8. **foreach** 图 G_i 中从 v_i 出发的、出发时刻为 t 的边 e **do**
9. 生成一条仅包含 e 的路径ρ,加进 H 中;
10. **while** $H\neq\varnothing$ **do**
11. 设$\rho=\langle u,t_a^*,c^*,\rho_{pred}\rangle$是 H 的堆顶的路径,摘除ρ;
12. 设最近添加进$\mathcal{B}(u)$的路径是ρ'. **if** ρ'的到达时刻$\leqslant t_a^*$ **then continue**;
13. 将ρ添加进$\mathcal{B}(u)$;
14. **foreach** 图 G_i 中从 u 出发的每条边 $e=\langle u,w,t_d,t_a,c\rangle$ **do**
15. **if** $t_d<t_a^*$ **then continue**;
16. 生成路径$\rho_{new}=\langle w,t_a,c^*+c,\rho\rangle$;
17. **if** $c^*+c\geqslant\mathcal{C}(e)$ **then continue**; **else** 令$\mathcal{C}(e)=c^*+c$,将ρ_{new}加进 H;
18. **foreach** G_i 中的顶点 u **do**
19. 设 $L_{v_i}^u$ 是$\mathcal{L}_-(u)$中表示从 v_i 到 u 的路径的标签组.调用 *MarkPrune*($\mathcal{B}(u)$, $L_{v_i}^u$);
20. **foreach** $\rho=\langle u,t_a,c,\rho_{pred}\rangle\in\mathcal{B}(u)$,其中,$u$ 为 G_i 中的顶点 **do**
21. **if** ρ标记为保留但ρ_{pred}标记为删除 **then**
22. 标记ρ_{pred}为保留.追溯检查ρ_{pred}的前缀路径 ρ'_{pred} ,直到 ρ'_{pred} 标记为保留;
23. **foreach** $\rho=\langle u,t_a,c,\rho_{pred}\rangle\in\mathcal{B}(u)$,其中,$u$ 为 G_i 中的顶点 **do**
24. 设 $L_{v_i}^u$ 是$\mathcal{L}_-(u)$中表示从 v_i 到 u 路径的标签组.生成标签$\langle t,t_a,c,l_{ptr}\rangle$加进 $L_{v_i}^u$,其中,l_{ptr} 指向表示ρ_{pred}的标签;
25. **foreach** G_i 中的顶点 u **do** 对$\mathcal{L}_-(u)$中的标签组 $L_{v_i}^u$ 内的标签排序;
26. 仿照第 2 行~第 25 行生成到达 v_i 的基本路径,选择部分路径生成标签集合;
27. 将 v_i 从 G_i 中删掉,生成 G_{i+1};
28. 返回所有顶点 $v\in V$ 的$\mathcal{L}_+(v)$和$\mathcal{L}_-(v)$;

如何从满足等级约束的路径中有效地计算出非支配路径?假设计算从 v_i 出发的基本路径.由于在 t 时刻出发的路径仅可能被在 t'时刻($t'\geqslant t$)出发的路径支配,所以按 v_i 的出发时刻降序的顺序调用 Dijk-CCMTP(第 6 行). 第 7 行~第 17 行的处理与 Dijk-CCMTP 类似,但做了以下改动.

(1) 保存在堆 H 中的路径$\rho=\langle u,t_a,c,\rho_{pred}\rangle$增加一个分量$\rho_{pred}$,表示$\rho$的前缀路径.

(2) 对于 G_i 中的每个顶点 u,用一个包$\mathcal{B}(u)$收集从 v_i 到 u 的非支配路径.ρ从 H 摘除后(第 11 行),仅当ρ不被$\mathcal{B}(u)$中的路径支配时才将ρ加进$\mathcal{B}(u)$.回顾 H 中的路径,按照费用升序出堆;另一方面,第 10 行~第 17 行的 while 循环计算的路径有相同的起点(s)和出发时刻(t),可推出路径按到达时刻降序的顺序加入$\mathcal{B}(u)$,所以只需要比较ρ和最近一条加入$\mathcal{B}(u)$的路径的到达时刻,即可得出ρ是否被$\mathcal{B}(u)$内的路径支配(第 12 行).

(3) 由于终点 d 和费用上限θ在构建索引时未知,所以在 BuildIndex 中不利用 d 和θ进行剪枝.

