基于空间关键词的Top-k最优路径加速查询框架

张天钰,李艳红*,肖梦,王佳佳
(中南民族大学 计算机学院(人工智能学院),湖北 武汉 430074)

摘 要 针对空间关键词路径查询中的高计算复杂度问题,提出了一种高效的Top-k最优路径加速查询(OPAQ)框架.该框架以用户指定的起点、终点和空间关键词为输入,返回满足所有关键词约束的k条最优路径.为提升查询效率,OPAQ框架包括多项关键优化策略:利用兴趣点(POI)子图筛除无关顶点以缩小搜索空间;构建LG-Tree以支持任意两点之间的快速最短距离查询;构建SP-Tree共享路径前缀信息;结合启发式偏离查询策略,有效发现满足剩余关键词的路径.此外还定义了路径有效性的形式化条件,并结合共享前缀结构与启发式策略提出高效的路径扩展方法,用于支持Top-k路径结果的快速生成.最后在多个真实世界数据集上进行实验,验证了OPAQ框架的优越性能,其在查询时间和路径质量方面均优于现有主流算法,尤其在Top-k查询中展现了良好的效率与可扩展性.

关键词 空间关键词;最优路径查询;加速查询框架;LG-树;SP-树
中图分类号 TP399 **文献标志码** A **文章编号** 1672-4321(2026)05-0683-14
doi:10.20056/j.cnki.ZNMDZK.20260712

An accelerated query framework for Top-k optimal paths with spatial keywords

Zhang Tianyu, Li Yanhong*, Xiao Meng, Wang Jiajia
(College of Computer Science (College of Artificial Intelligence), South-Central Minzu University, Wuhan 430074, Hubei China)

Abstract To address the high computational complexity in spatial keyword path queries, an efficient Top-k Optimal Path Acceleration Query (OPAQ) framework is proposed. Given a user-specified start point, end point, and a set of spatial keywords, the framework returns k optimal paths that satisfy all keyword constraints. To improve query efficiency, OPAQ introduces several key optimization strategies: constructing a point of interest (POI) subgraph to eliminate irrelevant vertices and reduce the search space; building an LG-Tree to support fast shortest distance queries between arbitrary node pairs; designing an SP-Tree to share path prefix information; and applying heuristic deviate search strategy to efficiently discover paths that cover the remaining keywords. Additionally, it defines the formal conditions for path validity and proposes an efficient path expansion method that combines the shared prefix structure and heuristic strategies to support the rapid generation of Top-k path results.Finally, experiments are conducted on multiple real-world datasets to verify the superior performance of the OPAQ framework. It outperformed existing mainstream algorithms in both query time and path quality, especially demonstrating excellent efficiency and extensibility in Top-k queries.

Keywords spatial keywords; optimal path query; OPAQ framework; LG-Tree; SP-Tree

随着智慧城市和位置感知服务的快速发展,空间关键词路径查询已成为智能交通、物流配送和位置服务的核心需求[1-3].此类查询要求从起点s到终点t路径中,必须访问包含用户指定空间关键词(如

收稿日期 2025-07-19
* **通信作者** 李艳红(1973-),女,教授,博士,研究方向:时空、时序数据处理,网络大数据,DB与AI融合,E-mail:liyanhong@mail.scuec.edu.cn
基金项目 湖北省自然科学基金资助项目(2017CFB135);中央高校基本科研业务费专项资金资助项目(CZY23019);网络创新及应用型人才课程实践教学研究项目(2019年第一批)

“加油站”、“餐厅”)的节点,并返回满足关键词感知的 k 条最优路径[4-14].然而,由于不同的用户可能有不同的兴趣偏好,对于从起点到终点距离最短的路径也有可能不是用户的最佳选择[5-6].

例 1.如图 1 所示,用户从公司(起点 s)出发准备回家(终点 t),用户需要为小汽车补充燃料,还需要去吃饭并且购买水果再回家,那么用户的需求就是经过加油站、餐厅和水果店.虽然绿色路线距离最短,但是它并不包含加油站,不符合用户的途径需求.而红色路线和紫色路线均包含加油站、餐厅和水果店,其中紫色路线是符合用户兴趣偏好距离最短的路线,大部分路线推荐算法会返回紫色路线.但是红色路线也是符合用户兴趣偏好的路线,虽然红色路线的路程距离比紫色路线的路程距离稍长,但是因为餐厅和水果店的地理位置是重叠的,用户只需要停两次车就可以满足兴趣偏好,红色路线同样也可能被视为用户的最佳选择路线.因此,返回距离最短的路线或仅返回一条合适路线可能仍然不足以满足用户的路线需求.

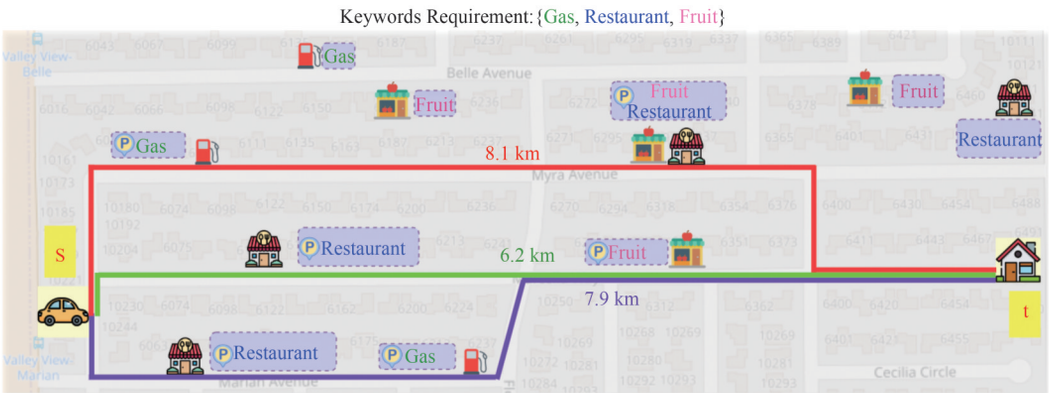


图 1 满足需求的替代路线

Fig. 1 Alternative routes meeting the requirements

受此启发,本文专注于研究基于空间关键词的 Top-k 最优路径查询问题,该问题为用户提供一组能够满足所有查询空间关键词并按照距离递增进行排序的 Top-k 最优路径集合.根据每个兴趣点(POI)中的访问顺序和关键词编号,现有问题可分为以下几类:(1)最优序列路径(OSR),每个 POI 只允许有一个具有固定访问顺序的关键词[7].KOSR 是 OSR 的 Top-k 版本[4];(2)旅行计划查询(TPQ),每个 POI 也只允许有单个关键词,但访问顺序可以是任意的[15];(3)集合空间关键词的最优路径查询(ORCSK),具有集体空间关键词的最优路线[8-9].K-ORCSK 是 ORCSK 的 Top-k 版本[16].

首先分析了现有最短距离查询和空间关键词搜索问题的相关工作,指出传统的最短路径方案难以满足用户多兴趣关键词、多组合偏好的复杂查询需求,路径搜索过程中存在大量冗余计算、查询效率低下的问题.

其次提出了一种高效的基于空间关键词的 Top-k 最优路径加速查询框架 OPAQ,该框架通过构建仅包含兴趣点的 POI 子图、采用 LG-Tree 存储顶点标签以支持快速最短距离计算、利用 SP-Tree 共享路径前缀信息来提升路径拓展效率,并结合启发式偏离查询策略加速 Top-k 路径生成.本文贡献可以总结如下:

(1)结合 LG-Tree 与 SP-Tree 两种高效的数据结构,设计并实现了路径搜索过程中距离计算的优化与前缀共享复用机制,从而显著降低了多次路径计算的重复开销,构建了一个适用于复杂空间关键词路径查询任务的加速框架(OPAQ),能够在保证结果精确性的同时提升查询效率;

(2)引入基于启发式的偏离查询策略,通过对搜索空间的动态调整与优先级控制,加快了 Top-k 有效路径的发现速度,显著缩短了高质量候选路径生成的时间;

(3)在多个真实世界数据集(包括道路网络与兴趣点数据)上进行了大规模实验评估,从查询时间、内存开销、路径质量等多个维度验证了 OPAQ 框架的高效性与优越性能,并与多种现有主流方法进行了对比分析.

1 相关工作

1.1 最短距离查询

最短距离查询是空间路径规划中最基础也是

最核心的计算任务之一.传统方法如Dijkstra算法和A*算法在小规模图上具有良好表现[17]，但在大规模道路网络中面临性能瓶颈，难以满足实时性需求[15].为此，后来的研究提出了多种索引结构与预处理策略以提升最短距离查询效率.

G-Tree方法是一种经典的分层索引结构，它通过将路网划分为多个递归子图并构建树状结构，实现了两点之间的快速最短路径查询[18-20].该方法利用边界顶点和边界距离的预计算，大大减少了查询过程中图的遍历范围.为了进一步降低路径距离计算成本，PLL方法通过对每个顶点预存多个关键“地标点”的距离信息，在不显式遍历图结构的情况下完成快速查询[21].然而，PLL对存储空间要求较高，且不适用于高频次、多源目标查询场景[22].

为了在查询效率与空间成本之间取得平衡，H2H提出将图压缩为多层高速通道，并在查询时优先沿高速边进行搜索[23]；而IG-Tree在路径选择阶段进行全图裁剪，适用于特定关键词相关路径查询[9].但这些结构均未针对“空间关键词+最短距离”这种组合查询进行优化.

