657	1.149	87.2%	122	3	15.24	1.69	56.1%	4	3	76.162	1.07	91.5%
4	3491	9	251,891	69	9	491.054	1.149	87.1%	491	7	18.317	1.96	54.7%	13	9	125.079	1.11	93.4%
6	7716	15	1,023,487	529	15	5959.541	1.651	83.4%	876	14	28.185	2.04	52.1%	29	14	362.494	1.33	94.7%
8	>10,000	**overtime**	INF	3470	18	8012.457	INF	INF	2493	16	78.526	INF	INF	37	18	729.091	INF	INF
10	>10,000	**overtime**	INF	7791	27	15871.098	INF	INF	7409	22	151.164	INF	INF	75	25	1026.342	INF	INF

DC is the number of Dominance Checking. AR is the Approximation Ratio of distance compared with the exact result obtained from KSP. Recall is the percentage of skyline paths that include the exact solution path. Recall is the proportion of exact skyline paths that appear in the skyline set produced by each algorithm

paths through POI replacement, the number of paths in dominance checking increases significantly as Q_{kw} increases. DA-SkyPrune is the best, which reduces the number of paths checked by an average of 1 to 2 orders of magnitude compared to DA-Prune. This benefit is attributed to the various optimization strategies we proposed, which effectively prune a large number of candidate paths during the expansion process.

In terms of path quality, we first compare the Approximation Ratio (AR), which is the average path length divided by the exact optimal path length obtained from KSP. All algorithms show a gradual increase as Q_{kw} increases. This indicates that a larger skyline space can further lead to a loss in accuracy. In particular, IG*-Sky exhibits the worst approximation ratio, especially when $Q_{kw} = 6$, where the ratio exceeds 2. The approximation ratio of DA-Prune is lower than that of IG*-Sky, further demonstrating the effectiveness of deviation-based expansion in improving result quality. DA-SkyPrune achieves the lowest approximation ratio, with result quality most comparable to that of the exact algorithm KSP. Then we compare the Recall, which is the ratio of the optimal skyline paths identified compared with the exact result. Again, DA-SkyPrune is the best with the recall always higher than 90%. This indicates that, although our algorithm is based on greedy expansion, the deviation-driven core strategy, together with many optimization strategies, effectively ensures the quality of the results.

7 Conclusion

This paper delves into the Skyline Optimal Route with Collective Spatial Keyword (Sky-ORCSK) problem, showcasing the development and evaluation of the DA-Sky and DA-SkyPrune algorithms. Addressing the need for routing solutions that cater to specific user preferences and stop hardness beyond shortest paths, our work introduces a refined approach to this multi-dimensional keyword-aware routing. Despite the inherent challenge of the NP-Hard problem, our proposed algorithms demonstrate enhanced efficiency and effectiveness, validated through experiments with real-world data. By integrating innovative pruning strategies and spatial indexing, we offer significant improvements in route selection, emphasizing the balance between route efficiency and the satisfaction of user-specified needs. The experiments also confirmed the robustness of our algorithms across different POI densities and distributions, indicating their suitability for diverse urban environments.

Acknowledgements The research work was partially supported by the Key Research and Development Program of Liaoning Province under Grant No.2023JH26/10300022, the Shenyang Young and Middle-aged Scientific and Technological Innovation Talent Support Plan under Grant No.RC220504, Natural Science Foundation of China #62202116,

Guangzhou?HKUST(GZ) Joint Funding Scheme #2023A03J0135, Guangzhou Basic and Applied Basic Research Scheme #2024A04J4455, Guangdong-Hong Kong Technology Innovation Joint Funding #2024A0505040012, Hong Kong Research Grants Council grant# 16202722, and partially conducted in the JC STEM Lab of Data Science Foundations funded by The Hong Kong Jockey Club Charities Trust.

Funding Open access funding provided by Hong Kong University of Science and Technology