当在 t 时刻从 v_i 出发的路径遍历完后,对于每个可达顶点 u,它的包$\mathcal{B}(u)$包含了到达 u 的非支配路径.第 19 行的 MarkPrune 算法从$\mathcal{B}(u)$中选择部分路径加入 ACCTL,选择细节将在第 5.2 节中讨论.MarkPrune 独立地选择$\mathcal{B}(u)$中的路径加进 ACCTL,即是说,MarkPrune 在选择$\mathcal{B}(u)$中的路径时并不考虑另一个顶点 v 的包$\mathcal{B}(v)$内的路径,其后果可能导致还原路径时出错.假设在选择$\mathcal{B}(u)$中的路径时,路径$\rho=\langle\cdot,\cdot,\cdot,\rho_{pred}\rangle$被选中,生成相应的标签 l 加进 ACCTL.为了保证标签 l 能正确被还原,表示路径ρ_{pred}的标签 l_{ptr} 必须在 ACCTL 中.然而 l_{ptr} 可能不在 ACCTL 中,因为 MarkPrune 在选择$\mathcal{B}(v_{pred})$内的路径(假设 v_{pred} 是ρ上 u 的直接前驱顶点)时并没有考虑ρ.因此,在调用完 MarkPrune 以后,需要自底向上地检查每条被保留的路径是否可被还原.加进 ACCTL 的标签是否可被还原,取决于它对应的路径ρ的前缀路径ρ_{pred}是否也生成对应的标签加进 ACCTL.对于每条被 MarkPrune 算法标记为保留的路径ρ,如果ρ_{pred}也被标记成保留,则对ρ的检查结束;否则,将ρ_{pred}标记成保留,并追溯ρ_{pred}的前缀路径是否标记

为保留,直到遇到前缀路径被标记为保留为止.假设 G_i 中所有顶点 u 的$\mathcal{B}(u)$集合的路径总数为 m,第 20 行~第 22 行自底向上检查的时间复杂度是 $O(m)$.由引理 2,m 不超过$|E|$.实际上,受出发时刻和路径支配约束,$m<<|E|$.因此,第 20 行~第 22 行不会产生太大耗费.

对于每条标记为保留的路径,第 23 行、第 24 行生成标签 l 并加进相应的标签组.最后,第 26 行调用反向 Dijk-CCMTP 算法计算到达 v_i 的基本路径,计算过程与第 4 行~第 25 行相似.

5.2 选择路径

本节关注算法 3 第 19 行的 MarkPrune 算法如何选择$\mathcal{B}(u)$中的路径加进 ACCTL.$\mathcal{B}(u)$包含了在 t 时刻出发的从 v_i 到 u 的路径.关键问题是:如何在$\mathcal{B}(u)$中选择尽量少的路径加进 ACCTL,同时能够保证对于任意的查询 Q,都能在 ACCTL 中找到标签生成近似解?换句话说,对于任意路径$\rho\in\mathcal{B}(u)$,如果表示ρ的标签没有加进 ACCTL,则在 ACCTL 中存在另一条标签对应路径ρ',使得$\rho'\alpha$-支配ρ.精确计算出保留的路径的最小集合是一个 NP-难问题,MarkPrune 采用贪心的策略选择路径,如算法 4 所示.

算法 4. MarkPrune.

输入:在 t 时刻出发从 v_i 到 u 的基本路径的集合$\mathcal{B}(u)$,$\mathcal{L}_{-}(u)$中表示从 v_i 到 u 的路径的标签组 $L^u_{v_i}$.

输出: $\mathcal{B}(u)$内的标签标记为保留或删除.

1. 假设$\mathcal{B}(u)$内的路径的加入顺序依次是$\rho_0,\rho_1,\ldots,\rho_{k-1}$,初始时对于 $0\leqslant i<k$,设 $d(\rho_i)=\rho_i$;
2. 标记ρ_{k-1}为保留;令 $x=k-1$;
3. **for** $y=k-2,k-3,\ldots,1,0$ **do**
4. **if** $\rho_x\alpha$-支配ρ_y **then** 标记ρ_y为删除;记录 $d(\rho_x)=\rho_y$;
5. **else** 标记ρ_y为保留;令 $x=y$;
6. **foreach** 标记为保留的路径$\rho\in\mathcal{B}(u)$ **do**
7. **if** $L^u_{v_i}$ 内存在标签表示路径ρ',使得$\rho'\alpha$-支配 $d(\rho)$ **then** 标记ρ为删除;
8. 返回;

假设加入$\mathcal{B}(u)$的路径的顺序依次是$\rho_0,\rho_1,\ldots,\rho_{k-1}$.由于路径按费用升序生成,受路径支配的限制,$\rho_0,\rho_1,\ldots,\rho_{k-1}$自动按到达时刻降序排列.因此,$\rho_i$ 仅可能α-支配比它早加入$\mathcal{B}(u)$的路径$\rho_j(j<i)$.MarkPrune 中,令 x 指向最近被标记为保留的路径,初始时令ρ_{k-1}标记为保留,x 指向ρ_{k-1}.接下来,从ρ_{k-2}开始依次往前扫描路径,令 y 指向当前被检查的路径.如果$\rho_x\alpha$-支配ρ_y,则ρ_y 标记为删除;否则,ρ_y 标记为保留.然后,继续检查ρ_y 可否α-支配更早期加入$\mathcal{B}(u)$的路径,令 x 指向ρ_y.