1.2　空间关键词搜索

空间关键词搜索是近年来位置感知服务中的重要研究方向，其目标是在地理空间中查找既满足地理位置约束，又满足语义关键词约束的兴趣点(POI).早期研究主要聚焦于静态点查询，例如IR-tree和SKI树，通过将R-tree空间索引与倒排索引结合，在支持地理位置筛选的同时实现对关键词的快速匹配[24-25].然而，这类方法主要面向单点检索，难以扩展到路径级别的多点组合查询任务.

随后，学界开始探索空间关键词路径查询问题.OSR模型首次引入了“访问顺序+关键词集合”的概念，要求用户沿路径依次访问不同类型的兴趣点，但该模型仅支持每个关键词映射一个POI，灵活性不足[7].TPQ模型允许任意顺序访问关键词，但仍局限于每个POI仅包含一个关键词的场景[15].为提升实用性，ORCSK问题提出支持单个POI包含多个关键词并实现关键词集合覆盖，逐渐成为复杂路径推荐研究的主流方向[8-9].

近年来，也有方法尝试将空间关键词查询引入到Top-k路径检索任务中，例如KSP-CSK基于路径枚举思路，暴力搜索所有覆盖关键词的路径后按距离排序，查询效率低下[26]；DA-CSK通过启发式扩展方式提升路径发现速度，但其未能充分利用路径结构中的冗余共享信息，导致重复计算较多[16].

2　问题定义

道路网络是一个图$G(V,E)$，其中V是一组顶点集合，$E\subseteq V\times V$是一系列边的集合，每一条边$(u,v)\in E$都有一个权重$\omega(u,v)$.一条路径p是多个顶点的序列$<v_0,...,v_n>$，其中$(v_i,v_{i+1})\in E$，$\forall i\in[0,n-1]$.p的距离被记为$\omega(p)=\sum_{i=0}^{n-1}\omega(v_i,v_{i+1})$.每个顶点$v_i\in V$包含零个或多个关键词.$K(\cdot)$表示顶点的关键词集，例如，$K(\{v_i\})=\{\mathrm{kw}_1,..,\mathrm{kw}_l\}$表示$v_i$的关键词集.需要注意的是，不同的POI对应的关键词集合长度可能是不一样的.类似的，路径p的集合关键词集表示为$K(p)$，G中所有顶点的关键词的并集表示为$K(V)$.此外，为每种类型的关键词kw_i创建了一个倒排列表$O(\mathrm{kw}_i)$，以存储包含关键词kw_i的顶点.

如图2所示，$K(V)=\{\mathrm{kw}_1,\mathrm{kw}_2,\mathrm{kw}_3\}$，$O(\mathrm{kw}_1)=\{v_2,v_3,v_7\}$.假设起点$s=v_1$，终点$t=v_9$，有一条路径$p=<v_1,v_2,v_3,v_6,v_9>$，其中关键词集$K(p_1)=\{\mathrm{kw}_1,\mathrm{kw}_2\}$，$K(\{v_2\})=\{\mathrm{kw}_1\}$，$K(\{v_3\})=\{\mathrm{kw}_1,\mathrm{kw}_2\}$，$K(\{v_6\})=\{\mathrm{kw}_2\}$.接下来，作如下定义.

Information

POIs	keywords
v_2	kw_1
v_3	kw_1, kw_2
v_5	kw_3
v_6	kw_2
v_7	kw_1, kw_3
v_8	kw_2

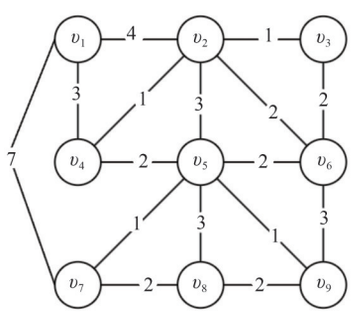


图2　带POI的道路网络

Fig. 2　Road network with POIs

定义1(有效路径)　给定一组关键词的集合Q_{kw}，存在一条路径p，如果p包含的关键词集合覆盖了关键词集合Q_{kw}，即$Q_{\mathrm{kw}}\subseteq K(p)$，则称路径$p$为有效路径.

定义2(空间关键词的Top-k最优路径查询)　给定一个道路网络$G(V,E)$，查询请求$Q=\{s,t,Q_{\mathrm{kw}},k\}$，其中$s$代表起点，$t$代表终点，$s,t\in V$，$Q_{kw}$表示需要满足的关键词集合，$Q_{\mathrm{kw}}\subseteq K(V)$，$K(V)$表示图中所有节点的关键词集合，$k$表示最终返回$k$条最优路径.$P$是所有从$s$到$t$可能的有效路径集合.对于$Q=(s,t,Q_{\mathrm{kw}},k)$，目标是返回$p\in P$，其中$p$满足：

$(1)Q_{kw} \subseteq K(p)$;$(2)w(p) \leqslant w(p'), p' \in P \backslash p$.

例2. 如图2所示,起点$s = v_4$,终点$t = v_6$,关键词集合$Q_{kw} = \{kw_1, kw_3\}$,$k = 2$的查询请求$Q = (v_1, v_9\{kw_1, kw_3\}, 2)$,这里有几条从$v_4$到$v_6$的路径:$p_1 =< v_4, v_5, v_6 >$,$p_2 =< v_4, v_2, v_5, v_6 >$,$p_3 =< v_4, v_5, v_2, v_6 >$.它们对应的关键词集合和长度分别是:$K(p_1) = \{kw_3\}$,$K(p_2) = K(p_3) = \{kw_1, kw_3\}$,$\omega(p_1) = 6$,$\omega(p_2) = 9$,$\omega(p_3) = 10$.因为$p_1$不满足$Q_{kw} \subseteq K(p)$,因此$p_1$不是有效路径,此次查询返回的2条路径是$\{p_2, p_3\}$.

3 框架介绍

3.1 朴素枚举算法和OPAQ框架

路径枚举算法. 该算法的目的是给定图$G(V, E)$,查询请求$Q(s, t, Q_{kw}, k)$,其中s表示起点,t表示终点,Q_{kw}表示要查询的关键词,输出k条最优路径集合φ.具体在算法1中展示.首先找到从起点s到终点t的最短路径p_0,并将p_0加入到优先队列PQ中.对于每条路径,都会记录它最后一个前驱顶点,例如对于p_0而言,s就是它最后一个前驱顶点(算法1第1-2行).当PQ非空时,对PQ中顶部的路径p_{top}进行检索,如果该路径满足了所有的查询关键词,则将其加入到结果集φ中.当φ中有k条路径时,算法终止(第3-8行);如果没有,就继续扩展p_{top}以生成新的候选路径.对p_{top}进行操作时,首先会找到p_{top}的最后一个前驱节点v到终点t的最短路径,后续的搜索会排除掉v和p_{top}上所有的顶点.通过连接p_{top}和从v到t的新路径,以形成一条从s到t的完整路径p',枚举算法一直循环直至找到k条路径并返回(第9-14行).

由于p_{top}最多能拥有$|V|$个顶点,每次扩展都需要耗费$O(|V|\log|V| + |E|)$,而且最终的结果集φ需要k条路径,因此总的时间复杂度是$O(K|V|(|V|\log|V| + |E|))$.

如图3所示,提出的OPAQ框架旨在从以下方面来减少时间复杂度:(1)原始图中大量的顶点都不包含关键词属性,因此可以忽略掉原始图中的非POI顶点,构造一个全是POI的子图;(2)最优路径的评判标准是最短距离路径,而在路径搜索中存在大量的最短距离计算过程,因此可以通过构造LG-Tree来存储顶点的标签信息以加速两点之间的最短距离计算;(3)在进行k条最优路径搜索时,可以通过构造共享前缀树SP-Tree来减少关键词的搜索成本;(4)在偏离查询时,对偏离查询进行启发式计算,以简化查询和计算过程.特此说明,顶点是针对于图结构,节点是针对于树结构.

算法1 Path Enumeration-Based k-ORCSK

输入 $G(V, E)$, $Q(s, t, Q_{kw}, k)$

输出 The Top-k Optimal Routes φ

```
1 p0 ← ShortestPath(s, t), Prefix[p0] ← {s}, φ ← ∅
2 PQ.insert(p0, δ(p0))
3 while |φ| < k do
4 p_top ← PQ.pop()
5   if K(p_top) = Q_kw then
6 φ ← φ ∪ p_top
7       if |φ| = k then
8           break
9 p_pre ← Prefix[p_top]
10      for v ∈ p_top/p_pre do
11 p' ← Prefix[p_top] ⊕ ShortestPath(v, t)
12 PQ.insert(p'.δ(p'))
13 p_pre ← p_pre ∪ {v}, Prefix[p'] ← p_pre
14 return φ
```



图3 OPAQ框架

Fig. 3 OPAQ framework

3.2 图的收缩

对图G进行搜索时,每一条候选路线中一定会包含从一个POI到另一个POI的最短路径.由于图G中只有一部分顶点包含关键词,因此可以省略没有关键词的顶点,并且通过保持距离来生成子图PG,从而减小总体的探索空间.

定义3(保持距离) 给定图$G = (V_G, E_G, \omega_e)$和图$PG = (V_{PG}, E_{PG}, \omega_e)$,如果$\forall u, v \in V_{PG}$,$sd_G(u, v) = sd_{PG}(u, v)$,则称PG保持了$G$中的距离.