References

1. 9th dimacs implementation challenge - shortest paths. http://users.diag.uniroma1.it/challenge9/download.shtml
2. Openstreetmap. https://www.openstreetmap.org/
3. Real datasets for spatial databases: Road networks and points of interest. http://www.cs.utah.edu/~lifeifei/SpatialDataset.htm
4. Real query data in bj. https://www.didiglobal.com//
5. Real query data in ny. https://www.kaggle.com/competitions/nyc-taxi-trip-duration/data/
6. Aljubayrin, S., He, Z., Zhang, R.: Skyline trips of multiple pois categories. In: DASFAA (2015)
7. Cao, X., Chen, L., Cong, G., Xiao, X.: Keyword-aware optimal route search. VLDB **5**(11), 1136–1147 (2012)
8. Chang, L., Lin, X., Qin, L., Yu, J.X., Pei, J.: Efficiently computing top-k shortest path join. In: EDBT (2015)
9. Chondrogiannis, T., Bouros, P., Gamper, J., Leser, U., Blumenthal, D.B.: Finding k-shortest paths with limited overlap. VLDB J. **29**(5), 1023–1047 (2020)
10. Deng, K., Zhou, X., Shen, H.T.: Multi-source skyline query processing in road networks. pp. 796–805. IEEE (2007)
11. Felipe, I.D., Hristidis, V., Rishe, N.: Keyword search on spatial databases. In: ICDE (2008)
12. Gao, J., Qiu, H., Jiang, X., Wang, T., Yang, D.: Fast top-k simple shortest paths discovery in graphs. In: CIKM, pp. 509–518 (2010)
13. Gong, Q., Cao, H., Nagarkar, P.: Skyline queries constrained by multi-cost transportation networks. In: 2019 IEEE 35th International Conference on Data Engineering (ICDE), pp. 926–937. IEEE (2019)
14. Haldar, N.A.H., Li, J., Ali, M.E., Cai, T., Chen, Y., Sellis, T., Reynolds, M.: Top-k socio-spatial co-engaged location selection for social users. IEEE Trans. Knowl. Data Eng. **35**(5), 5325–5340 (2022)
15. Haldar, N.A.H., Li, J., Reynolds, M., Sellis, T., Yu, J.X.: Location prediction in large-scale social networks: an in-depth benchmarking study. VLDB J. **28**(6), 623–648 (2019)
16. Haryanto, A.A., Islam, M., Taniar, D., Cheema, M.A., et al.: Ig-tree: an efficient spatial keyword index for planning best path queries on road networks. World Wide Web **22**(4), 1359–1399 (2019)
17. Haryanto, A.A., Islam, M.S., Taniar, D., Cheema, M.A.: Ig-tree: an efficient spatial keyword index for planning best path queries on road networks. World Wide Web **22**, 1359–1399 (2019)
18. Hershberger, J., Maxel, M., Suri, S.: Finding the k shortest simple paths: A new algorithm and its implementation. ACM Trans. Algorithms **3**(4), (2007)
19. Huang, X., Jensen, C.S.: In-route skyline querying for location-based services. In: W2GIS, pp. 120–135. Springer (2004)
20. Jin, Y.Y.: Finding the k shortest loopless paths in a network. Manage. Sci. **17**(11), 712–716 (1971)
21. Karypis, G., Kumar, V.: Multilevel algorithms for multi-constraint graph partitioning. In: SC'98: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing, pp. 28–28. IEEE (1998)
22. Kossmann, D., Ramsak, F., Rost, S.: Shooting stars in the sky: An online algorithm for skyline queries. In: VLDB (2002)
23. Kriegel, H.P., Renz, M., Schubert, M.: Route skyline queries: A multi-preference path planning approach. In: 2010 IEEE 26th international conference on data engineering (ICDE 2010), pp. 261–272. IEEE (2010)
24. Li, J., Hu, J., Li, L., Xiong, X., Xia, X.: Efficient multi-request route planning on road network. In: ISPA, pp. 617–624. IEEE (2020)
25. Li, J., Wang, Z., Zhang, Y., Zhao, L., Li, L., Zong, C.: Continuously monitoring optimal routes with collective spatial keywords on road networks. In: UIC, pp. 41–48. IEEE (2022)
26. Li, J., Xiong, X., Li, L., He, D., Zong, C., Zhou, X.: Finding top-k optimal routes with collective spatial keywords on road networks. In: ICDE, pp. 368–380. IEEE (2023)
27. Li, J., Yang, Y.D., Mamoulis, N.: Optimal route queries with arbitrary order constraints. TKDE (2013)
28. Li, K., Rao, X., Pang, X., Chen, L., Fan, S.: Route search and planning: a survey. Big Data Research **26**, 100246 (2021)
29. Li, L., Hua, W., Du, X., Zhou, X.: Minimal on-road time route scheduling on time-dependent graphs. VLDB **10**, 1274–1285 (2017)
30. Li, L., Wang, S.: Fastest path query answering using time-dependent hop-labeling in road network. TKDE (2020)
31. Li, L., Wang, S., Zhou, X.: Time-dependent hop labeling on road network. In: ICDE, pp. 902–913. IEEE (2019)
32. Li, L., Zheng, K., Wang, S., Hua, W., Zhou, X.: Go slow to go fast: minimal on-road time route scheduling with parking facilities using historical trajectory. VLDB J. **27**, 321–345 (2018)
33. Li, Y., Yuan, Y., Wang, Y., Lian, X., Ma, Y., Wang, G.: Distributed multimodal path queries. TKDE **34**(7), 3196–3210 (2020)
34. Lin, S.: An effective heuristic algorithm for the traveling salesman problem. Annals of Ops. Res **21**, 498–516 (1973)
35. Liu, H., Jin, C., Yang, B., Zhou, A.: Finding top-k shortest paths with diversity. TKDE **30**(3), 488–502 (2017)
36. Liu, H., Jin, C., Yang, B., Zhou, A.: Finding top-k optimal sequenced routes. In: ICDE, pp. 569–580. IEEE Computer Society (2018)
37. Liu, H., Tong, Y., Zhang, P., Lu, X., Duan, J., Xiong, H.: Hydra: A personalized and context-aware multi-modal transportation recommendation system. In: SIGMOD, pp. 2314–2324 (2019)
38. Liu, Z., Li, L., Zhang, M., Hua, W., Chao, P., Zhou, X.: Efficient constrained shortest path query answering with forest hop labeling. In: ICDE, pp. 1763–1774. IEEE (2021)
39. Liu, Z., Li, L., Zhang, M., Hua, W., Zhou, X.: Fhl-cube: multi-constraint shortest path querying with flexible combination of constraints. VLDB **15**(11), 3112–3125 (2022)
40. Liu, Z., Li, L., Zhang, M., Hua, W., Zhou, X.: Multi-constraint shortest path using forest hop labeling. VLDB J. **32**, 595–621 (2023)
41. Lu, H.C., Chen, H.S., Tseng, V.S.: An efficient framework for multirequest route planning in urban environments. TITS (2016)
42. Luo, Z., Li, L., Zhang, M., Hua, W., Xu, Y., Zhou, X.: Diversified top-k route planning in road network. VLDB **15**, 3199–3212 (2022)

