arXiv:2303.03720v1 [cs.DB] 7 Mar 2023

利用快捷方式查询大型时变道路网络中的最短路径

Zengyang Gong、Yuxiang Zeng、Lei Chen 中国香港特别行政区香港
科技大学 zgongae@connect.ust.hk，{yzengal,leichen}@cse.ust.hk

摘要

查询两个顶点之间的最短路径是许多应用中的一项基本操作，已在静态道路网络中得到广泛研究。然而在现实中，道路路段的出行成本会随时间变化，因此道路网络可以建模为时变图。本文研究大型时变道路网络中的最短路径查询。现有工作采用分层划分结构，导致索引构建和出行成本查询效率低下。为提高此类查询的效率，我们提出一种新颖的索引，将道路网络分解为树结构，并在树上选择一组快捷方式以加速查询处理。具体而言，我们首先正式定义了时变道路网络树分解上的捷径选择问题。该问题已被证明是 NP难，旨在选择并构建最有效的快捷方式集合。我们首先设计了一种能够得出精确结果的动态规划方法来解决捷径选择问题。为快速获得最优快捷方式集合，我们设计了一种保证 0.5 近似比的近似算法。基于这一新颖的树结构，我们设计了一种基于快捷方式的算法，用于回答时变道路网络中的最短路径查询。最后，我们使用大型真实道路网络进行了广泛的性能研究。结果表明，与当前最先进的方法相比，我们的方法在效率和可扩展性方面表现更佳。

1 引言

过去几十年来，由于其广泛的应用，针对静态道路网络中两个地点之间最短距离的查询已得到广泛研究 [5, 14, 18, 19, 21, 22, 33, 34], [26–28, 31]。然而，在真实道路网络中，道路路段的出行成本会随一天中不同时间而变化。例如，城市中央商务区（CBD）附近道路的交通流模式在昼夜之间可能发生显著变化。因此，许多应用在边上设置时变出行成本函数来表示出行成本，从而引出了对形式化时变图上最短路径查询的研究 [29]。

一方面，一种直接的方法是使用非索引方法。*Dijkstra-* 或 *A**- 算法会在遍历过程中访问边的旅行成本函数。因此，对于每次查询，这些方法都需要数秒以上才能得到最终结果。例如，TWO-STEP-LTT算法[11] 在包含 16,326 个顶点和 26,528 条边的道路网络上回答查询约需 10 秒。非索引方法不尽如人意的处理性能难以满足最短路径查询的实时要求，尤其是在时变道路网络规模庞大且查询顶点之间的出行模式快速变化时。

另一方面，人们提出了基于索引的技术来弥补查询性能的不足。TD-G-tree [29] 是基于索引的算法系列中最先进的解决方案之一，且性能显著优于其他方法。直观而言，TD-G-tree 以分层方式对时变道路网络进行划分，每个树节点对应一个分区并维护一个旅行时间矩阵，矩阵中缓存该分区所有边界点对之间的最短旅行成本函数。执行查询时，TD-G-tree 以自底向上的汇总方式遍历缓存的边界信息。显然，TD-G-tree 的性能很大程度上取决于访问的树节点数量以及每个节点关联的边界点数量。然而，如果查询顶点在道路网络中彼此接近、但在索引中相距甚远，汇总式方法的表现甚至可能不如基本的 Dijkstra算法 [19]。此外，TD-G-tree 还存在数据冗余和构建成本高的问题。由于采用分层分区结构，某个树节点显然是其父节点的子图，而且每个分区边界点对的最优旅行成本函数未必总是在同一个分区内。考虑到现实世界的道路网络通常包含数百万条边，同时出行模式频繁变化，TD-G-tree 在大规模时变道路网络中可能效率低下。例如，在我们的实验中，TD-G-tree 在包含 1,070,376 个顶点和 2,712,798 条边的佛罗里达州道路网络上构建索引约需 4 小时，回答查询则需 0.5 秒以上。

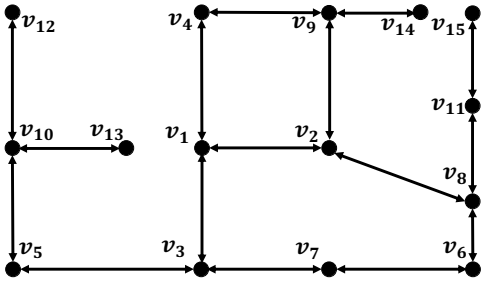
在本文中，为解决现有方法的不足，我们研究将基于树分解的索引应用于时变道路网络，以高效回答最短路径查询。基于树分解的方法在静态道路网络上查询最短路径方面取得了巨大成功 [5, 21, 32],，因为它能够将道路网络分解为树状结构，同时即使道路网络非常大，树高和树宽也都很小。

然而，将树分解技术扩展到时变道路网络并非易事。最大的挑战在于，如何在每个树节点与其若干祖先节点之间选择并构建快捷方式。由于存储快捷方式的成本相当高，我们需要谨慎选择快捷方式，以最大限度地利用有限的内存空间。本文首先证明，在树分解上进行快捷方式选择的问题是 NP难。接着，我们旨在选择并构建最有效的快捷方式，以加速树分解上的查询处理。最后，我们设计了具有理论保证的解决方案，并提出了一种基于选定快捷方式的新型最短路径查询算法。

我们将本文的贡献总结如下：

表 1：主要符号概述。

符号	描述
$G(V,E,W)$	时变道路网络
$w_{u,v}(t)$	边的权重函数$e_{u,v}$
$f_{s,d}(t)$	从 s 到 d的最短旅行成本函数
$Compound()$	复合算子函数
T_G	G 的树分解和树节点
$X(i)$	T_G的树节点
$w(T_G)$ $h(T_G)$	T_G的树宽和树高
$s_{\langle i,j\rangle}(t)$	从 $X(i)$ 到 $X(j)$ 的捷径



(a) 道路网络

边	权重 {(时间, 成本)}
$e_{1,2}$	{(0, 10), (20, 10), (60, 15)}
$e_{2,9}$	{(0, 5), (30, 10), (60, 15)}
$e_{1,4}$	{(0, 5), (30, 15), (60, 25)}
$e_{4,9}$	{(0, 5), (60, 15)}

(b) 权重为 $e_{1,2}$，$e_{2,9}$，$e_{1,4}$ 和 $e_{4,9}$

图 1：时变道路网络。($w_{u,v}(t) = w_{v,u}(t)$)

- 我们应用树分解方法，支持对时变道路网络进行最短路径查询。首先，我们将时变道路网络分解为树结构，同时在这棵树中保留出行成本信息。

- 我们在树结构上构建捷径选择问题，以加速查询处理。在空间成本有限的情况下，该问题会选择最有效的捷径。我们还证明了该问题是 NP难的。

- 为解决捷径选择问题，我们设计了一种基于动态规划的选择算法，并提出了一种有效且高效的近似算法，其近似比为 0.5。

- 基于选定的快捷方式，我们设计了一种新颖的算法来回答最短路径查询，其效率高于最先进的算法。

- 在真实世界的大规模道路网络上进行的大量实验表明，我们的解决方案效率很高，在查询响应方面大幅超越了最先进的算法。

本文其余部分组织如下。我们首先在第 2 节给出问题陈述。随后在第 4 节详细介绍我们的算法，并在第 5 节对其进行评估。第 6 节回顾相关工作，第 7 节总结全文。

2 问题陈述

本节首先定义用于对道路网络建模的时变有向图，随后阐述图中路径的旅行成本函数。最后给出问题定义。主要符号汇总于表 1。

定义1（时变有向图）。设有向图 G(V,E,W) 表示一个时变道路网络，其中 V是顶点集合，E是边集合。我们用 $n = |V|$ 和 $m = |E|$ 分别表示顶点数和边数。具体而言，每条边e_u∈E 都关联一个权重函数 $w_{u,v}(t)$ ∈W，该函数的值表示在时间 t从 u 出发到达v 的出行成本，且该值始终非负。按照先前的工作 [4, 29, 30]，每个函数 $w_{u,v}(t)$ 都由一组插值点 $I_{u,v} = \{(t_1, c_1), (t_2, c_2) \ldots, (t_k, c_k)\}$表示。

那么，与$e_{u,v}$ 关联的权重函数可形式化为

$$w_{u,v}(t) = \begin{cases} c_1, & t = t_1 \\ c_1 + (t - t_1)\frac{c_2 - c_1}{t_2 - t_1}, & t_1 < t \le t_2 \\ \cdots & \\ c_{k-1} + (t - t_{k-1})\frac{c_k - c_{k-1}}{t_k - t_{k-1}}, & t_{k-1} < t \le t_k \\ c_k, & t = t_k \end{cases} \quad (1)$$

在式（1）中，t_1 和 t_k 分别是从该边起点出发的最早出发时间和最后可用的出发时间。

例2.1。 图*1a*展示了一个包含*15*个顶点和*17*条边的时变道路网络。图*1b*展示了从 v_1 到 v_9的路径上四条边的权重函数。对于边$e_{1,2}$，权重函数$w_{1,2}(t)$ 由*3*个（时间，成本）对拟合而成：$I_{\langle 1,2\rangle} = \{(0, 10), (20, 10), (60, 15)\}$。考虑（(0,10)这一对，它表示在时间*0*，从 v_1 到 v_2需要*10*分钟。

定义2（最短旅行成本函数）。考虑一条从 v_1 到 v_k的路径（$e_{1,2}, e_{2,3}, \cdots e_{k-1,k}$），其旅行成本最小；该路径由一系列边$e_{i-1,i} \in E$组成，对所有 $0 \le i \le k$均成立。对于该路径，出发时间为 t时的最短旅行成本函数记为$f_{1,k}(t)$，其递归定义为$f_{1,k}(t) = Compound(f_{1,k-1}(t)\, f_{k-1,k}(t))$，其中$f_{1,k-1}(t)$是从 v_1到 v_{k-1}的子路径的最短旅行成本函数，$f_{k-1,k}(t)$是边$e_{k-1,k}$的权重函数，例如，$f_{v_{k-1},v_k}(t) = w_{v_{k-1},v_k}(t)$。$Compound()$定义为计算相连路径的旅行成本函数的函数，并且函数中还会记录中间顶点，例如，$Compound(f_{1,k-1}(t)\, f_{k-1,k}(t)) = f_{k-1,k}(t + f_{1,k-1}(t))$，并记录中间顶点 $k-1$。

例*2.2*。从 v_1 到 v_9，图*1a*中存在两条路径，*($e_{1,4}, e_{4,9}$)* 和 *($e_{1,2}, e_{2,9}$)*。图*2*展示了这两个顶点之间的复合旅行成本函数。对于 *($e_{1,4}, e_{4,9}$)*，沿此路径的出行成本通过复合这两条边的权重函数计算得出，$Compound(w_{1,4}(t)\, w_{4,9}(t)))$。类似地，路径 *($e_{1,2}, e_{2,9}$)* 的出行成本为 $Compound(w_{1,2}(t)\, w_{2,9}(t)))$。因此，从 v_1到v_9，最短旅行成本函数为 $f_{1,9}(t) = min\{Compound(w_{1,4}(t)\, w_{4,9}(t)), Compound(w_{1,2}(t)\, w_{2,9}(t)))\}$。