采用基于顶点收缩的子网生成方法[27],从图G中生成子图$PG = (V_{PG}, E_{PG}, \omega_e)$.对于在图$G$中一对

POI之间的最短距离路径，如果该路径途径非POI（即两个POI不是直接相连），那么需要为这两个POI添加一条边，边的权重等于图G中两个POI之间路径的最短距离，作为保持距离.从第一个非POI开始，通过检查其每个邻居顶点来收缩路径.如果经过该邻居顶点，则从G中删除该顶点和相应的边，并新增一条将最短距离作为权重的新的边，当所有的非POI被删除后，图的收缩结束.

3.3　LG-Tree

G-Tree的主要思路是将一个路网划分为若干个大小几乎相同的子图，并且为子图创建一棵树，这样可以尽量保证后续生成的G-Tree是一颗平衡树.利用树的结构进行预处理并构造索引，从而在静态图上实现两点之间高效的最短距离查询，而在G-Tree中存储顶点的标签，构造一颗LG-Tree，能够在大型路网中实现更快速的最短距离查询.

定义4（边界顶点）　给定两个顶点v_i，v_j和两个子图g_i，g_j，其中$v_i \in g_i$，$v_i \notin g_j$，$v_j \in g_j$，$v_j \notin g_i$.如果v_i和v_j之间存在一条边e_{v_i,v_j}，且$e_{v_i,v_j} \notin g_i$，$e_{v_i,v_j} \notin g_j$，$e_{v_i,v_j} \in G.E$，则称v_i和v_j是边界顶点.

定义5（LG-Tree）　LG-Tree是一颗平衡树，它的根节点表示整个图G，每个非根节点都对应其父节点的子图，LG-Tree具有以下性质：(1)每一个非叶子节点最少拥有2个孩子节点；(2)LG-Tree中的所有叶子节点只会在一个层次中出现，并且每一个叶子节点最少包含2个顶点；(3)边界顶点的标签列表被所有叶子节点共享.

定义6（边界图G_b）　$G_b = (V_{G_b}, E_{G_b}, \omega_{G_b})$是图$G$的边界子图，$V_{G_b}$由$G$中所有边界顶点组成.$E_{G_b}$有两种类型：(1)如果两个边界顶点在图$G$中直接相连，则保留原有边；(2)如果两个边界顶点在图G中没有直接相连接，但是为了存储它们之间的距离信息，则会在它们之间生成一条累计边，累计边的权重称为累计距离，代表这两个边界顶点之间的最短距离.

例3.如图4所示，在子图E中v_7和v_9相连，v_9和v_8也相连，但是v_9不是边界顶点，因此在v_7和v_8之间生成一条权重为4的累计边.

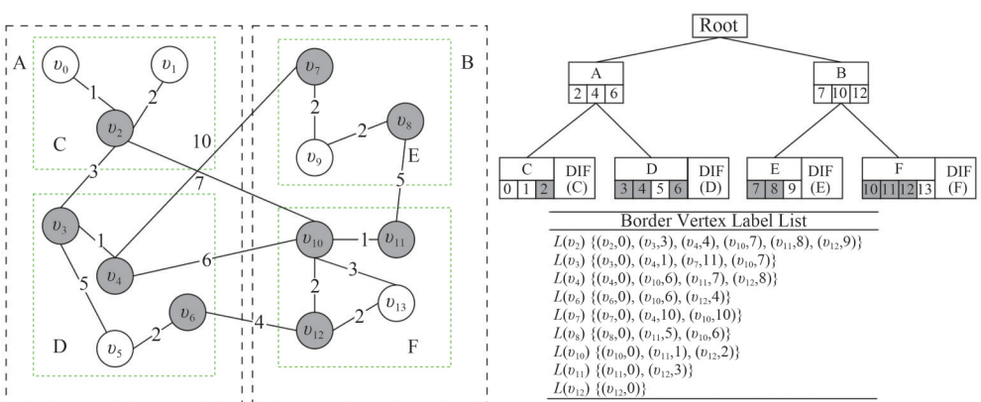


图4　分割图和其对应的LG-Tree

Fig. 4　Graph partition and the corresponding LG-Tree

在存储标签之前，需要为每个边界顶点设计一个层次值，这有利于后续计算从起点到终点的最短距离值$\mathrm{sd}(s,t)$以及最短路径$\mathrm{sp}(s,t)$.利用无邻居顶点集获取边界图G_b中边界顶点的层次.

定义7（无邻居顶点）　给定一个图$G = (V_G, E_G, \omega_e)$和一个顶点集V，V包含了图G中所有的无邻居顶点.其中$V \subseteq V_G$，$\forall u, v \in V, (u,v) \notin E_G$.$V$被称为$G$的无邻居顶点集合.

定义8（边界顶点的层次）　给定一个图G，它的边界子图$G_b = (V_{G_b}, E_{G_b}, \omega_{G_b})$，$G_b$的边界顶点层次结构由$(U, G)$组成，其中$U = \{U_1, \cdots, U_n\}$是一组边界顶点集，$G = \{G_1, .., G_n\}$是一组图，$U$和$G$满足以下性质：(1)$V_{Gb} = U_1 \cup \cdots \cup U_n$，并且当$1 \leqslant i < j \leqslant n$时，$U_i \cap U_j = \varnothing$；(2)对于$1 \leqslant i \leqslant n$，$U_i$是$G_i$的无邻居节点集合，也就是，$U_i$中的任意顶点之间是没有直接连接的边的；(3)$G_1 = G_b$，$G_i = (V_{G_i}, E_{G_i}, \omega_{G_i})$，$V_{G_i} = V_{G_b} - U_1 - \ldots - U_{i-1} (2 \leqslant i < n)$.

例4.对于图4给出的原始图G，依据定义3构建边界图G_b在图5中展示.首先，有$G_1 = G_b$.对于任意的单个顶点，它都能被视作一个无邻居顶点集合，所以设置$U_1 = \{v_2\}$.接着有$G_2 = G_1 - U_1$，并且在G_2中，$\{v_3, v_7\}$组成了无邻居节点集U_2.类似地，$\{v_4, v_8, v_6\}$和$\{v_{10}\}$分别组成了G_3中的U_3和G_4中的U_4.在G_5中，为了保持G_4中的距离，会新增权重为3

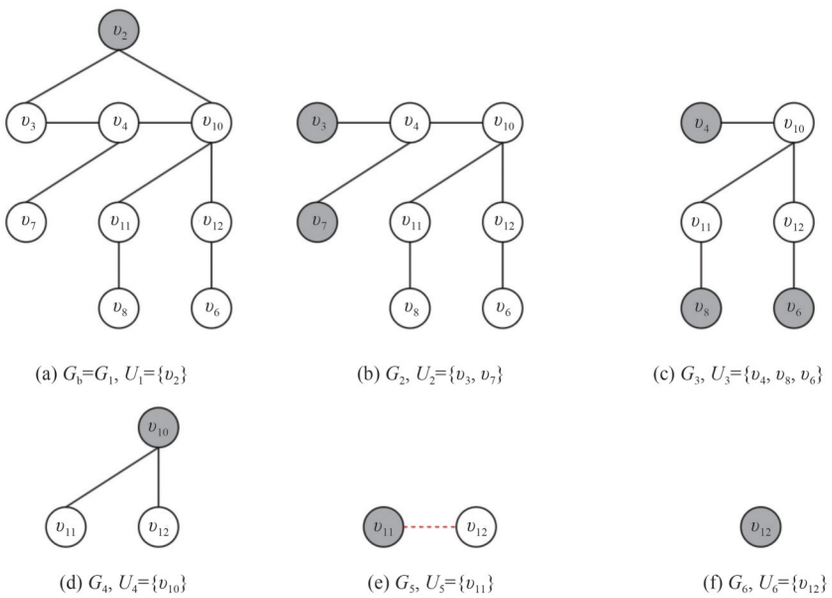


图5 边界顶点的层次结构

Fig. 5 Hierarchical structure of border vertices

的距离保持边(v_{11}, v_{12})到G_5中.U_5和U_6分别由单个节点$\{v_{11}\}$和$\{v_{12}\}$组成.至此,G_b的层次结构构造完毕.

对于距离保持边,作如下解释:对于任意的边界顶点$v \in U_{i-1}$,如果$(u,v) \in E_{G_{i-1}}$,$(v,m) \in E_{G_{i-1}}$,而$u, m \in V_{G_i}$,那么在G_i中的(u,m)的保持距离$\omega_{G_i}(u,m) = \omega_{G_{i-1}}(u,v) + \omega_{G_{i-1}}(v,m)$;如果$u$和$m$原始就是直接相连的,那么$\omega_{G_i}(u,m) = \min(\omega_{G_{i-1}}(u,m), \omega_{G_{i-1}}(u,v) + \omega_{G_{i-1}}(v,m))$.例如,$\omega_{G_5}(11,12) = \omega_{G_4}(11,10) + \omega_{G_4}(10,12) = sd(11,10) + \mathrm{sd}(10,12) = 1 + 2 = 3$.

顶点标签可以高效计算边界顶点之间的最短距离,而边界顶点层次构造又是计算顶点标签的最关键的一步.

定义9(边界顶点级别) 对于每一个边界顶点$v \in V_{G_b}$都会被分配一个级别,记作$l(v)$.如果$v \in U_i$,则$l(v) = i$.