43. Mouratidis, K., Lin, Y., Yiu, M.L.: Preference queries in large multi-cost transportation networks. In: 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010), pp. 533–544. IEEE (2010)
44. Ouyang, D., Qin, L., Chang, L., Lin, X., Zhang, Y., Zhu, Q.: When hierarchy meets 2-hop-labeling: Efficient shortest distance queries on road networks. In: SIGMOD, pp. 709–724. ACM (2018)
45. Papadias, D., Zhang, J., Mamoulis, N., Tao, Y.: Query processing in spatial network databases. In: VLDB, pp. 802–813 (2003)
46. Peng, Y., Ma, Z., Zhang, W., Lin, X., Zhang, Y., Chen, X.: Efficiently answering quality constrained shortest distance queries in large graphs. In: ICDE, pp. 856–868. IEEE (2023)
47. Sahasrabudhe, S., Dev, M.: Assessment of parking policy in case of tier ii cities in india: Case of aurangabad, maharashtra. In: Transport Research Arena Conference, pp. 702–709. Springer (2024)
48. Salgado, C.: Keyword-aware skyline routes search in indoor venues. In: SIGSPATIAL (2018)
49. Sharifzadeh, M., Kolahdouzan, M., Shahabi, C.: The optimal sequenced route query. VLDB J. **17**(4), 765–787 (2008)
50. Sharifzadeh, M., Shahabi, C.: The spatial skyline queries. In: VLDB (2006)
51. Wang, L., Wong, R.C.W.: Qhl: a fast algorithm for exact constrained shortest path search on road networks. SIGMOD **1**, 1–25 (2023)
52. Wang, S., Xiao, X., Yang, Y., Lin, W.: Effective indexing for approximate constrained shortest path queries on large road networks. Proceedings of the VLDB Endowment **10**(2) (2016)
53. Yu, Z., Yu, X., Koudas, N., Chen, Y., Liu, Y.: A distributed solution for efficient k shortest paths computation over dynamic road networks. TKDE (2024)
54. Yuan, Y., Lian, X., Chen, L., Yu, J.X., Wang, G., Sun, Y.: Keyword search over distributed graphs with compressed signature. TKDE **29**, 1212–1225 (2017)
55. Yuan, Y., Lian, X., Wang, G., Chen, L., Ma, Y., Wang, Y.: Weight-constrained route planning over time-dependent graphs. In: ICDE, pp. 914–925. IEEE (2019)
56. Zhang, M., Li, L., Hua, W., Mao, R., Chao, P., Zhou, X.: Dynamic hub labeling for road networks. In: ICDE, pp. 336–347. IEEE (2021)
57. Zhang, M., Li, L., Hua, W., Zhou, X.: Efficient 2-hop labeling maintenance in dynamic small-world networks. In: ICDE, pp. 133–144. IEEE (2021)
58. Zhang, M., Li, L., Zhou, X.: An experimental evaluation and guideline for path finding in weighted dynamic network. VLDB **14**(11), 2127–2140 (2021)
59. Zhang, U., Yuan, L., Li, W., Qin, L., Zhang, Y.: Efficient label-constrained shortest path queries on road networks: A tree decomposition approach. VLDB (2021)
60. Zhang, X., Özsu, M.T.: Correlation constraint shortest path over large multi-relation graphs. VLDB **12**(5), 488–501 (2019)
61. Zhao, J., Li, L., Zhang, M., Luo, Z., Zhao, X., Zhou, X.: A just-in-time framework for continuous routing. In: ICDE. IEEE (2024)
62. Zheng, B., Zheng, K., Xiao, X., Su, H., Li, G.: Keyword-aware continuous knn query on road networks. In: ICDE, pp. 871–882 (2016)
63. Zheng, K., Su, H., Zheng, B., Shang, S., Xu, J., Liu, J., Zhou, X.: Interactive top-k spatial keyword queries. In: ICDE (2015)
64. Zhong, R., Li, G., Tan, K.L., Zhou, L., Gong, Z.: G-tree: An efficient and scalable index for spatial search on road networks. TKDE (2015)
65. Zhu, H., Li, W., Liu, W., Yin, J., Xu, J.: Top k optimal sequenced route query with poi preferences. DSE **7**(1), 3–15 (2022)