回顾在 BuildIndex 算法中,从 v_i 出发的路径按出发时刻从晚到早的顺序生成,因此当前 $L^u_{v_i}$ 内的标签表示的路径的出发时刻均晚于$\mathcal{B}(u)$内的路径,也可能α-支配$\mathcal{B}(u)$内的某些路径.直观来说,如果在 $L^u_{v_i}$ 内存在某条标签对应的路径$\rho'\alpha$-支配$\mathcal{B}(u)$内的某条路径ρ,则可以把ρ删除.但这种判断会引起错误.假设α=1.1,$\mathcal{B}(u)$中存在两条路径ρ_{10} 和ρ_{11},费用分别是 10 和 11,且有$\rho_{11}\alpha$-支配ρ_{10}.于是保留ρ_{11},删除ρ_{10}.假设 $L^u_{v_i}$ 中存在某条标签表示路径ρ_{12},费用 12,且有$\rho_{12}\alpha$-支配ρ_{11}.如果因为ρ_{12}的存在删除ρ_{11},则没有路径α-支配ρ_{10}.因此,MarkPrune 采用 $d(\rho)$记录ρ的支配范围,即ρ能够α-支配的费用最小的路径是哪条. $L^u_{v_i}$ 中,某条标签表示的路径ρ'能够删除$\mathcal{B}(u)$中的某条路径ρ,仅当$\rho'\alpha$-支配 $d(\rho)$.

定理 2. BuildIndex 算法生成的 ACCTL 索引是正确的.

证明:假设 P_{opt} 是一个从顶点 s 到顶点 d 的查询 Q 的精确解,且 P_{opt} 上任意一段子路都是非支配路径.由引理 1,这样的路径 P_{opt} 是存在的.设 v_T是 P_{opt} 上等级最高的点,P_{opt} 上从 s 到 v_T的路段记为 P^+_{opt}, 从 v_T到 d 的路段记为 P^-_{opt}. 首先证明:在 ACCTL 中能找到标签生成路径 P,使得 $P\alpha$-支配 P_{opt}.BuildIndex 的第 6 行~第 17 行枚举了所有从 v_T出发的基本路径,保证生成了 P^-_{opt}; 第 18 行、第 19 行保证 ACCTL 存在路径 P_{-},$P_{-}\alpha$-支配 P^-_{opt}. 对称地,

ACCTL 也存在路径 P_+,$P_+$$\alpha$-支配 P_{opt}^+ 在费用方面,$c(P_+)+c(P_-)\leqslant\alpha\cdot c(P_{opt}^+)+\alpha\cdot c(P_{opt}^-)=\alpha\cdot c(P_{opt})$.在时间方面,$P_+$的到达时刻$\leqslant P_{opt}^+$ 的到达时刻$\leqslant P_{opt}^-$ 的出发时刻$\leqslant P_-$的出发时刻,所以 P_+与 P_-能够拼接成路径;同时,P_+的出发时刻$\geqslant P_{opt}^+$ 的,P_-的到达时刻$\leqslant P_{opt}^-$ 的,所以 P_+和 P_-拼接的路径α-支配 P_{opt}.最后,第 20 行~第 22 行保证了每条路径的前缀路径的标签都在 ACCTL 中,于是能够将标签正确还原成 G 上的一条路径. □

定理 3. 设查询 Q 的费用限制是θ,P_{opt} 是 Q 的精确解,且 P_{opt} 上任意一段子路都是非支配路径.设 P 是用 ACCTL 查询所得解.

(1) 若$\alpha\cdot c(P_{opt})\leqslant\theta$,则 P 是精确解;

(2) 若$\theta<\alpha\cdot c(P_{opt})$,则 P 有可能是精确解,也可能是近似解;若 P 是近似解,满足:

(2.1) $c(P)\leqslant\alpha\cdot c(P_{opt})$;

(2.2) P 不早于 P_{opt} 出发且不晚于 P_{opt} 到达.

证明:设 P_{opt}^+ 和 P_{opt}^- 的含义如定理 2 中的证明所述.