定义10(边界顶点标签) 对于每个边界顶点$v \in V_{G_b}$的标签,记作$L(v)$,是一个二元组$(u, \mathrm{ud}(v,u))$.首先,对每个边界顶点$v \in V_{G_b}$,添加$(v,0)$到$L(v)$中并且标记v.然后,对于v的每个祖先,取一个已经被标记并且具有最低级别$l(u)$的边界顶点u,抹除其标记.设置$l(u) = j$,对于每一个顶点$m \in \mathrm{ner}_{G_j}(u)$,如果$l(m) > j$并且$(m, \mathrm{ud}(v,m))$还未被加入$L(v)$,就将元组$(m, \mathrm{ud}(v,u) + \omega_{G_j}(u,m))$添加到$L(v)$中去,同时标记$m$.其中$\mathrm{ud}(v,u)$是$\mathrm{sd}(v,u)$的上界距离,$\mathrm{ud}(v,m) = \min(\mathrm{ud}(v,m), \mathrm{ud}(v,u) + \omega_{G_j}(u,m))$.当没有顶点被标记时,边界顶点的标签计算结束.

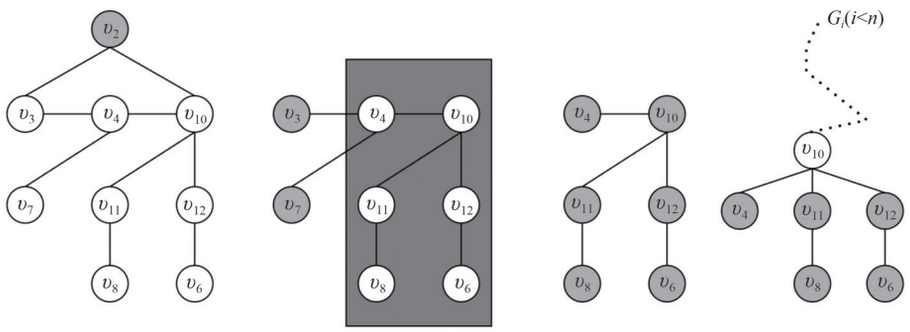
例5.图5(a)展示了图4中所有的边界顶点.标签计算过程从U_1所包含的v_2开始,此时在层次结构中$U_2 = \{v_3, v_7\}$,$U_3 = \{v_4, v_8, v_6\}$,$U_4 = \{v_{10}\}$,$U_5 = \{v_{11}\}$,$U_6 = \{v_{12}\}$.对于v_2的标签构造,首先将$(v_2, 0)$加入到$L(v_2)$中.因为$\mathrm{ner}_{G_b}(v_2) = \{v_3, v_{10}\}$,所以再将$(v_3, 2)$,$(v_{10}, 7)$加入到$L(v_2)$中;然后,通过在$G_3$中检查$v_3$的邻居$v_4$,将$(v_4, 4)$加入到$L(v_2)$中.通过检查$v_{10}$的邻居$v_{11}$,$v_{12}$,将$(v_{11}, 8)$和$(v_{12}, 9)$加入到$L(v_2)$中.至此,$v_2$的顶点标签构造完毕.图4中分割图里所有的边界顶点标签都在图4中LG-tree中展示出来(经过顺序处理).

树的分解是解决最大独立集问题的有效方法,但是它并不能限制节点层次的最高级别.在某些情况下,分解树的部分层数可能包含的节点很少,从而导致分解树的总层数很高,则需要大量时间去遍历分解树.构造边界顶点层次时也会遇到同样的问题:对于图中的每个边界顶点而言,随着路网图的增大,边界顶点的层次等级也会随之增大,从而导致更大的搜索开支,也会让后续的边界顶点标签构造变得更加复杂.因此,提出了一个限制边界顶点层次的启发式方法,目的是使边界顶点层次的构造限制在n层以下.

定义11(启发式顶点层次) 给定一个边界子

图 $G_b=(V_{G_b},E_{G_b},\omega_{G_b})$，它的边界顶点层次$(U,G)$最多有 n 级．假设 x 是 G_i 的等级，y 是无邻居顶点集 U_i 中的顶点数量，记为 $|U_i|$．假设在 x 和 y 之间存在一个函数 $y=f(x)$，依据观察可以推断函数 $y=f(x)$ 是非单调递增的．如果 x 是 $f(x)$ 的第一个拐点，那么 $n=x$．如果第一个拐点不是 x，那么 $n=\bar{x}$，其中 $\bar{x}$ 是 x 的平均值．对于每个边界顶点 $v\in U_i$，它的层次等级被设为 $l(v)=i,1\leqslant i<n$．对于每个边界顶点 $v\in V_{Gn}$，其层次等级被设为 $l(v)=n$．

例 6．如图 6(a)，(b)，(c)中构造的层次结构所示，$f(1)=1,f(2)=2,f(3)=3,f(4)=1,f(5)=1,f(6)=1$，可以观察到 $f(3)>f(4)$，因此 $x=3$ 是 $f(x)$ 的第一个拐点，$n=x=3$．最终，$U=\{U_1,U_2\}$，G_3 是最高等级的图．对于 G_3 中的每个边界顶点 v，$l(v)=3$．



(a) G_b=G_1, U_1=$\{v_2\}$　　(b) G_2, U_2=$\{v_3, v_7\}$　　(c) G_3

Heuristic Border Vertex Label List
$L(v_2)\{(v_2,0),(v_3,3),(v_4,4),(v_{10},7),(v_{11},8),(v_{12},9)\}$
$L(v_3)\{(v_3,0),(v_4,1),(v_7,11),(v_{10},7)\}$
$L(v_7)\{(v_7,0),(v_4,10),(v_{10},10)\}$
$L'(v_4)\{(v_{10},1,6),(v_{11},1,7),(v_{12},1,8),(v_6,2,8),(v_8,2,12)\}$

图6　启发式层次构造和搜索树

Fig. 6　Heuristic hierarchical construction and search tree

但是启发式构造边界顶点层次无法直接记录 G_n 中所有边界顶点的距离信息，当两个边界顶点分别属于 $G_{i<n}$ 和 G_n 时，则利用相对距离来替代边界顶点标签进行最短距离的搜索．而 G_3 中只有 v_4 有累计距离，是因为其它边界顶点并没有与 $G_i(i=n-1)$ 中的边界顶点直接相连．对于 G_3 中的所有顶点，作为根节点的依据是 $\max(|\mathrm{ner}(v_i)|)$，因此选取 v_{10} 作为搜索树根节点．通过启发式构造边界顶点层次，可以在更少的空间中存储更多的距离信息．

在启发式边界顶点层次构造中，需要计算两个边界顶点之间的距离，一共可以分为 3 种情况：(1)两个边界顶点都属于 $G_{i<n}$；(2)两个边界顶点都属于 G_n；(3)一个边界顶点属于 $G_{i<n}$，另一个边界顶点属于 G_n．对于第 1 种情况，可以直接获取两边界顶点之间的最短距离；对于第 2 种情况，需要判断两个边界顶点是否属于 G_n 中的同一个分支，如果属于同一个分支，则 $\mathrm{sd}(v,u)=|\mathrm{sd}(v,m)-\mathrm{sd}(m,u)|$，否则 $\mathrm{sd}(v,u)=\mathrm{sd}(v,m)+\mathrm{sd}(m,u)$，其中 m 是 u 和 v 的父节点；对于第 3 种情况，$\mathrm{sd}(v,u)=\mathrm{sd}(v,t)+\mathrm{sd}(t,u)$，其中 t 是 G_n 中拥有最大度的边界顶点．

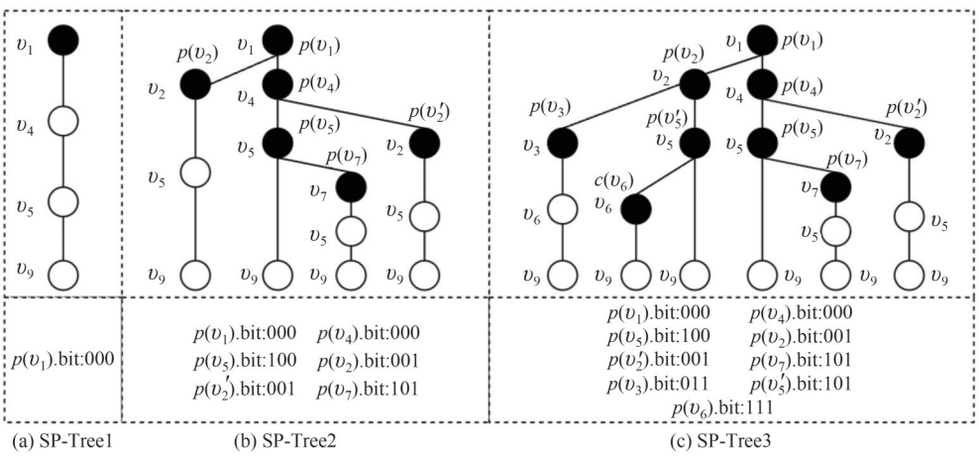
对于两个顶点 v 和 u 之间的最短距离搜索，也分为 3 种情况：(1)v 和 u 都是边界顶点；(2)v 不是边界顶点，u 是边界顶点；(3)v 和 u 都不是边界顶点．对于第 1 种情况，可以在边界图中的标签列表直接获取 $\mathrm{sd}_{G_b}(v,u)$；对于第 2 种情况，需要获取 v 所在的子图中所有的边界顶点 $\{\beta_j,j=1,2,...\}$，然后依据 $\mathrm{sd}(v,u)=\min(\mathrm{sd}(v,\beta_i)+\mathrm{sd}_{G_b}(\beta_i,u)$ 进行搜索；对于第 3 种情况，需要分别获取 v 所在的子图中所有边界顶点 $\{\beta_i,i=1,2,...,n\}$ 和 u 所在的子图中所有边界顶点 $\{\beta_j,j=1,2,...\}$，然后依据 $\mathrm{sd}(v,u)=\min(\mathrm{sd}(v,\beta_i)+\mathrm{sd}_{G_b}(\beta_i,\beta_j)+\mathrm{sd}(\beta_j,u))$ 来计算最短距离．

3.4　SP-Tree

在枚举算法中，由于所有满足查询条件的路径都是从一条由起点 s 到终点 t 的最短距离路径拓展得到的，而在搜索过程中可以共享拓展路径的前缀信息，再结合树的结构特性，因此可以创建一颗共享前缀树 SP-Tree，能够存储公共信息以便重复使用，进而提升查询效率和减少存储成本．SP-Tree 的根节点是起点 s，SP-Tree 上的节点是候选部分路径上的顶点，叶子节点是经过上一步进行偏离查询得

到的POI.显然,所有的叶子节点都可以满足至少一个查询关键词.用$p(v_i)$表示从s到v_i的前缀路径.为了快速检查路径上还未满足的关键词,为每个节点附加一个位图来表示关键词满足状态,可以进一步提高查询效率.在SP-Tree中,因为不同顶点经过偏离查询得到的顶点可能是同一个顶点,因此一个顶点可能会出现多次,所以可以在SP-Tree中对重复出现的顶点进行标记加以区分.