问题定义（时变道路网络上的最短路径查询）。给定一个时变道路网络 G、两个查询顶点以及一个特定的出发时间 $Q(s, d, t)$，

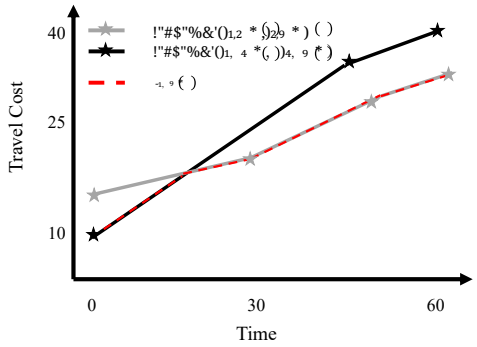


图 2：从 v_1 到 v_9的最短旅行成本函数

最短路径查询问题旨在求出从顶点 s 到顶点 d 在时间 t 出发时的最小成本，以及相应的最短旅行成本函数 $f_{s,d}(t)$。

例*2.3.* 考虑图*1a*中从 v_1 到 v_9 的路径，如图*2*所示的最短旅行成本函数 $f_{1,9}(t)$，我们可以看到，在开始时，最短路径为 $(e_1, 4, e_4, 9)$，中间顶点记录为 v_4；随着时间推移，路径 $(e_1, 2, e_2, 9)$ 的出行成本远低于之前的路径。因此，从 v_1 到 v_9 的最短路径将变为 $(e_1, 2, e_2, 9)$，并且中间顶点在 $f_{1,9}(t)$ 中更新为 v_2。

3 基于 OURTREE 分解的最短路径查询算法

本节首先介绍时变道路网络上的树分解方法，该方法能够高效地将顶点组织成树结构。然后，基于树分解的性质，我们提出一种基于树分解的基本查询算法，用于时变道路网络上的最短路径查询。

3.1 树分解

树分解 [24] 是解决静态图相关问题的高效方法 [5, 21, 23, 32], , 因为它可以将大规模图分解为树状结构。其形式化定义如下。

定义 3（树分解）。时变图 $G(V,E,W)$的树分解记为 T_G，它是一棵有根树，其树节点集为 $\{X(v_1)\ X(v_2)\ \ldots X(v_n)\}$。对于图中的每个顶点 v，都有一个记为 $X(v)$ 的树节点，且每个树节点都是 V 的子集（即，$X(v) \subseteq V$）。因此，T_G 满足以下性质：

(1) $X(v_1) \cup X(v_2) \cup \ldots X(v_n) = V$。 (2) $\forall e_{u,v} \in E, \exists X(v_i)$，使得 $u \in X(v_i)$ 且 $v \in X(v_i)$。 (3) $\forall v_i \in V$，$\{X(v_i) | v \in X(v_i)\}$ 是 T_G 的连通子树。

对于给定的图，构建树宽最小化的树分解是一个 NP完全问题 [23]。因此，本文采用一种次优树分解 [24] 方法。该方法的时间复杂度为 $O(n \cdot (w^2 + logn))$，已被用于多项静态道路网络 [21, 23, 32]研究中。在每次迭代中，算法贪心地选取度最小的顶点，并创建相应的树节点。处理完所有顶点后，即可得到给定图的树分解

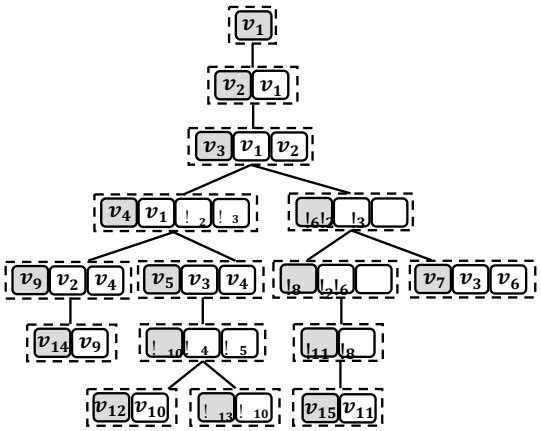


图3： G**的树分解**

因此，图的顶点与树节点之间存在一一对应关系。给定一个顶点 v，$X\ v$ ( ) 表示其在 T_G 中对应的树节点。对于每个树节点$X(v)$，$height(X(v))$ 和 $Anc(X(v))$ 分别表示树分解中 $X(v)$ 的高度和祖先节点集合。

例 *3.1*。图*3*展示了图*1a*中道路网络的树分解 T_G。T_G 有*15*个节点。在每个消去步骤中，我们选取度最低的顶点，然后创建一个包含其邻居的对应树节点。消去顺序为$\{v_{15}, v_{11}, v_{14}, v_{13}, v_{12}, v_{10}, v_9, v_8, v_7, v_6, v_5, v_4, v_3, v_2, v_1\}$。例如，考虑顶点 v_{10}，其对应的树节点是 $X(v_{10}) = \{v_{10}, v_4, v_5\}$。$X(v_{10})$ 的祖先节点集合为 $Anc(X(v_{10})) = (v_1, v_2, v_3\ v_4\ v_5 \quad v_{10} \quad , ,)$，树节点 $height(X( ))$的高度为 = 6。

定义 4（树宽和树高）。给定一棵生成树T_G，树宽和树高分别记作 $w(T_G)$和 $h(T_G)$。树宽 $w(T_G)$比 T_G中的所有树节点的最大规模小1，$w(T_G) = \max_{X(v) \in T_G} |X(v)| - 1$。树高 $h(T_G)$是 T_G 中所有树节点的最大高度，$h(T_G) = \max_{X(v) \in T_G} height(X(v))$，其中高度 $height(X(v))$表示 $X(v)$到根节点的距离。

示例 3.2。对于图 *3* 所示的树分解，T_G 的树宽($w(T_G)$) 为 *3*，因为树节点 $X(v_4)$ 最多包含 *4* 个顶点；树高 $h(T_G)$ 为 *7*，因为树节点 $X(v_{12})$、$X(v_{13})$ 和 $X(v_{15})$ 的高度最大，均为 *7*。

3.2 使用 TFP-图构建树

为了设计基于树结构的查询算法，我们首先给出下文中保留的最短旅行成本函数图的定义。

定义 5（TFP-图）。给定一个时变图$G(V, E, W)$，若图 $G'(V' E' W')$ 满足 $V' \subseteq V$，且对于任意一对顶点 $u \in V'$ 和 $v \in V'$，当 $e_{u,v} \in G'$ 关联的权重函数 $w'_{u,v}(t) = f_{u,v}(t)$；否则，$f'_{u,v}(t)$ 表示图 G' 中从 u 到 v 的路径的最短旅行成本函数，且 $f'_{u,v}(t) = f_{u,v}(t)$。这里，我们用 $G' \sqsubseteq G$表示 G' 是原图 G 的 TFP-图。

算法1： 约简算子 $\ominus$

输入： 时变图 $G(V,\ E,\ W)$ 和顶点 v

输出： TFP-图 G_v

$G_v \leftarrow G$；

对于 $v_i \in N(v)$ **执行**

  对于 $v_j \in N(v)$ **执行**

    如果 $i \neq j$ 且 $e_{i,\ j},\ e_{j,\ i} \notin G$ **则** 插入两条边 $e_{i,\ j}$, $e_{j,\ i}$ 及关联权重函数

    $w'_{i,j}(t) \leftarrow Compound(w_{i,v}(t)\, w_{v,j}(t))$,

    $w_{v,j,\ i}(t) \leftarrow Compound(w_{j,\ v}(t)\, w_{v,\ i}(t))$ 到 G_v；

  else

    将 $w_{i,\ j}(t)$ 和 $w_{j,\ i}(t)$ 在 G_v 中更新为

    $w'_{i,j}(t) \leftarrow min\{Compound(w_{i,\ v}(t)\, w_{v,\ j}(t))\, w_{i,\ j}(t)\}$

    $w'_{j,i}(t) \leftarrow min\{Compound(w_{j,v}(t)\, w_{v,i}(t))\, w_{j,i}(t)\}$；

返回 $G_v(V_v, E_v, W_v)$;

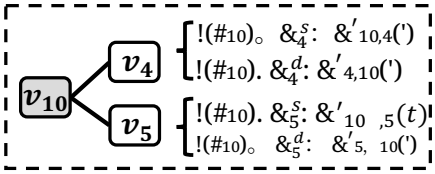


图4： X**中保留的旅行函数**（v_{10}）

时变图约简。令操作 $G \ominus v$ 表示时变图 G中顶点 v 的图约简算子。算法1展示了该 $\ominus$ 算子的具体算法。该操作从 G 中移除 v，并处理与该顶点相关联的边，因此将 G 转换为另一个 TFP-图 G'。具体步骤如下。给定图 G和顶点 v，令 $N(v)$表示 v的邻接顶点。对于每一对邻居 v_i、$v_j \in N(v)$，如果 G中不存在边e_i、$_j$和e_j、$_i$，则构建并插入以v 为桥接顶点的新边e_i、$_j$和e_j、$_i$到 G'中。因此，

$e_{i\,j}$ $e_{i\,v}$ $e_{v\,j}$ $w_{i\,j}$ =的 $Compound\ w_{i\,v}\ t\ \ w_{v\,j}\ t$ $e_{i\,j}$

权重函数计算为，以及，，′，(t)（，（），（））。否则，将边，的权重更新为 $w'i$、$_j(t) = min\{Compound(w_{i,v}(t)\, w_{v,j}(t))\, w_{i,}$ $_j(t)\}$。对于e_j、$_i$，其方向与e_i、$_j$不同，$w'j$、$_i(t) = Compound($ $w_{j\,v}$ $w_{v\,i}$ $e_{j\,i}$ $w_{j\,i}$ =、(t)、min (t))；如果，不是 G'中的边，^{t}则′，(t)$^t\{Compound(w_{j,v}(\,)\, w_{v,i}(\,))\, w_{j,}$ $_i(\,)\}$。

旅行函数保持树分解。对于时变图，我们希望将顶点组织成第 3.1 节介绍的树结构，同时保留 v与 X（v）中其他各个顶点之间的权重函数。例如，图 4 展示了树节点 X（v_{10}）中保留的权重函数。这里有两个函数列表 X（v_{10}）$.W^s$和 X（v_{10}）$.W^d$。列表 X（v_{10}）$.W^s$保留了从 v_{10}到 v_4和v_5的两个权重函数。列表 X（v_{10}）$.W^d$则保留了从 v_4和 v_5到 v_{10}的权重函数。我们可以对每个顶点应用图约简算子，直到所有顶点都被移除，以实现这一目标。我们将这一过程称为旅行函数保持（TFP）树分解。

算法2： TFP树分解

输入： 时变图 $G(V,\ E,\ W)$

输出： TFP树分解 T_G

$G_v \leftarrow G$；$T_G \leftarrow \emptyset$；

对于 $i \leftarrow 1$ 到 n **执行**

  $v \leftarrow$ 选取 G_v 中度最小的顶点；

  $X(v) \leftarrow$ 添加 v；

  对于 $u \in N(v)$ **执行**

    $X(v) \leftarrow$ 添加 u；

    $X(v).W_u^s \leftarrow w'_{v\,,u}(t)$, $X(v).W_u^d \leftarrow w'_{u\,,v}(t)$;

