doi:10.11959/j.issn.1000-436x.2015230

BiRch：一种处理 *k* 步可达性查询的双向搜索算法

周军锋[1,2]，陈伟[1]，费春苹[1]，陈子阳[1,2]

（1. 燕山大学 信息科学与工程学院，河北 秦皇岛 066004；2. 河北省计算机虚拟技术与系统集成重点实验室，河北 秦皇岛 066004）

摘 要：针对现有方法低效或索引规模庞大的问题，提出一种双向搜索算法 BiRch。当判断顶点 *u* 是否满足 *k* 步可达顶点 *v* 时，首先比较 *u* 的出度和 *v* 的入度，优先处理度小的顶点。其优点体现在使用较小的索引，同时避免由于 *u* 的出度过大所带来的效率下降问题；提出基于双向广度层数和双向拓扑层数的剪枝策略来辅助过滤，减少需要访问的顶点数量。基于 19 个真实数据集进行测试，实验结果从索引构建时间、索引大小、查询响应时间、处理顶点数量以及扩展性方面验证了所提方法相对于现有方法的高效性。

关键词：*k* 步可达性查询；双向搜索；广度层数；拓扑层数

中图分类号：TP391 **文献标识码：**A

BiRch: a bidirectional search algorithm for *k*-step reachability queries

ZHOU Jun-feng[1,2], CHEN Wei[1], FEI Chun-ping[1], CHEN Zi-yang[1,2]

(1. School of Information Science and Engineering, Yanshan University, Qinhuangdao 066004, China;
2. Key Laboratory for Computer Virtual Technology and System Integration of Hebei Province, Qinhuangdao 066004, China)

Abstract: A new bidirectional processing algorithm, namely BiRch was proposed. When checking whether a vertex *u* can reach *v* within *k* steps, BiRch firstly compared the out-degree of *u* and the in-degree of *v*, and processed the one with smaller degree, such that to avoid large indexes and the inefficiency due to large degree. Two pruning strategies were proposed based on bidirectional breadth-first levels and bidirectional topological levels, such that to reduce the number of visited vertexes. Experimental results on 19 real datasets verify the efficiency of the proposed method in terms of different metrics, including indexing time, index size, query processing time, the number of visited vertexes, and scalability.

Key words: *k*-step reachability query; bidirectional search; breadth level; topological level

1 引言

给定有向无环图 *G*，可达性查询用于回答从 *G* 中的一个顶点 *u* 出发是否可以到达另一个顶点 *v*，即从 *u* 到 *v* 是否存在路径，因而是多种数据管理应用中的核心操作之一，如 XML 和 RDF 数据库以及社交网络和生物信息网络等，是研究者广泛关注的热点问题[1~15]。在很多实际应用中，如无线传感器网络、互联网、电信网及社交网络等，顶点 *u* 对 *v* 的影响力受制于从 *u* 到 *v* 的路径长度（例如无线传感器网络的广播消息可能在传输过程中的任何一步丢失，其他顶点接收到的概率会根据路径长度以指数级速度衰减），而可达性查询仅能回答从 *u* 到 *v* 是否存在路径，并不能确定路径的长度，因而在类似的应用环境中，仅知道顶点之间是否可达意义不大。

本文研究 *k* 步可达性查询问题。与可达性查询相比，*k* 步可达性查询用于回答从顶点 *u* 到 *v* 是否存在长度为 *k* 步之内的路径；*k* 步可达可以看作可达性查询的一般形式，当 *k*=∞时，*k* 步可达性查询即传统意义上的可达性查询。由于 *k* 步可达性查询能够体现顶点的影响力，因而在许多实际应用中，如传感器网络和社交网络等，可为用户提供比可达性查询更多的信息。

收稿日期：2015-03-10；**修回日期：**2015-05-20

基金项目：国家自然科学基金资助项目（61040023, 61272124, 61303040, 61472339）；河北省教育厅研究计划基金资助项目（Y2012014）

Foundation Items: The Natural National Science Foundation of China (61040023, 61272124, 61303040, 61472339); The Research Funds from Education Department of Hebei Province (Y2012014)