- 用反证法证明:(1) 当$\alpha\cdot c(P_{opt})\leqslant\theta$时,$P$ 是精确解.因为 P_{opt} 上任意一段子路都是非支配路径,所以 P_{opt}^+ 是非支配路径,于是在 ACCTL 中存在标签 l_+^*,使得 l_+^* 表示的路径α-支配 P_{opt}^+;同理,ACCTL 存在标签 l_-^*,使得 l_-^* 表示的路径α-支配 P_{opt}^-. 于是,l_+^* 和 l_-^* 拼接的路径 P 满足费用限制:$c(P)\leqslant\alpha\cdot c(P_{opt}^+)+\alpha\cdot c(P_{opt}^-)=\alpha\cdot c(P_{opt})\leqslant\theta$.另一方面,由$\alpha$-支配的定义,$P$ 的出发时刻$\geqslant P_{opt}^+$ 的,到达时刻$\leqslant P_{opt}^-$ 的,与 P_{opt} 是最优解矛盾.
- (2) 当$\theta<\alpha\cdot c(P_{opt})$时,如果 ACCTL 中存在的标签 l_+^* 和 l_-^* 拼接的路径 P 满足 $c(P)\leqslant\theta$,则同情形(1)的证明,P 是精确解;若 $c(P)>\theta$,则 P 是近似解,可采用定理 2 的证明过程证明情形(2.1)和情形(2.2). □

定理 4. BuildIndex 算法的时间复杂度是 $O(|V|\cdot\Delta_{\max}\cdot|E|\cdot(\log|E|+\Delta_{\max}))$,其中,$|V|$表示顶点数,$|E|$表示边数,$\Delta_{\max}$表示顶点的最大度数.

5.3 点序o的计算

与文献[14,15,21,22]相同,尽管顶点的等级序 o 的选取不影响查询结果的正确性,但会影响索引所包含的标签数目,进而影响查询效率.要精确计算出 o,使得索引所包含的的标签数最少是一个 NP-难的问题.因此,本文采用文献[14,21]的方法,对 o 进行启发式计算.如果顶点 v 在路径 P 上,则称 v 覆盖 P.ACCTL 的作用好比把一张记录任意查询 Q 的解的数据库表 T 分解成两张表 T_1 和 T_2,使得 T 中的记录可以通过 T_1 和 T_2 的连接操作生成.T_1 好比记录了所有顶点 v 的$\mathcal{L}_+(v)$内的标签,T_2 记录了$\mathcal{L}_-(v)$内的标签.为使 T_1 和 T_2 尽量小,应该使得 T_1 和 T_2 之间能连接上的记录尽量多.这等同于:v 覆盖的带费用限制的最小时态路径越多,v 的等级就应该越高.由于带费用限制的最小时态路径太多,不可能逐一枚举,所以本文采用这样的方法确定 o:从图中随机选取一个顶点集合 $V_{sub}\subset V$,分别从 V_{sub} 的每个顶点 v 出发扩展路径.留意到路径具有时间和费用两个属性,两个点之间有多条不会被相互支配的路径.本文采用近似的方法将路径的时间和属性量化成一个值:一条路径 P 的“长度”等于 P 的费用$\times\beta+P$ 的耗时$\times(1-\beta)$,其中,β是一个[0,1)范围内的实数,在从 v 出发扩展路径之前随机生成.这样,从顶点 v 出发,采用类 Dijkstra 算法搜索全图,可得出 v 到其他顶点的最短路径,生成一棵以 v 为根的 Dijkstra 树.树上任一顶点 u 到 u 的子孙的路径也是最短路径,因此 u 在图中覆盖的最短路数目等于以 u 为根的子树包含的顶点数,包括 u 自身.u 在$|V_{sub}|$棵树覆盖的最短路径的总数视为 u 覆盖最短路径数.算法对$|V_{sub}|$棵树分别做自底向上的扫描,统计出每个顶点覆盖的最短路径数目,从中选出覆盖最短路径数最多的点 v_0 作为等级最高的顶点;然后,在每棵树中删除以 v_0 为根的子树,再更新各个顶点覆盖的最短路径数,在“剩下的”路径中,选择覆盖数目最大的顶点 v_1 作为等级次高的顶点...依此次类推.若$|V_{sub}|$棵树删除完后仍有顶点的 o 序未确定,则按顶点的度数从大到小排序确定.