  $G' \leftarrow G' \ominus v$；

  $\pi(v) \leftarrow i$；

对 $v \in V$ **执行**

  若 $|X(v)| > 1$，**则**

    $u \leftarrow X(v)\backslash\{v\}$ 中顺序最小的顶点在 $\pi(u)$ 中给出的值；

    节点 $X(u)$ 被设为 T_G 中 $X(v)$ 的父节点；

返回 T_G;

算法 2 详细介绍了这一过程。在第 1 行，算法 2 在每次迭代中选取度最小的顶点 v，并构建相应的树节点 X（v）。X（v）不仅包含顶点 v，还包含第 4–6 行中的所有邻接顶点。此外，第 7 行中，每个树节点 X（v）还存储与出行成本相关的信息；这里使用列表 X（v）$.W^s$保留从 v到 X（v）中其他顶点的权重函数。另一个列表 X（v）$.W^d$则保留从 X（v）中的顶点到 v的权重函数。随后，在第 8 行，我们对 v 应用时变图约简算子，以移除该顶点并更新与之关联的边；第 9 行则在 π（v）中记录 v的处理顺序。基于顶点的消去顺序，第 10–13 行构建树结构。对于每个树节点 X（v），如果 $|X$（v）$|$大于 1，则 X（v）不是根节点。当 X（v）是非根节点时，如果 u 属于 X（v）$\backslash v$ 且具有最小的 π值，我们就将 X（u）指定为 X（v）的父节点。

复杂度分析。对于时间复杂度，给定算法2第8行中任意选取的顶点 v，算法1会调用一次约简算子。对于每个约简算子，算法1第2–3行需要 $O(w(T_G)^2)$ 时间来处理邻接顶点，随后第4行和第6行中的 $Compound()$ 函数需要 $O(c)$ 时间来计算新的权重函数，其中 c 是一个常数参数，用于表示边对 $(time,\ cost)$ 的平均数量。因此，算法2第8行中一次约简算子的时间成本为 $O(w(T_G)^2 \cdot c)$。算法2第2–4行需要 $O(m + n \cdot log(n))$ 时间来选取度最小的顶点，因此算法2的总体时间复杂度为 $O(n \cdot (w(T_G)^2 \cdot c + log(n)))$。对于空间复杂度，每个权重函数占用 $O(c)$ 空间，每个树节点维护 $O(w(T_G))$ 个权重函数，因此算法2的空间复杂度为 $O(n \cdot w(T_G) \cdot c)$。

3.3 基本查询算法

本节中，我们将介绍基于时变

图的旅行函数保持树分解的基本查询算法。在介绍具体细节之前，我们先介绍该树结构所具有的若干性质。

性质 1 如果 X 是 T_G中 $X(s)$和 $X(d)$ 的最近公共祖先（LCA），那么 X是 G中 s和 d的一个顶点割。

参见 [21]中的性质 1 和定义 4.7。给定树分解 T_G，如果 G中有两个查询顶点 s和 d，假设在 T_G中，与 $X(d)$关联的树节点 $X(s)$既不是其祖先节点，也不是其后代节点，令 VC表示 T_G中 $X(s)$和 $X(d)$的最近公共祖先（LCA），那么 VC是 G中 s和 d的一个顶点割。顶点割 VC表示，从 s到 d的任意路径都必须至少包含 VC中的一个顶点。因此，我们可以计算从 $min_{v\in VC}\ Compound\ f_{s\,v}\ t\ \ f_{v\,d}\ t$

到 d的最短旅行成本函数：$f_{s,}(t)\ \{\ (.(\ ).(\ ))\}$。

性质 2 对于任意树节点 $X(v)\in T_G, X(v)\backslash\{v\}\subseteq Anc(X(v))$。

参见 [21] 中的性质 2。给定一个树分解 T_G 及其每个树节点 $X(v)\in T_G$，对于任意 $u\in X(v)\backslash\{v\}$，$X(u)$ 是 $X(v)$ 的祖先。

性质 3 对于任意顶点 $v\in V$，令 $G(v)$表示包含从 $X(v)$到 T_G根节点的顶点的图，且 $G(v)\sqsubseteq G$。

参见 [21] 中的定义 6.6 和引理 6.8。给定一个顶点$v\in V$，$G(v)$ 是 G 的子图，且同时保留最短旅行成本。因此，如果 $\forall s,\ d\in G(v),\ \ f\prime_{s,\ d}(t)$ 是 $G(v)$ 中 s 到 d 之间的最短旅行成本函数，我们可以得到 $f\prime_{s,\ d}(t)=f_{s,\ d}(t)$。

示例 *3.3*。回到前一个示例图 *3*，对于查询处理 $Q(v_{12},v_{15},t)$，我们首先在 T_G 中定位相关的树节点 $X(v_{12})$ 和 $X(v_{15})$。考虑 $X(v_{12})$，它维护了到 $X(v_{12})\backslash\{v_{12}\}\ =v_{10}$ 的权重函数，而 v_{10} 是 $Anc(X(v_{12}))=(v_1,v_2,v_3,v_4,v_5,v_{10})$ 中的一个祖先节点。对于 $X(v_{12})$ 和 $X(v_{15})$，最近公共祖先是 T_G 中的 $X(v_3)$，因此 $X(v_3)=\{v_3,v_1,v_2\}$ 是 v_{12} 与 v_{15}，$f_{12,15}(t)=\ min_{u\in\{3,1,2\}}Compound(f_{12,u}(t)\ f_{u,15}(t))$ 之间的顶点割。为了计算 $f_{12,u}(t)$ 的最短旅行成本函数，其中 $u\in\{3,1,2\}$，可以直接在 $G(12)$ 上应用传统的基于 *Dijkstra*算法的算法。原因在于，$G(12)$ 是一个 *TFP*-图，它保留了从 $X(v_{12})$ 到其所有祖先节点的最短旅行成本。类似地，我们可以计算从 $X(v_{15})$ 到顶点割中所有顶点的最短旅行成本函数。

根据性质 1-3，我们已准备好在 T_G 上设计查询算法。查询算法的具体细节如算法 3 所示。给定查询 $Q(s,d,t)$，该算法返回从源点 s 到目的地 d 的最短旅行成本函数，其时间参数为 t。第 1 行初始化列表

$cost_s[]$，该列表维护从源点 s到 G中的顶点（s）的最短旅行成本函数。我们在 G（s）上应用基于修改版 Dijkstra算法的方法。它在第 4-8 行中遍历 T_G，从 X（s）到树的根节点。具体而言，对于节点X（v），我们在第 5 行检查其维护的、指向 X（v）\{v} 的权重函数。如果顶点 u在第 8 行处理 X（v）之前尚未被访问，则可将 v视为桥接顶点，计算从 s到 u的最短旅行成本函数。否则，我们在第 7 行检查是否可以通过 v获得更短的出行成本。然后，我们在 G（d）上重复相同的过程，以计算列表 $cost_d[]$；该列表缓存了从 G（d）中的所有顶点到 d的最短旅行成本函数。最后，根据顶点割性质，我们得到 X（s）和 X（d）的 LCA，即第 11 行中的 VC；通过检查指向集合 VC中顶点的已存储最短旅行成本函数，我们可以在第 12-13 行计算出最短旅行成本函数的结果。

算法 3： 基本查询算法

输入： 树分解 T_G和查询 $Q(s,d,t)$

输出： 旅行成本函数 $f_{s\ \ ,d}(t)$

1 **对于** $w\in G(s)$ **则** $cost_s[w]\leftarrow+\infty$；

2 **对于** $u\in X(s)\backslash\{s\}$，$cost_s[u]\leftarrow X(s).W_u^s$；

3 $v\leftarrow X(s)$ 的父节点；

4 **当** $v\neq T_G.root$ **时**

5 　**对于** $u\in X(v)\backslash\{v\}$ **执行**

6 　　**如果** $cost_s[u]\neq+\infty$，**则**

7 　　$cost_s[u]=$ $min\{cost_s[u],Compound(cost_s[v],X(v).W_u^s)\}$；

8 　　**else**

9 　　$cost_s[u]=Compound(cost_s[v],X(v).W_u^s)$；

10 　$v\leftarrow X$的父节点（v）；

11 重复第 1-8 行，以计算 $cost_d[\cdot]$ 在 $G(d)$ 上的值；

12 $VC\leftarrow LCA(X(s)\ X(d))$；

13 **对于** $w\in VC$，$f_{s,\ w(t)}=cost_s[w]$，$f_{w,\ d}(t)=cost_d[w]$；

14 $f_{s,\ d}(t)=min_{w\in VC}\{Compound(f_{s,\ w}(t)f_{w,\ d}(t))\}$；

15 **返回** $f_{s\ \ ,d}(t)$;

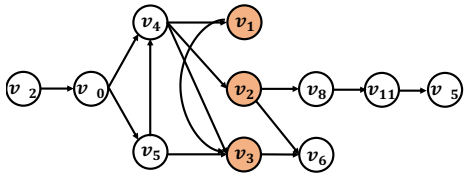


图 5： v_{12} **与** v_{15}**之间的查询处理**

时间复杂度。查询算法在第 3-9 行中从下至上遍历整棵树，耗时 O（h（T_G））。对于每个树节点 X（v），其包含的权重函数至多为 2·w（T_G）个，因此每个树节点需要 O（w（T_G））的时间来计算指向 X（v）中顶点的最短旅行成本函数。因此，查询算法的总时间复杂度为 O（h（T_G）· w（T_G）· c），其中 $Compound()$ 函数的时间复杂度为 O（c），而 c为常数因子。

示例*3.4*。考虑查询 Q（v_{12}、v_{15}、t），图*5*展示了基于 T_G在 v_{12}和 v_{15}之间的处理过程。标记为黄色的顶点是查询顶点之间的顶点割。左侧展示了在图 *G(12)*上的改进的基于*Dijkstra*的算法处理过程，右侧则展示了在图 *G(15)*上的处理过程。考虑左侧，开始时，源点 v_{12}到顶点 v_{10}的最短旅行成本函数存储在$cost_{12}[v_{10}]=X$（v_{12}）$.W_{10}^s$中。接下来，我们可以得到 X（v_{12}）的父节点为 X（v_{10}）。由于 X（v_{10}）包含两个尚未访问的顶点，因此我们可以得到 $cost_{12}[v_4]=Compound$（$cost_{12}[v_{10}]$、X（v_{10}）$.W_4^s$）和 $cost_{12}[v_5]=Compound$（$cost_{12}[v_{10}]$、X（v_{10}）$.W_5^s$）。此外，X（v_5）是 X（v_{10}）的父节点，其中包含一个已访问的顶点v_4和另一个顶点 v_3。对于 v_4，我们可以将最短旅行成本函数更新为 $cost_{12}[v_4]=min\{cost_{12}[v_4]$、$Compound$（$cost_{12}[v_5]$、$X$（$v_5$）$.W_4^s$）$\}$，该函数尝试通过 v_5寻找更短的路径。算法会一直运行，直到到达 T_G的根节点。我们对 X（v_{15}）重复相同的过程。然后，我们可以得到最短旅行成本函数

分别从源点和目的地到顶点割的最短旅行成本函数，并计算出最终结果。

4 捷径选择

本节首先定义 T_G 上的捷径，接着形式化捷径选择问题，然后证明该问题是 NP-hard 的。我们首先使用基于动态规划的方法，在第 4.3 节中求解捷径选择问题。然后，在第 4.4 节中，我们提出一种近似算法，可保证 0.5 的近似比。最后，在第 4.5 节中，我们提出一种基于选定快捷方式、效率更高的查询算法。