例7. 对于图2中的查询请求$Q=(v_1, v_9, \{\mathrm{kw}_1, \mathrm{kw}_2, \mathrm{kw}_3\}, 2)$,首先找到从$v_1$到$v_9$的最短距离路径$\mathrm{sp}=(v_1, v_4, v_5, v_9)$,构造初始的共享前缀树$\mathrm{SP\text{-}Tree}_1$,如图7所示.在$\mathrm{SP\text{-}Tree}_1$中通过偏离$v_1$, v_4, v_5得到最近的POI节点v_2, v_2, v_7,此时$<v_1, v_2>$,$<v_1, v_4, v_2>$,$<v_1, v_4, v_5, v_7>$多条偏离路径会生成并最终组成$\mathrm{SP\text{-}Tree}_2$.与此同时,这些路径上所包含的关键词也会被记录在位图里,例如,$<v_1, v_4, v_5, v_7>$包含了kw_1和kw_3,因此$p(v_7).\mathrm{bit}=101$.由于v_1和v_4偏离得到的最近POI顶点都是v_2,因此用$p(v_2)$和$p(v_2')$来进行区分.用类似的操作可以得到$\mathrm{SP\text{-}Tree}_3$.



(a) SP-Tree1 (b) SP-Tree2 (c) SP-Tree3

图7 对于查询请求$Q=(v_1, v_9, \{\mathrm{kw}_1, \mathrm{kw}_2, \mathrm{kw}_3\}, 2)$的SP-Tree

Fig. 7 SP-Tree for query request $Q=(v_1, v_9, \{\mathrm{kw}_1, \mathrm{kw}_2, \mathrm{kw}_3\}, 2)$

3.5 启发式偏离查询

在共享前缀树SP-Tree中,从叶子节点到终点t的最短路径上还需要不断地进行偏离查询,而这个过程中依然存在着大量偏差,如果只考虑将距离作为标准来扩展SP-Tree,会导致生成SP-Tree的时间开销和空间开销大大增加.因此,在偏离查询中,可以选择将未满足的关键词集成到一个启发式距离,并应用最优范例来引导路径的拓展以便能够更早地找到有效路径.

一个有效路径前缀$p(v_i)$的成本取决于两个因素,其一是$p(v_i)$的成本,其二是从v_i到下一个能够满足剩余未满足关键词的POI的成本.前者可以轻易得到,但是后者充满着不确定性,因为后者的成本取决于查询关键词组合和访问顺序.

为了提高效率,需要简化计算,忽略掉查询关键词组合,只关注哪种关键词是最难满足的.将具有最大下界绕行成本作为最难满足的标准.具体步骤是:首先,获取每条部分路径$p(v_i)$还未满足的剩余关键词集合$Q'_{\mathrm{kw}}=Q_{\mathrm{kw}}\backslash p(v_i).bit$.然后,对于$Q'_{kw}$中的每个关键词,利用倒排索引计算它的最小绕行成本.最后,取其中最大的一个作为满足剩余关键词的下界成本.以$p(v_i)$为前缀有效路径的启发式成本计算公式如下:

$$\mathrm{lb}\left(p(v_i)\right)=\omega\left(p(v_i)\right)+\max_{\mathrm{LB}}=\omega\left(p(v_i)\right)+\max_{\forall \mathrm{kw}_m\in Q'_{\mathrm{kw}}}\left(\min_{\forall v_j\in O(\mathrm{kw}_m)}\left(\mathrm{sd}(v_i, v_j)+\mathrm{sd}(v_j, t)\right)\right).$$

例8. 如图8所示,查询关键词集合$Q_{kw}=\{\mathrm{kw}_1, \mathrm{kw}_2, \mathrm{kw}_3, \mathrm{kw}_4\}$,偏离路径$K\left(p(v_i)\right)=\{\mathrm{kw}_4\}$,所以$Q'_{\mathrm{kw}}=\{\mathrm{kw}_1, \mathrm{kw}_2, \mathrm{kw}_3\}$,其它POI的绕行成本可以通过LG-Tree存储的最短距离标签信息快速得到.对于kw_1,POI集合$O(\mathrm{kw}_1)=\{v_2, v_5, v_6\}$,其中$v_2$具有最小的绕行成本10.类似地,对于$\mathrm{kw}_2$,$v_1$具有最小的绕行成本5,对于$\mathrm{kw}_3$,$v_3$具有最小的绕行成本12.因此,最大的绕行成本$\max_{\mathrm{LB}}=\{10, 5, 12\}=12$,将$v_3$作为下一个偏离节点进而拓展路线.

然而当有许多关键词还未满足时,需要计算大量顶点之间的最短距离,因此采取启发式值优化策略.(1)每一个POI的绕行成本最多只会被计算一次;(2)如果包含了关键词kw_b的POI的绕行成本小

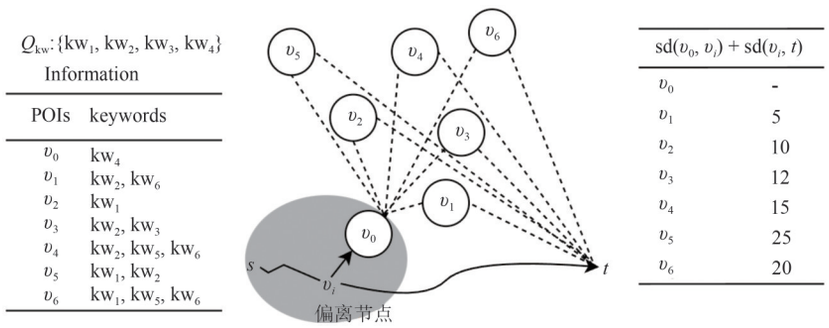


图8　计算v_0的$\max_{LB}$

Fig. 8　Calculate the $\max_{LB}$ of v_0

于现有的$\max_{LB}$，就意味着kw_b的最小绕行成本不会改变$\max_{LB}$，因此包含了kw_b的其它POI的绕行成本也不用计算了．如图9所示，对于kw_2，因为v_1的绕行成本为5，已经小于$\max_{LB}=10$，因此，后续的v_3, v_4, v_5都不被计算．

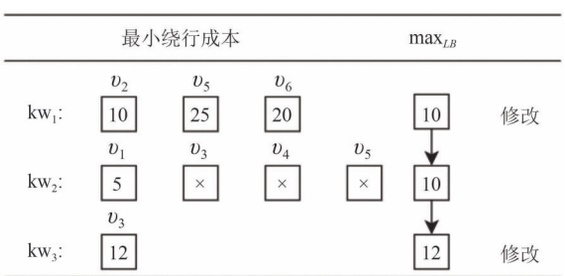


图9　简化图8中的$\max_{LB}$计算

Fig. 9　Simplify the calculation of $\max_{LB}$ in Fig. 8

4　算法介绍

本文提出的算法框架核心部分是依据LG-Tree进行最短距离查询和依据SP-Tree进行Top-k路径查询．

最短距离查询算法．该算法的目的是给定LG-Tree，起点s和终点t，任意边界顶点之间的最短距离$sd_{G_b}(\beta_i,\beta_j)$，输出$s$到$t$的最短距离$sd(s,t)$，具体在算法2中展示．首先初始化$sd(s,t)$，然后分别对$s$和$t$是否为边界顶点进行不同情况的判断处理（算法2第1-2行）．如果s和t都是边界顶点，可以直接在LG-Tree中获取最短距离（第3-4行）；如果s和t中有一个是边界顶点，另一个不是边界顶点（假设s为边界顶点，t不是边界顶点），则会对两段距离进行拼接形成最短距离，其中s到中间边界顶点β_i的最短距离可以直接在LG-Tree中获取，而中间边界顶点β_i到t的最短距离可以在收缩子图中快速查找，最终取最小值就能形成s到t的最短距离（第5-8行）；如果s和t都不是边界顶点，则对三段距离进行拼接形成最短距离，还需要两个中间边界顶点β_i和β_j，其中β_i到β_j的最短距离可以直接在LG-Tree中获取，s到β_i的距离和β_j到t的距离均取最小值，最终形成s到t的最短距离并返回（第9-11行）．