6 实 验

实验采用 C++语言编写测试代码,测试平台 Windows 10,64 位操作系统,机器配置 Intel(R) Core(TM) i7-8700K CPU,64GB 内存.实验采用文献[21]提供的数据集做测试.数据采集自 GTFS[25],记录了一些地区的公共交通数据.数据集中每条边 e 只记录了出发时刻和到达时刻,实验给 e 生成一个费用值.为了模拟真实环境,一条从 u 到 v 的边的费用取值图中所有从 u 到 v 的边的耗时的平均值.实验数据的特征见表 4,$|V|$表示顶点数,$|E|$表示边数,$\Delta_{\max}$ 表示顶点最大度数,$\bar{\Delta}$ 表示平均顶点度数.下文就 ACCTL 索引的查询时间、索引大小和建立索引的时间进行分析.

Table 4 Characteristics of datasets

表 4 数据集特征

数据集	$\|V\|$	$\|E\|$	$\Delta_{\max}$	$\bar{\Delta}$	数据集	$\|V\|$	$\|E\|$	$\Delta_{\max}$	$\bar{\Delta}$
Austin	2.7K	317.5K	0.79K	118.83	Madrid	4.6K	1 912.2K	2.86K	412.52
SaltLakeCity	6.3K	329.0K	0.57K	52.74	Berlin	12.8K	1 965.8K	6.48K	162.02
Denver	9.5K	708.7K	1K	74.71	Rome	8.8K	2 267.7K	2.93K	259.91
Houston	9.8K	1 111.8K	1.57K	113.01	Toronto	10.8K	3 296.1K	2.14K	305.85
Budapest	5.5K	1 375.2K	2.24K	264.42	Sweden	51.4K	3 926.5K	7.97K	79.26
LosAngeles	15.0K	1 903.2K	1.4K	128.21	–	–	–	–	–

6.1 查询时间

查询沿用文献[21]提供的查询集.每个数据集的查询集内有 10 万个查询.每个查询包括起点 s、终点 d、时刻 t 和 t'、费用上限θ.s 和 d 从顶点集中随机产生,t 和 t'从数据集的时间域随机产生.费用上限θ是在$[\theta_{\min},\theta_{\max}]$范围内的一个随机数:$\theta_{\min}$ 表示不考虑路径上相邻的边之间的时间顺序,从 s 到 d 的最小费用;$\theta_{\max}$ 表示从 s 到 d 的某条随机的边出发的最短耗时路径的费用.带费用限制的最早到达路径查询使用 s,d,t 和θ作为查询参数;带费用限制的最晚出发路径查询使用 s,d,t'和θ;带费用限制的最短耗时路径查询使用 s,d,t,t'和θ.

实验将 ACCTL 与文献[1]提出的一种 Dijkstra 变种算法作对比.因为文献[1]解决的问题与本文的有差别(回顾文献[1]求解不超过费用上限的、路径上相邻的边满足时间顺序约束的、长度最短的路径),因此将文献[1]的 Dijkstra 变种算法修改成解决本文问题的算法,也即第 2 节讨论的 Dijk-CCMTP 的变种算法进行 3 类查询.下文中,对用基于 Dijkstra 方法求解带费用限制的最早到达路径、最晚出发路径和最短耗时路径的方法标记为 Dijk-CCEAP、Dijk-CCLDP 和 Dijk-CCSDP;对采用 ACCTL 索引查询的方法标记为 ACCTL-CCEAP、ACCTL-CCLDP 和 ACCTL-CCSDP.

图 2 显示了基于 Dijkstra 变种算法和基于 ACCTL 索引方法的查询时间对比.实验一共测试了 11 组数据,y 轴以 ms 为单位,用对数的比例显示每组数据的平均查询时间.ACCTL 索引的α取 1 求精确解,即调用 BuildIndex 时不执行第 18 行~第 22 行去掉部分α-支配路径的代码.