4.1 初步

在算法 3 中，算法频繁地自底向上遍历树结构，这会严重降低查询效率。直观而言，基于上一节介绍的 T_G 的性质 1–3，如果为每个 $X(v_i)$ 构建一组捷径 $\{s_{\langle i,j\rangle}(t)|v_j \in Anc(X(v_i))\}$，就可以跳过遍历。这组捷径由 T_G 中 $X(v_i)$ 与其所有祖先节点之间的最短旅行成本函数组成。然而，由于主存空间有限，为 T_G 中每个树节点构建并维护其与所有祖先节点之间的所有快捷方式是不可行的。我们将在实验部分展示相关结果。因此，我们需要明智地选择捷径，以便在受限内存空间下提高查询效率。

定义 6（捷径）。给定一个树节点 $X(v_i) \in T_G, Anc(X(v_i)) = \{l_1, l_2 \ldots l_m\}$ 是 $X(v_i)$ 的祖先节点列表，并根据 $height(X(v_j))$ $v_j \in \{l_1, \ldots l_m\}$ 按递增顺序排列。根据性质 2，对于任意 $v \in X(v_i)\backslash\{v_i\}$，$v \in Anc(X(v_i))$。为便于表述，给定一个树节点 $X(v_i)$ 及其一个祖先节点 $X(v_j)$，一对 i, j 表示这两个树节点之间存在快捷方式。对于一对实例 i, j，有两条快捷方式 $s_{i,j}(t)$ 和 $s_{j,i}(t)$ 被选中并构建于 T_G 中。$s_{i,j}(t)$ 是从 $X(v_i)$ 到 $X(v_j)$ 的最短旅行成本函数，而 $s_{i,j}(t)$ 是从 $X(v_j)$ 到 $X(v_i)$ 的最短旅行成本函数。每条捷径均建模为分段线性函数。

定义 7（效用值与权重）。一个快捷边对实例 i，j 的效用值表示为 $u_{\langle}$ $_{i\ j}$ $-$ $_{,}$ $_{\rangle}$ $=$ $height\ X\ v_j$ $w\ T_G$ $p_{\ i\ j}$ $(height(X(v_i))\ (\ (\ ))\cdot(\ )\cdot\langle,\ \rangle)$，用于衡量选择并构建快捷方式 $s_{\langle i,\ j\rangle}(t)$ 对查询带来的收益。具体而言，$p_{\langle i,\ j\rangle}$ 表示快捷边对实例 i，j 提高查询效率的概率，可定义为

$vertexes_k,$

$|V|$ ^{j} $\forall v_k \in, , , (\ (\ )$

$vertexes_{k\ j}\ LCA\ X\ v_i\ \ X\ v_k\ \ = X\ v_j$

$(\ ))\ (\ )$. 对于选定的快捷边对实例 i, j，捷径 $s_{i,j}(t)$ 和 $s_{j,i}(t)$ 也被建模为分段线性函数。如前所述，缓存了两组插值点 $I_{i,j} = \{(t_1, w_1), (t_2, w_2) \cdots (t_z, w_z)\}$ 和 $I_{j,i} = \{(t'1, w'1), (t'2, w'2) \cdots (t'z, w'z)\}$，分别用于表示这两条捷径。该快捷边对实例的权重为 $(|I_{i,j}| + |I_{j,i}|)$。

示例 4.1. 以图 3 中的 T_G 的快捷边对实例 $\langle 12, 3\rangle$ 为例。在图 6 中，展示了 $X(v_{12})$ 与 $X(v_3)$ 之间的快捷方式。我们首先假设，从 v_{12} 到 v_3 的最短旅行成本函数由 3 个（时间，权重）对表示，而从

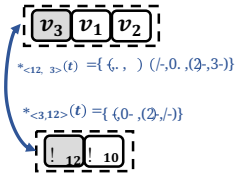


图 6： X（v_{12}）**与** X（v_3）**之间的捷径**

v_3 到 v_{12} 的最短旅行成本函数由 2 个（时间，权重）对表示。因此，快捷方式 $\langle 12, 3\rangle$ 的权重为 5。对于效用值，$\{v_{15}, v_{11}, v_8, v_6, v_7\}$ 与 $X(12)$ 有相同的最近公共祖先节点 $X(3)$，换句话说，快捷方式 $\langle 12, 3\rangle$ 有助于提高从 v_{12} 查询这 5 个顶点的效率。因此，我们可以得到 $p\langle 12,3\rangle = \frac{1}{3}$。此外，这条快捷方式最多可以避免 $(height(X(12))$ $-$ $height\ X\ 3$ $w\ T_G$ $(\ (\ )))\cdot(\ )$ 次分段线性函数复合运算，因此效用值 $u\langle 12,3\rangle = 3\cdot 4\cdot 13$。

对于每个树节点 $X(v_i)$，根据定义 5，$G(v_i)$ 包含 T_G 中 $X(v_i)$ 的所有祖先节点。为收集 T_G 中的所有捷径，对于每个树节点 $X(v_i)$，我们需要先构建图 $G(v_i)$，然后可以采用基于 $Dijkstra$ 的方法，计算 $G(v_i)$ 中 v_i 到其他顶点的最短旅行成本函数。然而，为每个树节点单独计算捷径会消耗大量计算资源。根据性质 2，$X(v)$ 是 $X(v_i)$ 的祖先节点，其中 $\forall v \in X(v_i)\backslash v_i$，且 v 和 v_i 之间的最短旅行成本函数在 $X(v_i)$ 中得以保留。此外，参见 [21], 中的引理 6.11，如果 $v_j \in Anc(X(v_i))$，那么 $G(v_j)$ 是 $G(v_i)$ 的超图。因此，可以采用自顶向下的方式计算 T_G 的快捷方式集合。具体而言，在计算 $X(v_i)$ 的捷径时，可以重用 $X(v_i)\backslash v_i$ 中节点的捷径。给定一个树节点 $G(v_i)$，下面介绍快捷边构建引理：

事实 1（引理 6.11 [24]）。对于任意 $v_j \in Anc(X(v_i))$，我们计算 $X(v_i)$ 的快捷方式如下：

$s_{i,j}(t) = min_{v\in X(v_i)\backslash\{v_i\}}\{Compound(X(v_i).W_v^s, s_{v,j}(\{v21)))\}$

$s_{j,i}(t) = min_{v\in X(v_i)\backslash\{v_i\}}\{Compound(s_{j,v}(t)\ X(v_i)\ \ .W_v^d))$

给定一个树分解 T_G，捷径选择问题是在内存空间有限的条件下，选择一组“最佳”的树节点及其祖先节点对。正如上一节所述，“最佳”的快捷方式集合应能最大程度地提高查询效率。换言之，我们希望在特定权重约束下，选择一组效用值最高的快捷方式。

定义8（捷径选择）。给定 T_G中的所有可能捷径集合，$\mathbb{S} = \{i, j|X(v_i) \in T_G,\ v_j \in Anc(X(v_i))\}$。令$x_{i,\ j} \in \{0,\ 1\}$表示一个快捷边对实例$i$，$j$是否被选中；若$x_{i,\ j} = 1$，则表示选择了 $X(v_i)$ 与 $X(v_j)$之间的快捷方式，反之亦然。因此，捷径选择问题可表述为：

$$\max \sum_{\langle i,j\rangle \in \mathbb{S}} u_{\langle i,j\rangle} \cdot x_{i,j}$$

$$\text{subject to.} \sum_{\langle i,j\rangle \in \mathbb{S}} (|I_{\langle i,j\rangle}| + |I_{\langle j,i\rangle}|) \cdot x_{i,j} \leq N$$

4.2 捷径选择的NP难性

在本小节中，我们证明捷径选择问题的NP难性。

定理 1. 捷径选择问题是NP难的

证明 我们通过将0-1 背包问题归约为捷径选择问题来证明该定理。0-1 背包问题可以形式化如下。给定一个集合 $C = 1、2、\cdots n$，其中有 n个编号从1到 n的物品，每件物品 i对应一个价值y_i和一个权重 w_i。在最大重量容量W的约束下，0-1 背包问题旨在找出一个子集 C, of C，使总价值最大：$\max \sum_{i \in C'} y_i$，约束条件为 $\sum_{i \in C'} w_i \leq W$

对于给定的0-1 背包问题实例，可以按如下方式构造捷径选择问题实例。给定一个时变道路网络 $G = (V, E, F)$，我们在多项式时间内生成其上的生成树 Tover G，并进一步重新编号顶点。我们将 v_i映射到树的第 i层，因此 v_0是叶节点，$v_{|V|}$是根节点（例如，对于图1a中的图，我们可以轻松生成一棵高度为15的树，因为它有15个顶点，其中叶节点为 v_0，根节点为 v_{15}）；对于 v_i，v_{i+1}是父节点，并且可以为其祖先{v_{i+2}、$\cdots$ $v_{|V|}$} 构建潜在捷径。然后，我们可以得到 $\sum_{0 \leq i \leq}$（$_{|v|-1}$） i 个可能的快捷边对实例，使得对于每个快捷边对实例i, j，效用值$u_{i,j} = y_i$，捷径实例权重（$|I_{i,j}| + |I_{j,i}|$） $= w_i$。此外，内存开销约束$N = W$。因此，对于该捷径选择实例，我们希望获得一个最优选择集合 $\mathbb{I}$，使总体效用最大化：$\max \sum_{i,j \in \mathbb{I}} u_{i,j}$，约束条件为$\sum_{i,j \in \mathbb{I}} (|I_{i,j}| + |I_{j,i}|) \leq N$。

根据该映射，我们推导出，当且仅当转换后的捷径选择问题可解时，0-1 背包问题实例可解。

因此，捷径选择问题可由 0-1 背包问题归约而来，并且它是一个NP难问题。□

4.3 动态规划算法

为搜索所有可能的快捷方式，在本小节中，我们首先说明基于动态规划的选择算法的细节，并分析其时间复杂度。

在描述该算法之前，我们首先介绍一些有用的记号。我们使用 $U(i, l_k\ w)$ 表示在考虑实例 i，l_k 且权重约束为 w 时，选定快捷方式的最大效用值。实例 i，l_k 表示 $X(v_i)$ 与其在 $Anc(X(v_i))$ 中第 k 个祖先节点之间的快捷方式。令根节点 表示树分解 T_G 的根节点，该算法迭代地将每个树节点及与其相关联的祖先节点加入捷径选择集合，直至 $root$。因此，我们将状态转移规则定义为：

$$U(\langle i, l_k \rangle, w) = \begin{cases} u_{\langle i, l_k \rangle} + U(\langle i, l_{k-1} \rangle, w - (|I_{i,l_{k-1}}| + |I_{\langle l_{k-1}, i \rangle}|), \textit{case 1} \\ U(\langle i, l_{k-1} \rangle, w), \textit{case 2} \end{cases} \tag{2}$$

在公式（2）中，如果在情况 1 下将 $U(i, l_k\ w)$ 更新为 $u_{\langle i, l_k \rangle} + U(i, l_{k-1}\ w - (|I_{\langle i,l_{k-1} \rangle}| + I_{\langle l_{k-1},i \rangle}|))$，则表明，如果选择快捷边对实例 i, l_k，最大效用值可以更新为更高的值。否则，在情况 2 下，最大效用值与前一次迭代中的值相同，在检查 v_i 与其第（$k-1$）个祖先节点之间的快捷方式时。因此，在情况 2 的权重约束 w 下，不会选择实例 i, l_k。