算法2　最短距离查询算法

输入 An LG-Tree, s, t and $sd_{G_b}(\beta_i,\beta_j)$

输出 $sd(s,t)$

1 $sd(s,t) \leftarrow 0$;
2 while s and t are not the same vertex do
3　　if both s and t are border vertices then
4 $sd(s,t) \leftarrow sd_{G_b}(\beta_s,\beta_t)$
5　　else if s is a border vertex and t is a non-border vertex then
6 $sd(s,t) \leftarrow \min(sd_{G_b}(s,\beta_i) + sd(\beta_i,t))$
7　　else if s is a non-border vertex and t is a border vertex then
8 $sd(s,t) \leftarrow \min(sd(s,\beta_i) + sd_{G_b}(\beta_i,t))$
9　　else if both s and t are non-border vertices then
10 $sd(s,t) \leftarrow \min(sd(s,\beta_i) + sd_{G_b}(\beta_i,\beta_j) + sd(\beta_j,t))$
11 Return $sd(s,t)$

Top-k路径查询算法．该算法的目的是给定子图$PG=(V_{PG}, E_{PG})$，查询请求$Q(s,t,Q_{kw},k)$，其中s表示起点，t表示终点，Q_{kw}表示要查询的关键词，输出k条最优路径集合φ，具体在算法3中展示．首先初始化一个优先队列PQ，k条最优路径集合φ，起点s为根节点构造的SP-Tree，最大绕行成本$\max_{LB}$和前缀路径$p(s)$及其成本$\omega(p(s))$都插入到PQ中（算法3第1-4行）．只要优先队列非空，优先弹出下界最小的前缀路径$p(v_m)$，其中下界$lb(p(v_i)) = \omega(p(s)) + \max_{LB}$．如果$p(v_m)$是一个有效路径，就将它加入到最优路径集合$\varphi$中，当找到$k$条路径时就会结束查询（第5-11行）；如果$p(v_m)$不是一个有效路径，那么从$v_m$到$t$的最短路径$sp(v_m,t)$上的所有顶点都会被获取，并被当作新的偏离路线（第12行）．对于每个偏离路线，依据启发式偏离查询来满足还未找到的剩余关键词集合

Q'_{kw}并拓展SP-Tree,关键词位图也会及时更新.具有最大下界的绕行成本的路径会被插入到PQ中,最终返回k条最优路径集合φ(第13-21行).

算法3 Top-k路径查询算法

输入 An SP-Tree, $PG=(V_{PG}, E_{PG})$ and $Q(s, t, Q_{kw}, k)$

输出 The Top-k Optimal Routes φ

1 Initialize a priority queue PQ

2 $\varphi \leftarrow \varnothing, p(s) \leftarrow s$, SP-Tree$_1 \leftarrow \{p(s)\}$

3 $\max_{LB} \leftarrow$ CalMaxLB$(p(s), \varnothing)$

4 PQ.insert($< p(s), \omega(p(s)), \max_{LB} >$)

5 while PQ is not empty do

6 $p(v_m) \leftarrow$ PQ.extract_Min()

7 Bit $\leftarrow p(v_m)$.bit in SP-Tree

8 if $K(p(v_m)) \vee$ Bit $= Q_{kw}$ then

9 $\varphi \leftarrow \varphi \cup \{p(v_m)\}$

10 if $|\varphi| = k$ then

11 break;

12 $C \leftarrow$Get all vertices on the sp$(v_m$,t$)$ utilizing SP-Tree for each v_i in C do

13 $p(v_i) \leftarrow p(v_m) \oplus v_i$

14 $NN_{v_i} \leftarrow$ FindNN$(v_i, K(p(v_i)))$

15 cost $\leftarrow \omega(p(v_i)) + \delta(v_i, NN_{v_i})$

16 $P_a \leftarrow p(v_i) \oplus$ sp(v_i, NN_{v_i})

17 SP-Tree $\leftarrow$ SP-Tree$\oplus P_a$

18 $K(P_a) \leftarrow K(p(v_i)) \vee K(\{NN_{v_i}\})$

19 $\max_{LB} \leftarrow$ CalMaxLB$(P_a, K(P_a))$

20 PQ.insert($< P_a$, cost, $\max_{LB} >$)

21 Return φ;

5 实验评估

5.1 实验设置

数据集:用表1中展示的4个带有POI数据的真实世界道路网络进行实验.其中,CAL,NYC和COL的POI数据来自于真实数据,分别占据60%,10%和8%的顶点数量.FLA的POI和关键词类型采取独特方法生成[4].每种类型的顶点通过均匀分布固定数量,然后依据幂律因子$f=1.4$的齐夫定律将这些顶点分配给POI.选择COL数据集作为默认映射来测试其他参数的影响.对于每个查询,都会随机选择起点和终点以及所需的关键词集,表2中列出了参数,默认设置以蓝色标记.

评估标准:首先提供索引大小和预处理时间,然后,以查询时间和平均距离来评估性能,这反映了算法的有效性.如果查询无法在1 h内停止或内存不足,则将其相应的查询时间表示为INF.所有结果均为1000个查询的平均值.

表1 真实世界道路网

Tab. 1 Real world map

Maps	$\|V\|$/K	$\|E\|$/K	#*POI*/K	$\|K(V)\|$
CAL	21	43	12	63
NYC	264	733	20	427
COL	435	1057	32	548
FLA	1070	2712	73	739

表2 字段

Tab. 2 Field

Parameter	Value
Q_{kw}	1, 2, 4, 6, 8, 10
k	1, 5, 10, 20, 40

方法:实现并扩展了以下算法进行比较:(1)OptDist[9]:通过搜索k个必需的POI来修改它,这些POI可以共同覆盖路径周围所有需要的关键词作为候选关键词,然后删除POI以找到替代POI,并贪婪地选择最短路径作为下一个结果;(2)kRA-MS[8];(3)KOSR-MA[16]:将其NN算法修改为任意关键词顺序,并使用PLL和A*来加速搜索和路径扩展;(4)KSP-CSK[21]:枚举无环路径,直到检索到k个有效路径.将前缀路径中包含的关键词缓存起来,并将关键词信息存储在SP-Tree中;(5)DA-CSK[11];(6)OPAQ是本文提出的算法框架.

5.2 实验结果

(1)索引大小和预处理时间:kRA-MS和KSP-CSK都是直接扩展网络,因此它们没有索引.如图10所示,OptDist、KOSR-MA和DA-CSK比OPAQ的索引结构需要更大的内存,因为OptDist方案的IG-Tree和KOSR-MA的PLL都需要占据大量内存,而DA-CSK的H2H是一个分层结构,只存储了一小部分标签,节省了空间.这些算法的预处理主要包括索引构建时间和POI子网络生成时间.如图11所

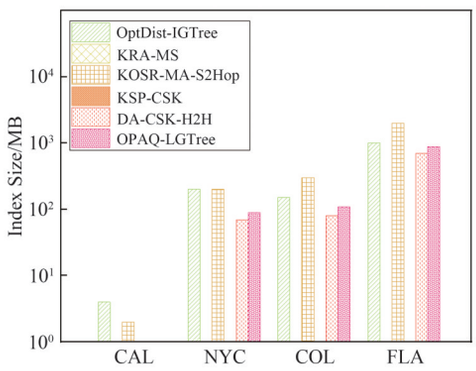


图10 索引结构大小

Fig. 10 Index structuressize

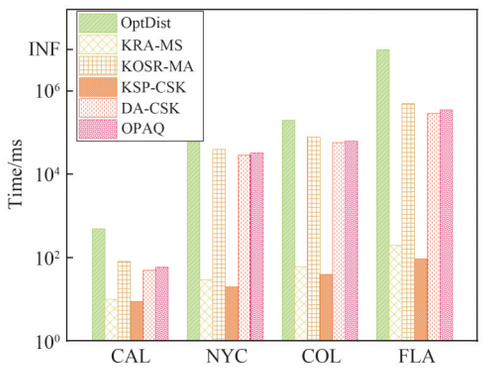


图 11　预处理时间消耗

Fig. 11　Preprocessing time consumption

示，OptDist 消耗的时间最长，因为 IG-Tree 需要递归地对网络进行分区．而 DA-CSK 和 OPAQ 表现最好，OPAQ 表现稍差的原因是其在预处理阶段需构建多个索引结构，用于支持后续查询中的高效访问和路径生成．尽管这一过程在离线阶段耗时较长，但所构建的结构使得在线阶段的搜索过程更加高效，从而在总响应时间上展现出更好的性能．

（2）k=1 时的性能：$|Q_{kw}|$ 的影响．查询时间方面的结果如图 12 所示，CSK 最差，其次是 kRA-MS、KOSR-MA 和 DA-CSK，而 OptDist 和 OPAQ 是最好的．质量方面的结果如图 13 所示，KSP-CSK 和 kRA-MS 表现最差，OPAQ 表现最好．KSP-CSK 在距离递增顺序中盲目地列举了路径，并且不考虑关键词的分布，导致效率低下．kRA-MS 需要大量最近邻查询，其贪婪策略也导致了它需要依靠更长的距离才能得到最终的结果．虽然 KOSR-MA 的扩展效率要好得多，但并没有考虑关键词的组合，导致大量的路径回溯，进一步降低了性能．在 COL 和 FLA 两个数据集中，最具竞争力的算法是 OptDist，OPAQ 的查询时间几乎是 OptDist 的两倍，但仍然处于相同的数量级．在其它数据集中，查询时间没有太大差异．但是在路径距离上，OPAQ 比 OptDist 短了约 10%．此外，数据集规模越大，查询关键词就越多，差距就越大．OptDist 可以快速找到查询关键词与 IG-Tree 的组合，比路径扩展的方法更快．但是，当 POI 接近于最短路径并被组合成一个有效路径时，而且由于邻近的 POI 不在同一侧，可能会导致整体路径增加；相反，OPAQ 基于 LG-Tree 查询最短路径，不断偏离搜索包含未满足的关键词的 POI，并尝试用各种 POI 组合来查找距离最小的路径．