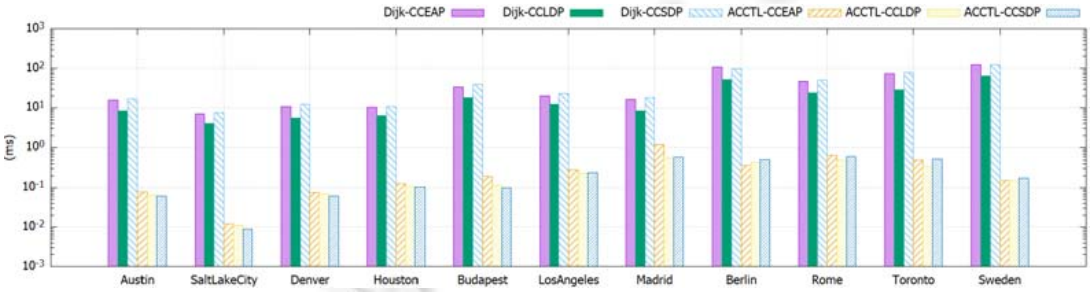
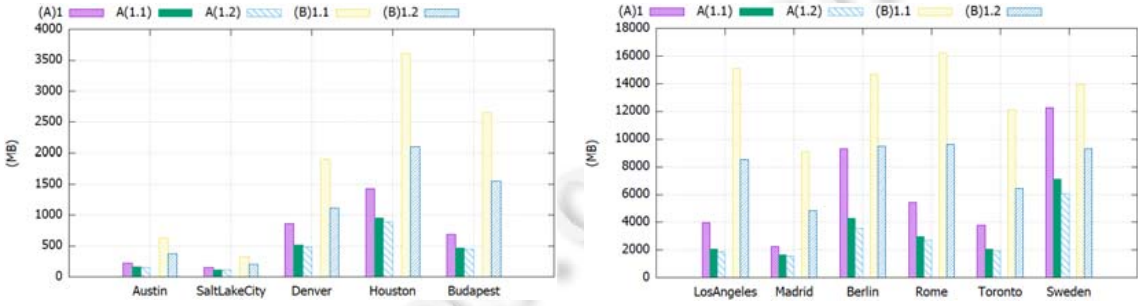


Fig.2 Query times of the Dijkstra variant method and ACCTL

图 2 基于 Dijkstra 搜索的变种算法和 ACCTL 索引的查询时间

总体看来,基于 ACCTL 索引的平均查询时间比基于 Dijkstra 的搜索快 2 个~3 个数量级.这是因为 ACTTL

在预先计算的路径集中查找路径生成解,避免在原图上盲目搜索.就各组数据的基于 ACCTL 索引的查询时间对比,查询时间与索引大小和图的度数有关.总体来说,索引越大,查询时间就越长.图 3 的各组数据的第 1 根柱子显示了α=1 时的索引大小.留意到 Madrid 的索引比 LosAngeles 和 Berlin 的都小,但查询时间长.这是因为 Madrid 的平均顶点度数大.顶点 v 的度数越大,v 到达图中其他顶点(或从图中其他顶点到达 v)的可能性越大,即$\mathcal{L}_+(s)$和$\mathcal{L}_-(d)$有拥有相同端点的标签组越多,所以需要匹配的标签组越多,查询时间越长.此外,最短耗时路径查询的速度比最早到达路径查询的稍慢,甚至在个别数据中最短耗时路径的查询比最早到达路径的更快(例如 Berlin 的 Dijk-CCEAP 与 Dijk-CCSDP、SaltLakeCity 的 ACCTL-CCEAP 与 ACCTL-CCSDP),这是因为最短耗时路径查询受到到达时刻 t'的约束,可以帮助剪枝掉部分搜索;而最早到达路径的到达时刻没有 t'的约束,所以搜索量反而更大.



(a) 前 5 组数据 (b) 后 6 组数据

Fig.3 Index size (MB) of group A and B, with α=1, 1.1, 1.2 respectively

图 3 A 组α取 1、1.1、1.2 和 B 组α取 1.1、1.2 时,各个图的索引大小(MB)

6.2 索引的大小

本节讨论α取值以及路径的费用值浮动对索引大小的影响.实验设置两大组数据集:A 组数据集的设置如上文所述,一条从 u 到 v 的边的费用取值图中所有从 u 到 v 的边的耗时的平均值;B 组数据集的图中一条边 e 的费用取值为[0.7×π,1.3×π]范围内的一个随机整数,其中,π表示通过 e 的耗时,即令边的费用取值多样化.换句话说,A 组的图的边的费用取值单一,B 组的图的边的费用取值多样.

图 3 的每组数据的前 3 根柱子显示了 A 组数据集下α取 1,1.1 和 1.2 时各个图的索引大小,以 MB 为单位,分别标记为(A)1、(A)1.1、(A)1.2.当α递增时,索引变小.因为 ACCTL 利用α-支配去掉部分基本路径,α放得越宽,可以被去掉的基本路径就越多.另一方面,α从 1.1 递增到 1.2 时,索引变小的幅度明显比α从 1 递增到 1.1 时的小.这是因为 ACCTL 需要保证原图中(α=1 时)的每条基本路径 P,在索引中都存在路径α-支配 P,而不是在α=1.1 时保留的路径的基础上去掉α-支配的路径.

图 3 的每组数据的后两根柱子表示 B 组数据集取α=1.1 和α=1.2 的索引大小,分别标记为(B)1.1 和(B)1.2.留意到 B 组的索引明显比 A 组要大.因为 B 组的图中边的费用取值多样,导致两点间路径的费用取值也多样,因此基本路径数也比 A 组要多.在 B 组数据集中,从α=1.1 递增到α=1.2 时,索引大小有 40%~50%的降幅.这是因为路径费用值取值多样,使得α变大后,有更多的路径因为α-支配被去掉.