算法 4 展示了基于动态规划的捷径选择算法的详细流程。在第 3–4 行中，我们枚举每个树节点及其每个祖先节点，以更新最大效用值。我们将约束值 w从 1 更新到 N，当完成对值 $U(i,\ l_k\ N)$ 的更新后，枚举便停止。在第 6–8 行的每次迭代中，如果该值是根据公式 (2) 中的情形 1 更新的，我们就选取快捷边对实例 i, l_k，并将其放入集合 $\mathbb{S}$。最后，我们返回集合 $\mathbb{S}$作为选定的快捷方式集合。

算法 4： 动态规划选择

输入： 树分解 T_G 和权重约束 N
输出： 选定的快捷方式集合 $\mathbb{S}^*$

```
1 𝕊 ← {i, j | X(v_i) ∈ T_G, v_j ∈ Anc(X(v_i))};
2 𝕊* ← ∅;
3 对于 X(v_i) ∈ T_G 执行
4     对于 l_k ∈ Anc(X(v_i)) 执行
5         对于 w ← 1 至 N 执行
6             若式 (2) 中的情况 1 则
7                 快捷边对实例 i, l_k 被选中;
8                 𝕊 ← 𝕊 + i, l_k;
9 返回 𝕊*;
```

复杂度分析。 第 3 行的树节点数量为 $O(n)$。对于每个树节点，我们在第 4 行检查其所有祖先节点，祖先节点的数量受树高 $O(h(T_G))$ 的限制。因此，算法 4 的总体时间复杂度为 $O(n \cdot h(T_G) \cdot N)$。

例 4.2。 回到例 3.1 中的树 T_G。我们首先假设 $N = 100$，并从 $X(v_{15})$ 开始。算法 4 检查从 v_{15}, v_1、$v_{15}, v_2 \ldots$直到 v_{15}, v_{11} 的快捷方式实例，并计算相应的值 $U(v_{15}, v_1, 1)$、$U(v_{15}, v_2, 1) \ldots$、$U(v_{15}, v_{11}, 1)$。然后，我们将值 w 从 1 更新为约束 100。我们以类似的方式检查其他树节点。最后，我们计算值 $U(v_2, v_1,$ 100)，并得到最终选定的快捷方式集合 $\mathbb{S}$。

4.4 近似算法

基本思路。 为了提高索引构建的效率，我们设计了一种同时考虑两种不同贪心策略的近似算法。对于第一种贪心策略，我们按照效用值的顺序考察快捷方式实例，并将实例添加到选择集合中，直到总权重超过约束。对于第二种策略，我们以相同的方式，按照“密度值”（$\frac{u_{\langle i\ j \rangle}}{|I_{\langle i\ \ ,j \rangle}|}$）的顺序考察快捷方式实例。我们

如果独立实施这两种贪心策略，可能会导致性能不佳（例如，近似保证可能差到任意程度）。一方面，从叶节点到根节点的一条捷径可能占用大量空间，但效用却不如从叶节点到附近若干祖先节点的多条捷径。另一方面，从叶节点到根节点的一条捷径，可能使其他与该叶节点具有相同最近公共祖先的顶点所发起的许多查询受益。因此，我们同时考虑这两种贪心策略。我们先分别实施

这两种贪心策略，然后选出效用值最高的选定集合。

算法 5 展示了近似选择算法的详细过程。第 2 行初始化了两个优先队列 Q_1 和 Q_2，其中 Q_1 按效用值排序，而 Q_2 按“密度值”排序。第 3 行中，我们将 $\mathbb{S}^*_1$ 定义为从队列 Q_1 中选出的快捷方式集合，其权重之和为 W_1。相应地，$\mathbb{S}^*_2$ 是从 Q_2 中选出的集合，而 W_2 是 $\mathbb{S}^*_2$ 中快捷方式的总权重。我们首先在第 4 行将所有快捷边对实例收集到两个队列中。然后，分别在第 5–8 行和第 9–12 行实施两种贪心选择策略。最后，我们比较两个选定集合的总效用值，并将效用值较大的集合返回为最终选定集合。

算法 5： 近似选择

输入： 树分解 T_G 和权重约束 N

输出： 选定的快捷方式集合 $\mathbb{S}^*$

1 $\mathbb{S} \leftarrow \{\langle i,j\rangle | X(v_i) \in T_G, v_j \in Anc(X(v_i))\}$；

2 Q_1，$Q_2 \leftarrow$ 空队列按 $u_{\langle i,j\rangle}$ 优先处理，并且 $\frac{u_{\langle i,j\rangle}}{|I_{\langle i,j\rangle}|}$；

3 $\mathbb{S}^*_1, \mathbb{S}^*_2 \leftarrow \emptyset; W_1, W_2 \leftarrow 0$；

4 **对于** $i, j \in \mathbb{S}$，$Q_1.push(i,j)$ $Q_2.push(i,j)$；

5 **当** Q_1 非空且 $W_1 < N$ **执行**

6 $i,j \leftarrow Q_1.pop()$；

7 **if** $W_1 + (|I_{i,j}| + |I_{j,i}|) > N$ **中断**；

8 $\mathbb{S}^*_1 \leftarrow i,j$ $W_1 += (|I_{i,j}| + |I_{j,i}|)$；

9 **当** Q_2 不为空且 $W_2 < N$ **执行**

10 i' $j' \leftarrow Q_2.pop()$；

11 **如果** $W_2 + (|I_{i',j'}| + |I_{j',i'}|) > N$ **中断**；

12 $\mathbb{S}^*_2 \leftarrow i'$ j' $W_2 += (|I_{i',j'}| + |I_{j',i'}|)$；

13 **如果** $\sum_{\langle i,j\rangle \in \mathbb{S}^*_1} u_{\langle i,j\rangle} > \sum_{\langle i',j'\rangle \in \mathbb{S}^*_2} u_{\langle i',j'\rangle}$ **则**

14 **返回** $\mathbb{S}^*_1$；

15 **否则**

16 **返回** $\mathbb{S}^*_2$；

性能分析。在定理 2 中，我们分析了算法 5 的近似比。根据理论结果，该算法应适用于大规模树分解。

定理 2. 算法 5 的近似比为 0.5。P
Rove. We consider two greedy strategies: 算法, 从集合 U_1 和 U_2 分别获得选定集合 $\mathbb{S}^*_1$ 和 $\mathbb{S}^*_2$ 所获得的总效用值，令 OPT 表示选定快捷方式集合的最大效用值。我们定义两个函数 $utility_1(k)$ 和 $utility_2(k)$，这两个函数分别返回两个不同优先队列 Q_1 和 Q_2 中第 k 个快捷边对实例的效用值。对于基于“密度”值的贪心策略，$\mathbb{S}^*_2$ 是选定快捷方式的集合；令 Q_2 中的第 y 个捷径为首个因权重约束 N 而无法放入 $\mathbb{S}^*_2$ 的实例。对于集合 $\mathbb{S}^*_1$，采用的贪心策略会首先将效用值最大的单个快捷边对实例选入集合 $\mathbb{S}^*_1$。因此，$\mathbb{S}^*_1$ 必然包含效用值最大的快捷边对实例，其总效用值必定大于 Q_2 中的第 y 个实例。我们可以得到

$$U_1 \geq utility_2(k)$$

对于集合 $\mathbb{S}^*_2$，我们选择基于“密度”值的策略来选取 Q_2 中的捷径。令 d_y 表示 Q_2 中第 y 个实例的“密度”值。由于第 y 个实例是首个无法放入 $\mathbb{S}^*_2$ 的捷径，因此 $\mathbb{S}^*_2$ 中的其他快捷边对实例，其“密度”值都大于 d_y。

主要观察是，如果我们可以从 y 号实例中截取一部分，在 Q_2 中恰好填满权重约束，那么显然，在允许选择部分实例的情况下，这就是最优解：它使用了所有“密度”值大于 d_y 的快捷边对实例，并用“密度”值等于 d_y 的部分填满剩余权重；所有未选中的其他实例的“密度”值都小于 d_y。这表明，当我们取 Q_2 中第 y 个实例的一部分时，所得值就是最优值。因此，如果我们取 Q_2 中整个第 y 个实例，真正的最优值只会更小：

$$\sum_{k=1}^{|\mathbb{S}^*_2|} utility_2(k) + utility_2(y) \geq OPT$$

根据 U_2 的定义，它是 $\mathbb{S}^*_2$、$U_2 = \sum_{k=1}^{|\mathbb{S}^*_2|} utility_2(k)$ 中的总效用值，因此我们可以得到：

$$U_1 + U_2 \geq \sum_{k=1}^{|\mathbb{S}^*_2|} utility_2(k) + utility_2(y) \geq OPT$$

这意味着 $U_1 + U_2 \geq OPT$，因此 U_1 和 U_2 中较大的一个必须至少为 OPT 的 $\frac{1}{2}$；而具有较大总效用值的选定集合 $\mathbb{S}^*_1$ 和 $\mathbb{S}^*_2$ 必须至少为 OPT 的 $\frac{1}{2}$。 –

最后，我们得出近似比为 0.5。

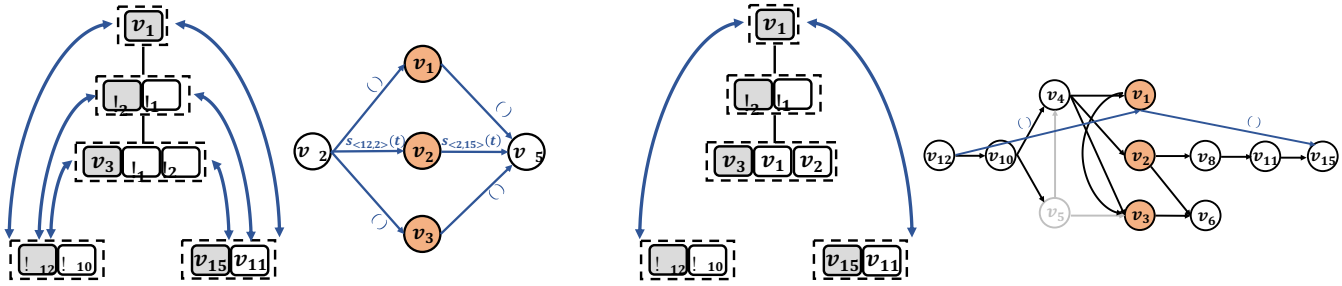
□

复杂度分析。在算法 5 的第 2 行中，创建了两个优先队列，并在整个选择过程中对其进行维护，因此该算法的时间复杂度为 $O(n \cdot h(T_G) \cdot log(nh(T_G)))$。

4.5 使用快捷方式进行查询处理

基本思路。给定查询 $Q(s,d,t)$，根据 T_G 的性质，关键步骤是分别求出从 s 和 d 到 $LCA(s,d)$ 中各顶点的最短旅行成本。基于选定的快捷边对实例集合，有 3 种情况可以利用快捷方式加速查询处理：(1) 如果从 s 和 d 到 $LCA(s,d)$ 中各顶点的快捷方式均已选定，则查询时间可限定为 $O(w(T_G))$；(2) 如果选定了从 s 和 d 到 $LCA(s,d)$ 中各顶点的部分快捷方式，则可利用这些选定的快捷方式计算从 s 到 d 的一个上界最短旅行成本，进而利用该上界值指导对 T_G 的搜索；(3) 如果从 s 和 d 到 $LCA(s,d)$ 中的快捷方式均未选定，则需要调用基本查询算法来回答查询。