（3）k>1 时的性能：k 的影响．查询时间方面的结果如图 14 所示，结果质量如图 15 所示．当 k 增加时，除 OptDist 外，其它所有算法的查询时间增加缓慢，

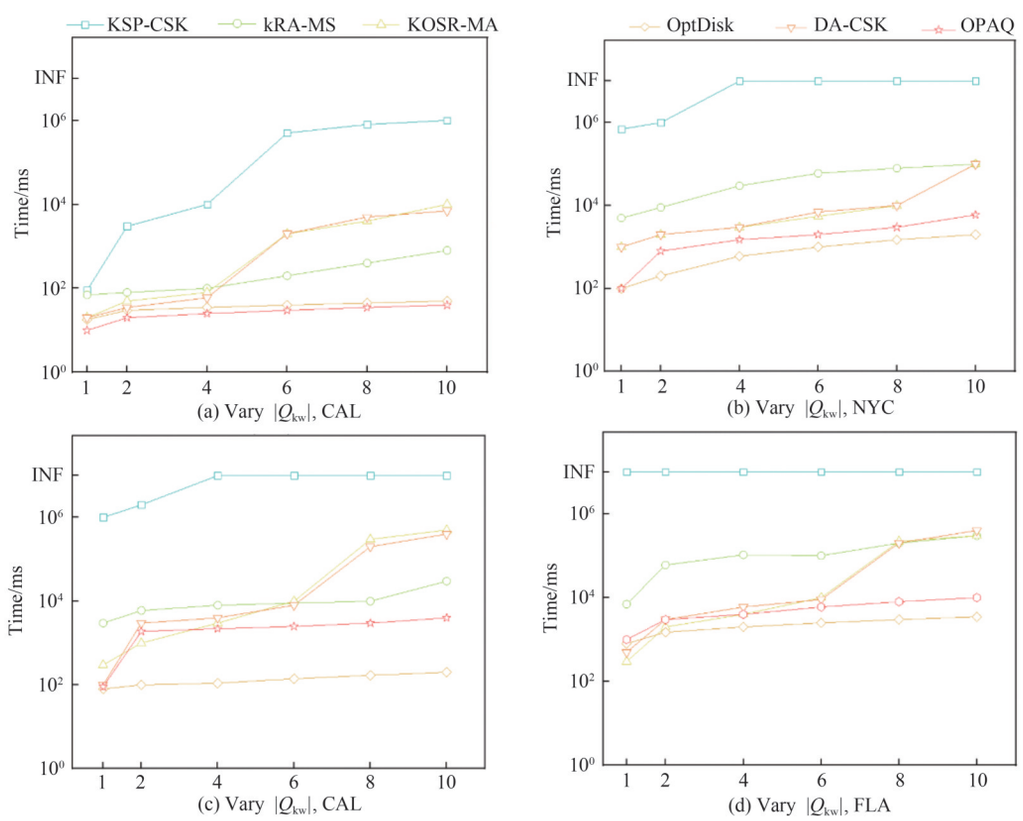


图 12　k=1 时的查询时间

Fig. 12　Query time when k=1

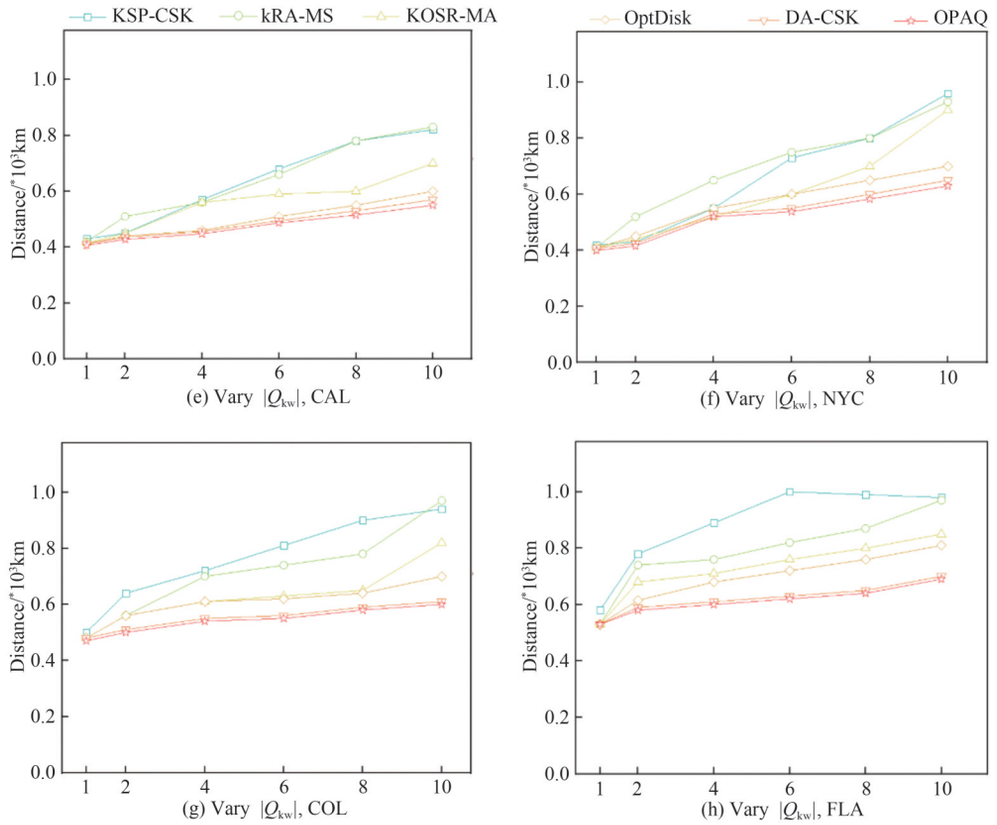


图 13 k=1时的结果质量

Fig. 13 Result quality when k=1

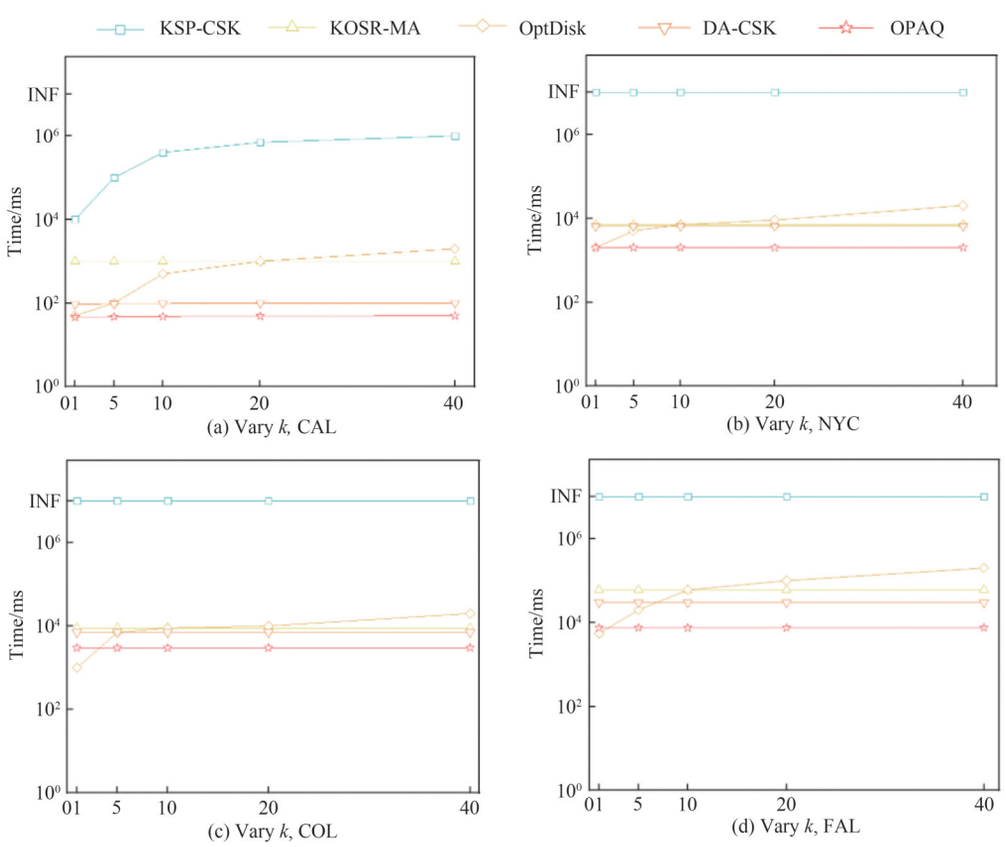


图 14 k>1时的查询时间

Fig. 14 Query time when k>1

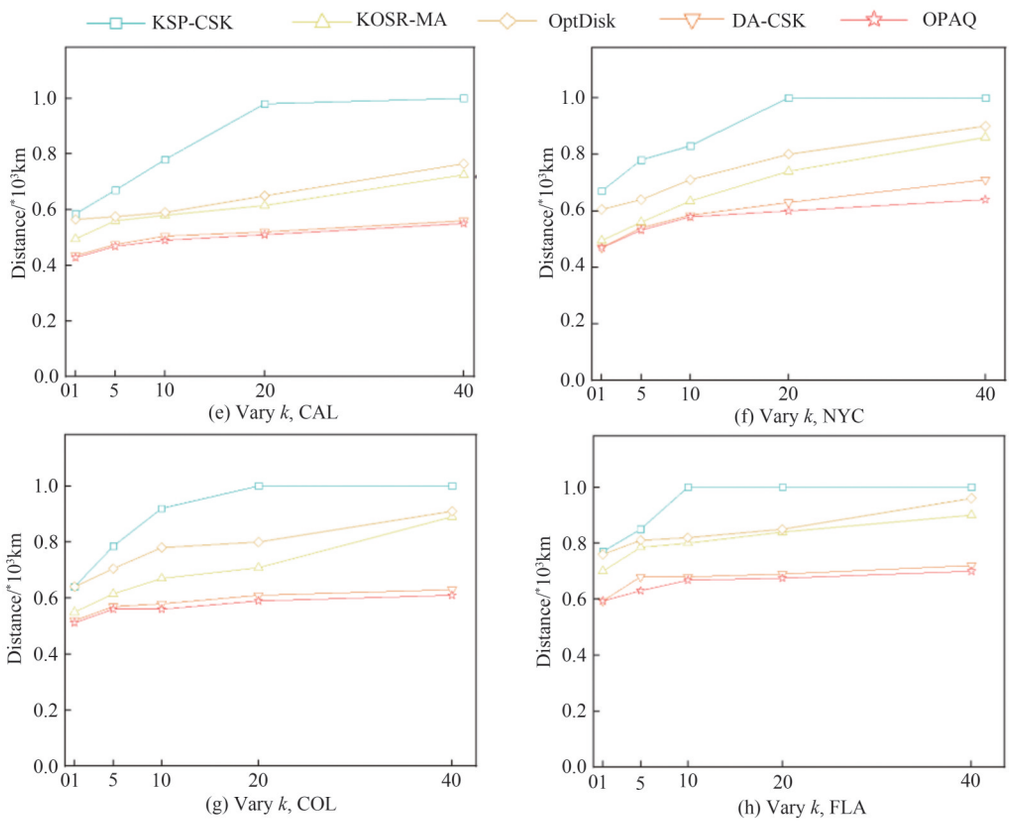


图 15　k>1 时的结果质量

Fig. 15　Result quality when k>1

表现出良好的可扩展性.这是因为这些算法都是基于路径扩展，在搜索第一个最优路径时计算了大量的中间结果，因此通过继续扩展可以很容易地获得有效路径.而 OptDist 是基于空间 POI 选择和贪婪策略，而最大组合消耗了大量的时间成本，导致进行 Top-k 路径搜索时效率低下.当 k≥10 时，OPAQ 比 OptDist 快 1-2 个数量级，并且返回的路径比 OptDist 短了约 30%.路径距离是取所有计算结果的平均值得出的，因此 k 越大，路径距离越长.OPAQ 无论是在查询时间还是路径距离上都是表现最好的.

6　结论

围绕基于空间关键词的 Top-k 最优路径查询问题，本文提出了一个高效的加速查询框架——OPAQ.该框架面向用户在实际场景中提出的多关键词、个性化路径需求，充分考虑了查询精度与计算效率之间的平衡.通过构建 POI 子图以缩减搜索空间，利用 LG-Tree 进行快速的最短距离计算，并结合 SP-Tree 共享路径前缀信息以及启发式偏离查询策略，显著提升了 Top-k 路径的生成效率.

实验结果表明，OPAQ 框架在多个真实道路网络数据集上均表现出优异的性能，不仅在查询时间上远优于现有方法，在路径质量方面也更加贴合用户的实际需求，尤其在查询关键词数量较多或 k 值较大时，具有更好的扩展性与稳定性.实验结果表明本框架在大规模、复杂约束条件下依然有稳定且出色的表现.

未来的研究方向包括：进一步引入关键词优先级排序机制，以支持更加复杂的用户偏好建模；探索动态图或交通流量预测信息下的实时路径规划方法；融合隐私保护机制以适应更加敏感的数据使用场景，从而推动这项研究课题在实际应用中的落地与发展.

参　考　文　献

[1] Al Hasan Haldar N, Li J, Reynolds M, et al. Location prediction in large-scale social networks: An in-depth benchmarking study[J]. The VLDB Journal, 2019, 28(5): 623-648.

[2] Al Hasan Haldar N, Li J, Ali M E, et al. Top-k socio-spatial co-engaged location selection for social users[C]// IEEE Transactions on Knowledge and Data Engineering. IEEE, 2023: 5325-5340.

[3] Chen L, Cong G, Cao X, et al. Temporal spatial-keyword top-k publish/subscribe[C]//2015 IEEE 31st International Conference on Data Engineering. Seoul: IEEE, 2015: 255-266.

[4] Liu H, Jin C, Yang B, et al. Finding top-k optimal sequenced routes[C]//2018 IEEE 34th International Conference on Data Engineering (ICDE). Paris: IEEE, 2018: 569-580.

[5] Liu H, Jin C, Yang B, et al. Finding top-k shortest paths with diversity[C]//IEEE Transactions on Knowledge and Data Engineering. IEEE, 2018: 488-502.

[6] Zhu H, Li W, Liu W, et al. Top k optimal sequenced route query with POI preferences[J]. Data Science and Engineering, 2022, 7(1): 3-15.

[7] Sharifzadeh M, Kolahdouzan M, Shahabi C. The optimal sequenced route query[J]. The VLDB Journal, 2008, 17(4): 765-787.

[8] Lu E H, Chen H S, Tseng V S. An efficient framework for multirequest route planning in urban environments [C]//IEEE Transactions on Intelligent Transportation Systems. IEEE, 2017: 869-879.

[9] Haryanto A A, Islam M S, Taniar D, et al. IG-Tree: An efficient spatial keyword index for planning best path queries on road networks[J]. World Wide Web, 2019, 22(4): 1359-1399.

[10] Xie Q, Zhu F, Feng X. Efficient and secure spatial fuzzy keyword query[J]. IEEE Internet of Things Journal, 2025, 12(10): 14752-14770.