综合上述比较,索引大小与α取值和图的费用值浮动的大小有关:α越大,索引越小,但随着α的增大,索引大小降幅变慢.另一方面,当图的路径的费用值浮动越大时,α的影响也越大.

6.3 建立索引的时间

图 4 显示了两大组数据建立索引的时间(单位:s).总体来说,A 组的建立索引时间小于 B 组,因为 A 组的基本路径数小于 B 组.两组数据的α=1.1 和α=1.2 的建立索引时间差别不大,可见,建立索引的瓶颈在于生成基本路径,不在于计算α-支配路径.这与第 5.1 节对 BuildIndex 算法的第 18 行~第 22 行的讨论一致.在 A 组数据内,α=1

时建立索引的时间略小于α=1.1 和 1.2,因为α=1 时不需要执行 BuildeIndex 的第 18 行~第 22 行.留意到图 Sweden 在α=1 时的执行时间反而大于α=1.1 和 1.2,这是因为α=1 时基本路径数多,对标签进行排序的时间不可忽略.

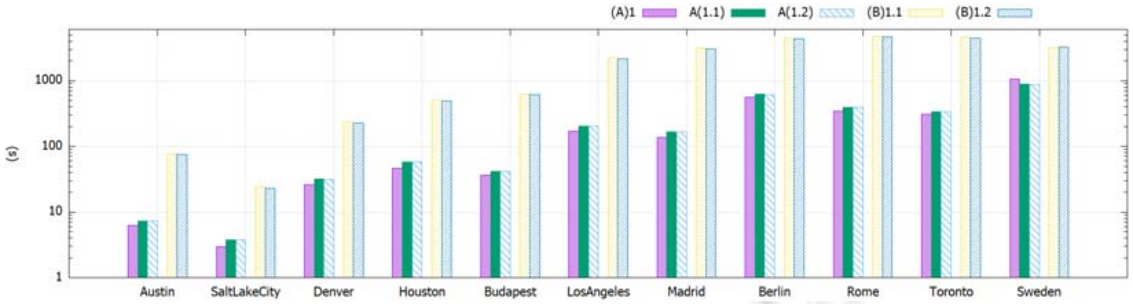


Fig.4 Index construction time of group A and B, with with α=1, 1.1, 1.2 respectively

图 4 A 组α取 1、1.1、1.2 和 B 组α取 1.1、1.2 时,各个图建立索引的时间

7 结束语

本文研究了公交网络下带费用限制的最小时态路径查询问题,提出了一种高效索引结构 ACCTL.ACCTL 通过预计算图中部分路径的信息,使得任意查询可以通过在 ACCTL 中检索路径生成近似解,避免在原图上盲目搜索,从而提高查询效率.本文对 ACCTL 的存储结构、建立索引方法和查询算法做了多方面的讨论和优化.实验验证了 ACCTL 索引支持的查询速度比在原图上采用 Dijkstra 变种算法的查询速度快 2 个~3 个数量级,并分析了 ACCTL 的存储空间和建立索引的时间的影响因素.

References:

[1] Sudip B, Arnab G, Rahul S. Restricted shortest path in temporal graphs. In: Proc. of the DEXA 2015, Part I. LNCS 9261, 2015. 13–27.

[2] Yang YJ, Gao H, Li JZ. Finding the optimal path under time-dependent cost function on graphs. Chinese Journal of Computers, 2012,35(11):2247−2264 (in Chinese with English abstract).

[3] Yang Y, Gao H, Yu JX, Li J. Finding the cost-optimal path with time constraint over time-dependent graphs. Proc. of the VLDB Endowment, 2014,7(9):673–684.

[4] Wang L, Yang LX, Gao ZY. The constrained shortest path problem with stochastic correlated link travel times. European Journal of Operational Research, 2016,255:43−57.

[5] He SX, Fan BQ, Yan L. Improved optimal path searching algorithm in transit network. Journal of University of Shanghai for Science and Technology, 2006,28(1):63−67 (in Chinese with English abstract).

[6] Wei JL, Fan XH, Liu LL, Liu Y, Ren JM, Sun QL. Solution of the optimization model of bus network time-limited free transfer based on DFS-backtracking algorithm. Science Technology and Egineering, 2017,17(10):304−307 (in Chinese with English abstract).

[7] Garey MR, Johnson DS. Computers and Intractability: A Guide to the Theory of NP-completeness. New York: W.H. Freeman & Co., 1979.