算法 6 展示了使用选定捷径集合时的详细查询算法。在第 1–2 行，我们检查从 s 和 d 到其 LCA 中各顶点的所有捷径是否都已在 $\mathbb{S}^*$ 中选定；若是，则可以直接得出最短旅行成本函数：$f_{s,d}(t) = min_{w \in LCA(X(s)\ X(d))} \{Compound(s_{s,w}(t)\ s_{w,d}(t))\}$。此过程耗时 $O(w(T_G))$。然后我们得到 W_s，即从 s 到 LCA 中一个顶点子集的捷径集合。接着，我们从查询顶点 d 得到类似的集合 W_d。如果 W_s 和 W_d 的交集非空，就意味着存在



(a) 通往 LCA的快捷方式　(b) 在 (a) 中的查询处理　(c) 通往 LCA子集的快捷方式　(d) 在 (c) 中的查询处理

图7：使用快捷方式查询从 v_{12} 到 v_{15} 的路径。

算法6： 使用快捷方式进行查询

输入： 查询 $Q(s, d, t)$、T_G以及选定快捷方式 $\mathbb{S}^*$
输出： 旅行成本函数 $f_{s,d}(t)$

1 **如果** $\forall w \in LCA(X(s) X(d))$, s, w 且 w, $d \in \mathbb{S}^*$ **则**
2　$f_{s,d}(t) = min_{w \in LCA(X(s) X(d))} \{Compound(s_{s,w}(t) s_{w,d}(t))\}$;
　　返回 $f_{s,d}(t)$;
3 $W_s \leftarrow \{w | \forall w, s, w \in \mathbb{S}^*\}$, $W_s \subseteq LCA$;
4 **对于** $v \in G(s)$ **执行**
5　**若** $v \in W_s$ **则** $cost_s[v] \leftarrow s_{\langle s,v \rangle}(t)$;
6　**否则** $cost_s[v] \leftarrow \infty$;
7 $W_d \leftarrow \{w | \forall w, w, d \in \mathbb{S}^*\}$, $W_d \subseteq LCA$;
8 **对于** $v \in G(d)$ **执行**
9　**若** $v \in W_d$ **则** $cost_d[v] \leftarrow s_{\langle v,d \rangle}(t)$;
10　**否则** $cost_d[v] \leftarrow \infty$;
11 $f^+_{s,d}(t) = min_{u \in W_s \cap W_d} \{Compound(cost_s[u], cost_d[u])\}$;
12 $X(v) \leftarrow X(s)$ 的父节点, $cost_s[v] = X(s).W^s_v$;
13 **当** $v \neq T_G.root$ **时**
14　**对于** $u \in X(v) \backslash \{v\}$ **执行**
15　　**如果** $u \in W_s$ 或 $cost_s[u] =$ 空值 **则继续**;
16　　**如果** $cost_s[u] \neq \infty$ **则**
17　　$cost_s[u] = min\{cost_s[u], Compound(cost_s[v], X(v).W^s_u)\}$;
18　　**else**
19　　$cost_s[u] = Compound(cost_s[v], X(v).W^s_u)$;
20　　**如果** $cost_s[u] > f^+_{s,d}(t)$ **则** $cost_s[u] \leftarrow$ 空值;
21　$v \leftarrow X$的父节点(v);
22 重复第 12—20 行，以计算 $cost_d[\cdot]$ 在 $G(d)$ 上的值;
23 $f_{s,d}(t) = min_{w \in LCA(X(s) X(d))} \{Compound(cost_s[w], cost_d[w]\}$;
　　返回 $f_{s,d}(t)$;

从 s和 d出发、且至少包含LCA中的一个公共顶点的捷径。因此，我们可以在第 11 行计算从 s到 d的旅行成本上界，即 $f^+_{s,d}(t)$。然后，这个上界可以指导我们在 T_G中的搜索。在第 12–20 行，遍历过程与基本查询类似。不过，当我们检查某个顶点时，如果其旅行成本大于 $f^+_{s,d}(t)$，就将该顶点的旅行成本设为 空值 在第19行。这意味着，在第15行的最终结果中，无需将该顶点检查为内部顶点；如果搜索过程到达出行成本为空值的顶点，我们可以直接跳过该顶点。与基本查询算法类似，在计算出 LCA中每个顶点到 s和 d的最短旅行成本函数后，我们可以在第22行得到最终查询结果。

复杂度分析。 基于选定的快捷方式，查询算法的复杂度上界为 $O(w(T_G))$。

例4.3。图7展示了一个使用选定快捷方式进行查询的示例。给定查询 $Q(v_{12}, v_{15}, t)$，图7a和图7b展示了使用快捷方式从 v_{12} 和 v_{15} 到其 LCA 树节点 $X(v_3)$中的所有顶点的查询处理。然后，图7c和图7d分别展示了使用一条从 v_{12} 到 v_1 的快捷方式以及另一条从 v_1 到 v_{15} 的快捷方式进行的查询处理。在图7b中，这是算法的第一种情况，我们可以直接基于快捷方式得到最终结果$f_{12,15}(t) = min_{w \in \{1,2,3\}} \{Compound(s_{12,w}(t) s_{w,15}(t))\}$。在图7c中，并非所有通往 $X(v_3)$中顶点的快捷方式都被选定并构建。不过，我们可以轻松得到一个出行成本上界$f^+_{12,15}(t) = Compound(s_{12,1}(t)) s_{1,15}(t)))$。图 7d 展示了在 $f^+_{12,15}(t)$ 的引导下进行的搜索过程。从$X(v_{12})$出发，在计算出树节点$X(v_{10})$中到 v_4 和 v_5 的出行成本后，如果满足$cost_{12}[5] > f^+_{12,15}(t)$，我们就将 $cost_{12}[5]$设为空值。无需将顶点 v_5检查为内部顶点，也无需遍历树节点 $X(v_5)$，以计算通过 v_5到 v_3 和 v_4 的出行成本。

5 实验研究

算法。 我们将所提出的算法与时变道路网络查询处理领域的最先进算法进行比较。我们实现并比较了 5 种算法：

- TD-G-tree：用于查询时变图中最短路径的最先进算法 [29];
- TD-H2H：时变 H2H 索引，将 H2H 索引扩展到时变场景 [17];
- TD-basic：基于出行函数保留树分解的基本查询算法（算法 3）；
- TD-dp：采用动态规划策略选择快捷方式的查询算法（算法 4 + 算法 6）；
- TD-appro：采用近似策略选取快捷方式的查询算法（算法 5 + 算法 6）。

表 2：数据集统计数据。

数据集	#(顶点)	#(边)	$h(T_G)$	$w(T_G)$	N
CAL(加利福尼亚州)	21，048	43，386	224	18	10M
SF(旧金山)	321，270	800，172	529	105	20M
科罗拉多州	435，666	1，057，066	511	122	50M
佛罗里达州	1,070,376	2，712，798	706	89	100M
美国西部	6，262，104	15，248，146	1041	386	200M

表3：在 *CAL*上的性能。

	查询成本	构建	内存
TD-G-tree	$0.16ms$	$0.006h$	$0.169G$
TD-H2H	$0.0001ms$	$0.12h$	$3.7G$
TD-basic	$4.4ms$	$0.0002h$	$0.089G$

表4：在 $W-USA$。

	查询成本	构建	内存
TD-G-tree	$30ms$	$15h$	$102G$
TD-H2H	N/A	N/A	N/A
TD-basic	$9,118ms$	$1.18h$	$66G$

实验在一台配备 40 个 Intel(R) Xeon(R) E5 2.30GHz 处理器和 512GB 内存的服务器上进行。这五种查询算法均使用 GNU C++ 实现。按照 [12, 13, 25]，中的设置，每项实验重复 10 次，并报告平均结果。

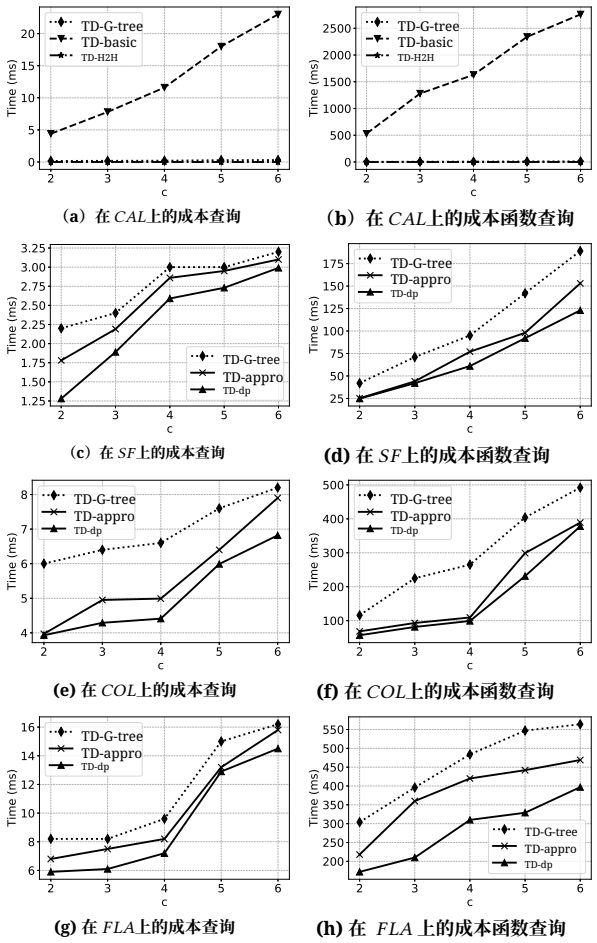
数据集。 我们使用 5 个公开可用的真实道路网络开展实验。这些道路网络是有向图，近期已广泛用于最短路径查询相关研究 [21] [29] [5]。我们将时间域设为一天，即 86400 秒，与 [29] 中的设置相同。我们采用 [17] 中相同的策略，构建分段线性函数来模拟图中每条边的权重。此外，为了评估算法的可扩展性，我们将每条边的插值点数量从 2 调整到 6（例如，参数 $c \in \{2，3，4，5，6\}$），这遵循了 [17] 中的参数设置。我们将 c 的默认值设为 3（例如，一条道路路段一天内的出行成本可能有 3 种不同取值）。这些数据集的详细信息见表 2。

对于每个数据集，我们遵循 [29] 中相同的设置来生成查询。具体而言，我们首先随机选择 1,000 对顶点，并在 10 个不同的时间间隔内均匀生成查询时间，因此每个数据集有 10,000 个查询。我们记录相应查询集内这 10,000 个查询的平均查询处理时间。

5.1 查询效率评估

出行成本查询评估。 我们评估了出行成本查询效率，并将所提出的算法与 TD-G-tree 和 TD-H2H 进行比较。结果如图 8 所示。

我们可以看到，基于树分解的方法的表现比 TD-G-tree 高出数个数量级，此外，它并非
将 TD-G-tree 扩展到现实世界中的大规模道路网络并不可行。对于数据集 $W-USA$，在设置为 $c \geq 3$（例如，每条道路的出行成本一天内变化两次以上）的情况下，构建索引需要超过 24 小时，因此我们不报告这些设置下的结果。这是因为分层划分构建方案存在数据冗余问题。当 $c = 2$ 时，$W-USA$ 的结果如表4所示。尽管