[11] Li J, An Q, Song Y, et al. Route optimization with collective spatial keywords: A skyline-based approach[J]. The VLDB Journal, 2025, 34(5): 61.

[12] Chan H K. CSKQS: A query system for collective spatial keyword queries[C]//2025 IEEE 41st International Conference on Data Engineering (ICDE). Hong Kong: IEEE, 2025: 4608-4611.

[13] 姚静怡, 李艳红, 黄银峰, 等. 灵活的属性社区搜索方法[J]. 中南民族大学学报(自然科学版), 2024, 43(3): 358-369.

[14] 李艳红, 涂锐. 基于密度峰值的top-k空间文本查询[J]. 中南民族大学学报(自然科学版), 2025, 44(2): 260-268.

[15] Li F, Cheng D, Hadjieleftheriou M, et al. On trip planning queries in spatial databases[M]//Advances in Spatial and Temporal Databases. Berlin: Springer, 2005.

[16] Li J, Xiong X, Li L, et al. Finding top-k optimal routes with collective spatial keywords on road networks[C]//2023 IEEE 39th International Conference on Data Engineering (ICDE). Anaheim: IEEE, 2023: 368-380.

[17] Goldberg A V, Harrelson C. Computing the shortest path: A search meets graph theory[J]. Proceedings of the 16thAnnual ACM-SIAM Symposium on Discrete Algorithms, 2009, 5515: 156-165.

[18] Delling D, Sanders P, Schultes D, et al. Engineering route planning algorithms[J]. Lecture Notes in Computer Science, 2009, 5515: 117-139.

[19] Zhong R, Li G, Tan K L, et al. G-tree: An efficient and scalable index for spatial search on road networks[J]. IEEE Transactions on Knowledge and Data Engineering, 2015, 27(8): 2175-2189.

[20] Dan T, Luo C, Li Y, et al. LG-tree: An efficient labeled index for shortest distance search on massive road networks[J]. IEEE Transactions on Intelligent Transportation Systems, 2022, 23(12): 23721-23735.

[21] Akiba T, Iwata Y, Yoshida Y. Fast exact shortest-path distance queries on large networks by pruned landmark labeling[C]//Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. New York New York: ACM, 2013: 349-360.

[22] Delling D, Goldberg A V, Pajor T, et al. Customizable route planning[M]//Experimental Algorithms. Berlin: Springer, 2011.

[23] Sanders P, Schultes D. Highway hierarchies hasten exact shortest path queries[M]//Algorithms – ESA 2005. Berlin: Springer, 2005.

[24] Li Z, Lee K C K, Zheng B, et al. IR-tree: An efficient index for geographic document search[J]. IEEE Transactions on Knowledge and Data Engineering, 2011, 23(4): 585-599.

[25] Chen L, Cong G, Jensen C S, et al. Spatial keyword query processing: An experimental evaluation [J]. Proceedings of the VLDB Endowment, 2013, 6(3): 217-228.

[26] Gao J, Qiu H, Jiang X, et al. Fast top-k simple shortest paths discovery in graphs[C]//Proceedings of the 19th ACM International Conference on Information and Knowledge Management. Toronto: ACM, 2010: 509-518.

[27] Geisberger R, Sanders P, Schultes D, et al. Contraction hierarchies: Faster and simpler hierarchical routing in road networks[M]//Experimental Algorithms. Berlin: Springer, 2008.

(责编 曹东,校对 雷建云)