[8] Storandt S. Route planning for bicycles-exact constrained shortest paths mad pratical via contraction hierarchy. In: Proc. of the ICAPS. AAAI Press, 2012. 234−242.

[9] Geisberger R, Sanders P, Schultes D, Delling D. Contraction hierarchies: Faster and simple hierarchical routing in road networks. In: Proc. of the WEA. LNCS 5038, Springer-Verlag, 2008. 319−333.

[10] Lozano L, Medaglia AL. On an exact method for the constrained shortest path problem. Computers and Operations Research, 2013, 40(1):378−384.

[11] Ma TY, An A^* label-setting algorithm for multimodal resource constrained shortest path problem. Procedia-Social and Behavioral Science, 2014,111:330−339.

[12] Tsaggouris G, Zaroliagis CD. Multiobjective optimization: Improved FPTAS for shortest paths and non-linear objectives with applications. Theory of Computer Systems, 2009,45(1):162−186.

[13] Sedeno-NodaA, Alonso-Rodriguez S. An enhanced K-SP algorithm with pruning strategies to solve the constrained shortest path problem. Applied Mathematics and Computation, 2015,265:602−618.

[14] Sibo W, Xiaokui X, Yin Y, Wenqing L. Effective indexing for approximate constrained shortest path queries on large road networks. Proc. of the VLDB Endowment (PVLDB), 2016,10(2):61−72.

[15] Abraham I, Delling D, Goldberg AV, Werneck RF. Hierarchical hub labelings for shortest paths. In: Proc. of the European Conf. on Algorithms. Springer-Verlag, 2012. 24−35.

[16] Cooke KL, Halsey E. The shortest route through a network with time-dependent intermodal transit times. Journal of Mathematical Analysis and Applications, 1966,14(3):493−498.

[17] Geisberger R. Contraction of timetable networks with realistic transfers. In: Proc. of the Int'l Symp. on Experimental Algorithms (SEA 2010). LNCS 6049, Springer-Verlag, 2010. 71−82.

[18] Dibbelt J, Pajor T, Strasser B, Wanger D. Intriguingly simple and fast transit routing. In: Proc. of the Int'l Symp. on Experimental Algorithms (SEA 2013). LNCS 7933, Springer-Verlag, 2013. 43−54.

[19] Wu H, Cheng J, Huang S, Ke Y, Lu Y, Xu Y. Path problems in temporal graphs. Proc. of the VLDB Endowment (PVLDB), 2014,7(9):721−732.

[20] Wu H, Cheng J, Ke Y, Huang S, Huang Y, Wu H. Efficient algorithms for temporal path computation. IEEE Trans. on Knowledge and Data Engineering, 2016,28(11):2927−2924.

[21] Wang SB, Lin WQ, Yang Y, Xiao XK, Zhou SG. Efficient route planning on public transportation networks: A labelling approach. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD). New York: ACM, 2015. 967−982.

[22] Delling D, Dibbelt J, Pajor T, Wernect RF. Public transit labelling. In: Proc. of the Experimental Algorithms. Springer Int'l Publishing, 2015. 273−285.

[23] Bast H, Delling D, Goldberg AV, Hannemnn MM, Pajor T, Sanders P, Wanger D. Route planning in transportation networks. In: Kliemann L, Sanders P, eds. Proc. of the Algorithm Engineering. LNCS 9220, Cham: Springer-Verlag, 2016.

[24] Foschini L, Hershberger J, Suri S. On the complexity of time-dependent shortest paths. Algorithmica, 2014,68(4):1075−1097.

[25] GTFS. 2017. https://code.google.com/p/googletransitdatafeed/wiki/PublicFeeds

附中文参考文献:

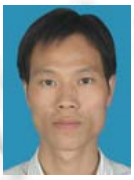
[2] 杨雅君,高宏,李建中.时间依赖函数下的最优路径查询问题研究.计算机学报,2012,35(11):2247−2264.

[5] 何胜学,范炳全,严凌.公交网络最优路径的一种改进求解算法.上海理工大学学报,2006,28(1):63−67.

[6] 魏金丽,范鑫贺,刘莲莲,刘阳,任杰睦,孙启龙.基于深度优先遍历算法-回溯算法的公交网络限时免费换乘优化模型求解.科学技术与工程,2017,17(10):304−307.

马慧(1981－),女,广东中山人,博士,副教授,CCF 专业会员,主要研究领域为数据库管理,图数据库与查询分析,算法设计.

汤庸(1961－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为学术社交网络,教育大数据服务.



梁瑞仕(1982－),男,博士,副教授,CCF 专业会员,主要研究领域为智能规划及其应用,大数据.