(a) 在 *CAL*上的成本查询 **(b) 在 *CAL*上的成本函数查询**

(c) 在 *SF*上的成本查询 **(d) 在 *SF*上的成本函数查询**

(e) 在 *COL*上的成本查询 **(f) 在 *COL*上的成本函数查询**

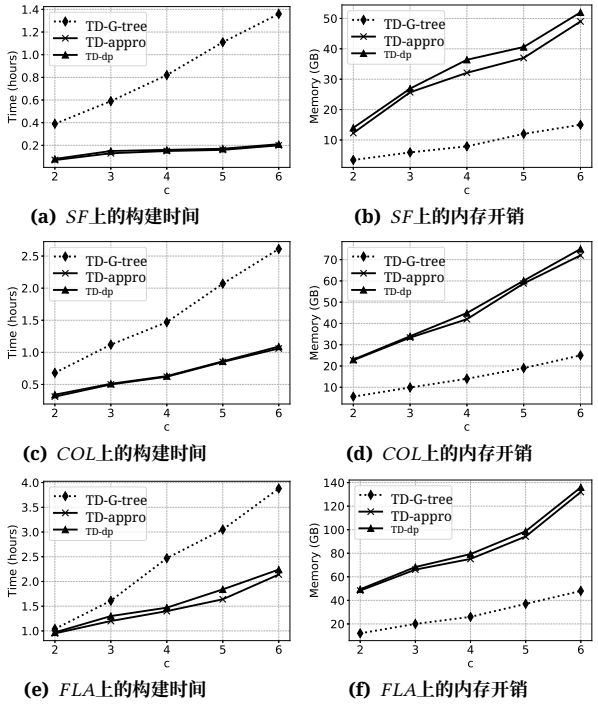
(g) 在 *FLA*上的成本查询 **(h) 在 *FLA* 上的成本函数查询**

图8：查询效率结果

TD-H2H 索引在表3的数据集 *CAL* 上表现最佳，但由于内存开销巨大，该索引无法扩展到拥有超过 100k 个顶点的其他道路网络（我们将在下一小节讨论这一点）。我们的方法 TD-dp 和 TD-appro 在数据集 *CAL* 上所需的处理时间与 TD-H2H 几乎相同，并且在顶点数从 100,000 到超过 5,000,000 的大规模道路网络上也表现良好。对于所有数据集，随着参数 c 变大（例如，出行成本变化频繁），TD-dp 和 TD-appro 的出行成本时间增长缓慢。与索引 TD-basic 相比，尽管它构建高效且占用的内存最少，但其查询时间表现最差。获取查询结果耗时超过 10 秒是不可接受的，这也表明有必要在 TD-basic 索引上构建快捷方式。

最短旅行成本函数的评估$f_{s,\ d}(t)$ 查询。我们测试了各个数据集所生成查询集上的最短旅行成本函数查询性能。结果如图 8 所示。

与出行成本查询相比，最短旅行成本函数查询耗时要长得多，这是因为算法遍历树索引时，最短旅行成本函数查询需要计算 *Compound*() 运算符函数。此外，随着参数 c（道路路段在一天内发生变化的次数）的增加，TD-G-tree 在每个数据集上的查询时间都会显著增加。



(a) *SF*上的构建时间 **(b)** *SF*上的内存开销

(c) *COL*上的构建时间 **(d)** *COL*上的内存开销

(e) *FLA*上的构建时间 **(f)** *FLA*上的内存开销

图9：构建成本的结果

结果还表明，基本树分解技术无法直接应用于时变道路网络；计算两个顶点之间的最短旅行成本函数需要 10 秒以上。基于基本树分解技术选择并构建快捷方式后，我们的算法 TD-dp 和 TD-appro 的性能显著提升。我们还发现，与 TD-G-tree 相比，TD-dp 和 TD-appro 在所有数据集上分别将查询时间平均缩短了 109 毫秒和 77 毫秒（$c = 2$）。

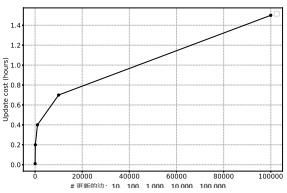


图10： *SF*上的索引更新

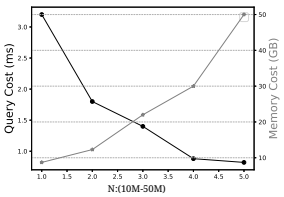


图11： *N*在 *FLA*上的评估

5.2 索引构建成本评估

构建时间评估。我们评估了每个索引的构建时间，并将我们提出的方法与 TD-G-tree 和 TD-H2H 进行了比较。图 9 展示了索引构建时间。可以看出，基于树分解的方法比 TD-G-tree 更快地构建索引。对于数据集 *CAL*，TD-basic 构建索引所需时间不超过 1 秒。然而，随着参数 c 变大，索引构建时间显著增加，例如，TD-G-tree 的构建时间从 3,761 秒增加到 13,972 秒（当 c 从 2 增加到 6 时）。此外，对于包含超过 5,000,000 个顶点的数据集 $W-USA$，当 c至少为 3 时，TD-G-tree 构建索引需要超过 24 小时，因此，在现实世界的大规模道路网络中采用 TD-G-tree 效率较低。对于 TD-dp 和 TD-appro，与基线方法相比，它们在不同 c 取值下的构建时间更加稳定。

内存开销评估。内存开销的实验结果如图 9 所示。我们可以观察到，随着参数 c 的值增大，所有方法的索引大小都会增加。对于数据集 *CAL*，TD-H2H 的索引大小（5.7G）至少是 TD-G-tree（169MB）的 34 倍。这是因为，TD-H2H 中的每个节点不仅维护与其祖先节点相关的辅助信息，还维护到所有祖先节点的最短旅行成本函数。当数据集变得更大时（例如，对于 *SF*，其顶点数超过 10,000），TD-H2H 会占用过多内存，无法支持查询。不过，TD-G-tree、TD-dp 和 TD-appro 的索引大小相近。这是因为，我们选择一组树节点对，在基本树分解索引 TD-basic 上构建快捷方式；TD-basic 在所有数据集上的索引大小都最小（例如，TD-G-tree 占用的空间是 TD-basic 的 5 倍）。

索引更新评估。我们改变边的更新比例，以评估 TD-appro 的更新效率，结果如图 10 所示。对于每个数据集，我们从边总数中随机选择 10、100、1,000、10,000 和 100,000 条边，以更新其权重函数。然后，对于每条选定的边，我们找到关联顶点 v，并根据**事实 1**中的自顶向下方式更新 $G(v)$ 中的快捷方式。最后，报告总时间成本。可以看到，随着更新比例增大，更新时间也随之增加，但总体更新开销仍可接受。

5.3 关于约束 N的评估

我们改变选择约束 N，以测试 TD-appro 的性能。N的默认值如表 2 所示。结果如图 11 所示，我们可以观察到，随着数值 N增大，构建时间和内存开销也会增加。当 N变大时，查询处理时间会缩短。这是因为，如果 N较大，将在索引 TD-dp 和 TD-appro 中选择并构建更多快捷方式，这将花费更多时间和空间。然而，更多快捷方式可以提高查询处理的效率。

5.4 实验总结

我们将结果总结如下：

- 现有索引难以高效地回答时变道路网络上的最短出行路径查询，尤其是在道路网络规模庞大且出行模式频繁变化时。例如，TD-G-tree 在 $W-USA$上构建索引需要 24 小时以上。TD-H2H 占用的空间过大，无法处理顶点数超过 300,000 的道路网络（例如，*SF*）。因此，将现有索引扩展到现实应用中并不可行。

- 对于最短旅行成本函数查询，所提出的索引 TD-dp 和 TD-appro 在 *COL* 上分别减少了 114 毫秒和 103 毫秒，在 *FLA* 上分别减少了 170 毫秒和 95 毫秒。在索引构建方面，由于这两种方法基于树分解进行构建，因此速度几乎是基线方法的 2 倍。
- TD-dp耗时0.9秒，比TD-appro构建索引所需时间多。结果证明了近似捷径选择算法的效率。TD-dp的查询处理时间略短于TD-appro，例如，不超过30毫秒。

6 相关工作

时间依赖最短路径。 时间依赖最短路径问题最早在 [7], 中提出，利用递推公式对该问题进行建模。基于Dijkstra算法的算法最早在 [11] 和 [8], 中设计，其时间复杂度为 $O((n \cdot logn + m) \cdot c)$。这些算法在大规模时变道路网络中的性能可能较差。在 [11], 中，该算法在包含1万个顶点的图上回答查询所需时间超过10秒。 A^* 中的算法也在 [15]中扩展到时变道路网络。然而，在最坏情况下，该方法可能需要指数级运行时间，才能扩展图中的所有可能最短路径。基于 A^* 算法，一些改进技术（如地标和双向搜索方案）也已应用于时变场景，见 [10] 和 [20]。然而，这些方法在真正的大规模道路网络中无法取得良好效果。

此外，还有一些基于层次结构和索引的研究，用于加速时变道路网络上的最短路径查询。 [9] 和 [2] 中提出的算法将 SHARC 和 CH 算法应用于时变道路网络；这两种算法是静态道路网络中的高效方法。然而，每个顶点可能需要占用千字节级的内存 [2], , 因此，将该方法应用于包含数千万个以上顶点的现实世界时变道路网络是不可行的。对于基于索引的方法 [29], , 它通过以分层方式对道路网络进行分区来构建基于树的索引，并维护分区中任意一对边界顶点之间的出行成本信息。当进行查询时，该索引会在树中自底向上汇总出行成本。分层索引构建方案会生成大量冗余的出行成本信息，从而阻碍了该方法在道路网络规模庞大且道路路段的出行成本频繁变化的情况下高效实施。

树分解。 树分解是一种高效技术，已用于许多应用中，以加速图结构上的某些计算问题 [3]。给定一个图，判断其树宽是否不超过给定值是NP完全问题 [1]。与以往研究相同，本文采用一种次优算法 [24] 。尽管该算法无法以有界近似比计算最小树宽，但即使在大型真实道路网络中，也能实现合理的常数树宽。考虑到我们的查询算法的时间复杂度关于树分解的树宽是多项式的，这种次优方法在时变道路网络中同样具有实际应用价值。

树分解方法已被广泛用于提高静态道路网络上的最短路径计算效率。在 [21], 中，树分解有助于克服基于层次结构的解决方案和基于跳数的解决方案各自的缺点。对于 [21], 中的树，每个树节点的所有祖先节点的层级都高于该节点，并且该节点维护到所有祖先节点的1跳最短距离。给定查询源点和目的地，算法分别检查源点和目的地到祖先节点的1跳距离，从而以 $O(w(T_G))$ 的时间复杂度完成查询。受 [21], 的启发，文献 [5] 尝试通过减小每个树节点的标签大小来提高最短距离查询的效率。除最短距离查询外，树分解方法还支持静态道路网络上的其他多种查询。在 [32],中，树分解构建了一个新的树索引，其中包含边的标签约束信息。基于树分解的标签约束最短路径查询显著优于基线方法。针对最短路径计数问题，文献 [23]提出了一种名为TL索引的新型索引。TL索引基于树分解方法构建，索引中的每个树节点都保存其到祖先节点的最短距离及对应的最短路径数量。

7 结论

本文研究了大规模时变道路网络上的最短路径查询问题。静态道路网络上的这类操作已得到广泛研究。然而，在许多现实应用中，路径的出行成本取决于路径的出发时间。因此，本文首先定义了时变道路网络上的最短路径查询问题。随后，我们采用树分解技术，在图上构建一棵树，以加速最短路径处理。此外，我们依据树的性质，选择并构建一些捷径，以提高在树结构上计算最短路径的效率。接下来，我们对树上的捷径选择问题进行建模，并证明其为 NP难问题。为利用有限的主存，我们提出了两种选择算法来解决该问题。第一种是基于动态规划的选择算法，另一种是基于贪心近似的算法，并且我们证明其近似比为 0.5。最后，我们进行了大量性能测试，验证了捷径选择算法以及基于选定快捷方式的查询处理的有效性和效率。

参考文献

[1] Stefan Arnborg、Derek G. Corneil 和 Andrzej Proskurowski。1987。K-树中嵌入的查找复杂度。*SIAM* 代数离散方法杂志8, 2（4月 1987 277 284 10 1137 0608024），–。https://doi.org/。/ Gernot [2] Veit Batz、Daniel Delling、Peter Sanders 和 Christian Vetter。2009。时变收缩层次结构。载于第十一届算法工程与实验研讨会论文集，*ALENEX 2009*，美国纽约州纽约市，*2009*年*1*月*3*日，Irene Finocchi 和 John Hershberger（编）。SIAM，97–105。https://doi.org/ 10 1137 1 9781611972894 10 。/。。J.A. Bondy 和 U.S.R Murty。。[3] 2008。《图论》（第1版）。Springer Publishing Company, Incorporated。[4] Di Chen、Ye Yuan、Wenjin Du、Yurong Cheng 和 Guoren Wang。2021。时变道路网络上的在线路径规划。载于*2021 IEEE* 第*37*届数据工程国际会议（*ICDE*）。325–335。https://doi.org/10.1109/ ICDE51399.2021.00035[5] Zitong Chen、Ada Wai-Chee Fu、Minhao Jiang、Eric Lo 和 Pengfei Zhang。2021。P2H：通过投影顶点在道路网络上高效进行距离查询

分隔点（SIGMOD '21）。计算机协会，纽约，NY，美国，313 325 10 1145 3448016 3459245 –。https://doi.org/。/。Kenneth [6] L Cooke 和 Eric Halsey。。具有时变节点间通行时间的网络中的最短路径。数学分析与应用杂志 14, 3 (1966), 493–498。[7] Kenneth L Cooke 和 Eric Halsey。1966。具有时变节点间通行时间的网络中的最短路径。数学分析与应用杂志 14, 3 (1966), 493 498 10 1016 0022 247 66 90009 6 –。https://doi.org/。/-X()- Frank [8] Dehne、Masoud T. Omran 和 Jörg-Rüdiger Sack。2012。时变 FIFO 网络中的最短路径。Algorithmica 62, 1-2 (2012), 416–435。https://doi.org/ 10 1007 00453 010 9461 6 。/s-– Daniel Delling。。时变 SHARC 路由。载于算法——ESA 2008，第16届欧洲研讨会，2008年9月15日至17日，德国卡尔斯鲁厄，论文集（计算机科学讲义），Dan Halperin 和 Kurt Mehlhorn（编），第5193卷。Springer，332–343。https://doi.org/10.1007/978-3-540-87744-8 28[10] Ugur De [9] miryurek、Farnoush Banaei Kashani、Cyrus Shahabi 和 Anand Ran-ganathan。2011。时变空间网络中最快路径的在线计算。载于《空间与时态数据库进展——第12届国际研讨会，SSTD 2011》，美国明尼苏达州明尼阿波利斯，2011年8月24日至26日，论文集（计算机科学讲义），Dieter Pfoser、Yufei Tao、Kyriakos Mouratidis、Mario A. Nascimento、Mohamed F. Mokbel、Shashi Shekhar 和 Yan Huang（编），第 6849 92 111 10 1007 978 3 642 22922 0_7 卷。Springer，–。https://doi.org/。/---- Bolin Ding、Jeffrey Xu Yu 和 Lu Qin。。大型图上的时变最短路径查找。载于第11届国际数据库技术拓展会议：数据库技术进展论文集（法国南特）（EDBT '08）。计算机协会，纽约，NY，美国， [11] 2008 205 216 10 1145 1353343 1353371 –。https://doi.org/。/。Bowen Du、Yongxin Tong、Zimu Zhou、Qian Tao 和 Wenjun Zhou。2018。电动汽车共享的需求感知型充电桩规划。载于 KDD。1330–1338。[13] Dawei Gao、Yongxin Tong、Jieying She、Tianshu Song、Lei Chen 和 Ke Xu。2016。空间众包中的 Top-k 团队推荐。载于 WAIM。191–204。[14] Shuai Huang、Yong Wang、Tianyu Zhao 和 Guoliang Li。2021。一种在道路网络上计算最短路径距离的学习方法。载于2021 IEEE 第37届数据工程国际会议（ICDE）。360–371。https://doi.org/ 10 1109 51399 2021 00038 。/ICDE。。Evangelos Kanoulas、Yang Du、Tian Xia 和 Donghui Zhang。2006。基于速度模式的道路网络最快路径查找。载于第22届国际数据工程会议论文集，ICDE 2006，2006年4月3日至8日，美国佐治亚州亚特兰大，Ling Liu、Andreas Reuter、Kyu-Young Whang 和 Jianjun Zhang（编）。IEEE 计算机学会， [15] 10 10 1109 2006 71 。https://doi.org/。/ICDE。。David E Kaufman 和 Robert L Smith。。用于智能车辆-公路系统的时变网络最快路径。智能交通系统杂志 1, 1 (1993), 1–11。[17] Jiajia Li、Cancan Ni、Dan He、Lei Li、Xiufeng Xia 和 Xiaofang Zhou。2022。时变道路网络中移动对象的高效 k近邻查询。VLDB 杂志 (2022)。https://doi.org/10.1007/s00778-022-00758-w [18] Lei Li、Mengxuan Zhang、Wen Hua 和 Xiaofang Zhou。2020。道路网络中批量最短路径处理的快速查询分解。载于2020 IEEE 第36届数据工程国际会议（ICDE）。1189–1200。https://doi.org/ 10 1109 48307 2020 00107 。/ICDE。。Zijian Li、Lei Chen 和 Yue Wang。。G*-Tree：道路网络上的高效空间索引。载于第35届 IEEE 数据工程国际会议，ICDE 2019，澳门，2019年4月8日至11日。IEEE，268–279。 [16] 1993 [19] 2019 https://doi.org/10.1109/ICDE. 2019.00032[20] Giacomo Nannicini、Daniel Delling、Dominik Schultes 和 Leo Liberti。2012。时变道路网络上的双向 A* 搜索。Networks 59, 2 (2012), 240 251 10 1002 20438 –。https://doi.org/。/net。Dian Ouyang、Lu Qin、Lijun Chang、Xuemin Lin、Ying Zhang 和 Qing Zhu。2018。当层次结构遇上2跳标记：道路网络上的高效最短距离查询。载于2018年数据管理国际会议论文集（美国得克萨斯州休斯敦）（SIGMOD '18）。计算机协会，纽约，NY，美国， [21] 709 724 10 1145 3183713 3196913 –。https://doi.org/。/。Miao Qiao、Hong Cheng、Lijun Chang 和 Jeffrey Xu Yu。。近似最短距离计算：一种查询相关的局部地标方案。载于2012 IEEE 第28届数据工程国际会议（ICDE）。462–473。https://doi.org/ 10 1109 2012 53 。/ICDE。。Yu-Xuan Qiu、Dong Wen、Lu Qin、Wentao Li、Ronghua Li 和 Ying Zhang。2022。大型道路网络中的高效最短路径计数。VLDB 基金会会刊 15, 10 (2022), 2098–2110。 [22] 2012 [23] https://www.vldb.org/pvldb/vol15/p2098-qiu.pdf [24] Neil Robertson 和 Paul D. Seymour。1984。图小项。III. 平面树宽。组合理论杂志B辑 36, 1 (1984), 49–64。https://doi.org/10.1016/0095-8956(84) 90013-3[25] Jieying She、Yongxin Tong、Lei Chen 和 Tianshu Song。2017。反馈感知的社交活动参与者安排。载于 SIGMOD。851–865。[26] Yongxin Tong、Ye Yuan、Yurong Cheng、Lei Chen 和 Guoren Wang。2017。时空众包数据管理技术综述。软件学报 28, 1 (2017), 35–58. [27] Yongxin Tong、Yuxiang Zeng、Zimu Zhou、Lei Chen 和 Ke Xu。2022。共享出行统一路线规划：一种基于插入的框架。*ACM* 数据库系统汇刊 47, 1 (2022), 2:1–2:48. [28] Yongxin Tong、Yuxiang Zeng、Zimu Zhou、Lei Chen、Jieping Ye 和 Ke Xu。2018。共享出行路线规划的统一方法。*VLDB* 基金会会刊 11, 11 (2018), 1633–1646. [29] Yong Wang、Guoliang Li 和 Nan Tang。2019。查询时变道路网络上的最短路径。*VLDB* 基金会会刊 12, 11 (2019年7月), 1249–1261. https://doi.org/10.14778/3342263.3342265[30] Yishu Wang、Ye Yuan、Hao Wang、Xiangmin Zhou、Congcong Mu 和 Guoren Wang。2021。在大型多模式时变网络上进行约束路线规划。载于*2021 IEEE* 第*37*届国际数据工程会议（*ICDE*）。IEEE 计算机学会，洛斯阿拉米托斯，美国加利福尼亚州，313–324。https://doi.org/ 10 1109 51399 2021 00034 . /ICDE . . Yuxiang Zeng、Yongxin Tong、Yuguang Song 和 Lei Chen。2020。越简单越好：一种用于共享路线规划查询的索引方法。*VLDB* 基金会会刊 13, 13 (2020), 3517–3530. [32] Junhua Zhang、Long Yuan、Wentao Li、Lu Qin 和 Ying Zhang。2021。道路网络上的高效标签约束最短路径查询：一种树分解方法。*VLDB* 基金会会刊 15, 3 (2021年11月), 686–698. https: //doi.org/ 10 14778 3494124 3494148 . / . Ruicheng Zhong、Guoliang Li、Kian-Lee Tan、Lizhu Zhou 和 Zhiguo Gong。2015。G-Tree：一种用于道路网络空间搜索的高效且可扩展索引。*IEEE* 知识与数据工程汇刊 27, 8 (2015), 2175–2189. https://doi.org/10.1109/TKDE.2015.2399306[34] Andy Diwen Zhu、Hui Ma、Xiaokui Xiao、Siqiang Luo、Youze Tang 和 Shuigeng Zhou。2013。道路网络上的最短路径和距离查询：迈向理论与实践的衔接。载于*2013*年*ACM SIGMOD*数据管理国际会议论文集（美国纽约州纽约市）（*SIGMOD '13*）。计算机协会，纽约，美国纽约州，857–868。 [31] [33] https://doi.org/10.1145/2463676.2465277