现有求解 k 步可达性查询的方法主要分为 3 类。1) 基于传统可达性求解的方法[1,2,7~9,11]来解决 k 步可达性查询。该类方法的基本思想是结合深度优先遍历和特定的剪枝条件来加速可达性查询。问题在于当 u 的出度变大时，需要访问的顶点变多，效率较低。例如，Human 数据集（ecocyc.org）中，顶点的最大出度为 28 570，无论广度优先还是深度优先遍历，系统性能都会迅速下降。2) 基于最短路径的方法[16]。基本思想是首先求 2 个顶点间的最短距离，之后判断该距离是否大于 k，当最短距离大于 k 值时，说明 k 步不可达，否则说明可达。该方法的问题在于当顶点之间不可达时，求解代价太高。另外，最短路径求解算法本身的复杂度远高于可达性查询算法，也会导致即使 u 可达 v 时，效率也没有第 1 类方法高。3) 基于 k 步索引的方法[17,18]。基本思想是首先求原图的一个顶点覆盖集，基于该覆盖集建立 k 步可达索引，最后基于该索引回答 k 步可达性查询。该类方法的问题在于 k 的取值难以界定。例如，当回答 u 到 v 是否 5 步可达时，如果索引基于 k=3 来构建，则无法回答该查询。同时，当 k=6 时，是否 5 步可达的判断也会变得复杂。如果对所有的 k 值都构建索引，则所需的索引空间会急剧膨胀。

针对以上问题，本文提出一种求解 k 步可达性查询的双向搜索方法，同时结合 2 种剪枝策略来加速查询处理。具体来说，本文的贡献如下。

1) 提出一种双向搜索算法，该算法的基本思想是为每个顶点附加 2 个区间标签，当判断 u 是否满足 k 步可达 v 时，首先比较 u 的出度和 v 的入度，优先处理度小的顶点，同时使用 2 个区间标签进行剪枝，以便快速到达另一个顶点。该方法的优点体现在使用较小的索引，同时提升系统的查询效率，避免由于 u 的出度过大所带来的效率下降问题。

2) 提出基于广度优先遍历的双向广度层数和基于拓扑排序的双向拓扑层数来辅助过滤，减少需要访问的顶点数量。

3) 基于 19 个真实数据集进行测试，实验结果显示，本文所提方法比现有方法更高效。

2　背景知识及相关工作

2.1　数据模型及相关概念

本文用 $G=(V,E)$表示有向无环图，其中，V 表示 G 中的顶点集合，E 表示 G 中有向边的集合，$|V|$ 表示 G 中顶点的个数，$|E|$表示 G 中边的个数。若 $(u,v)\in E$，则说明 G 中有一条从 u 到 v 的边。$inN(v,G)=\{u|(u,v)\in E\}$表示指向顶点 v 的顶点集，即 v 的入度顶点集，$outN(v,G)=\{u|(v,u)\in E\}$表示顶点 v 指向的顶点集，即 v 的出度顶点集。$inDeg(v,G)=|inN(v,G)|$ 表示 v 的入度，$outDeg(v,G)=|outN(v,G)|$ 表示 v 的出度。

对于 G 中给定的 2 个顶点 u 和 v，若从 u 出发存在一条路径可达 v，则说明 u 可达 v，用 $u\rightarrow v$ 来表示；反之说明 u 不可达 v；若从 u 到 v 之间存在一条长度小于等于 k 的路径，即路径上的顶点个数小于等于 k，说明 u 在 k 步之内可达 v，用 $u\rightarrow_k v$ 表示，反之说明 u 在 k 步之内不可达 v。

对于有向图来说，通过离线记录强连通分量中任意两点的距离，可以将该有向图转换为有向无环图，其中，强连通分量中的任意两点之间都是可达的，这时只需要判断它们之间的最短距离就能够得出结果，本文后续讨论中给定的图也都是有向无环图。

2.2　相关工作

2.2.1　基于传统可达性求解的方法

该类方法的基本思想是结合深度优先遍历和特定的剪枝条件来加速可达性查询。其基本做法是为每个顶点 v 附加一个区间 L，该区间能够包含所有 v 可达的顶点对应的区间。查询处理的基本思想是：如果 $v.L\not\subset u.L$，则 u 不可达 v，反之检查 u 指向的每个顶点（后续讨论中称为 u 的孩子顶点）和 v 的可达性关系。如果所有孩子顶点都不可达 v，则 u 不可达 v；否则从每个满足区间包含关系的孩子顶点出发，重复判断其孩子顶点和 v 的可达性关系。由于非树边的存在，使用一个区间可能导致误判的情况，即从区间关系来判断可知 u 的区间包含 v 的区间，但实际上 u 不可达 v。为了降低误判率，文献[2]通过不同的深度优先遍历方式为每个顶点附加 2 个区间 L_1 和 L_2 来减少误判率。其基本思想是：只有当 $u.L_1\supseteq v.L_1\wedge u.L_2\supseteq v.L_2$ 满足时，才检测 u 的孩子顶点和 v 的可达性关系，否则 u 不可达 v。用这种方法来回答 k 步可达性查询时，若 u 和 v 的区间满足，但 u 的出度非常大且可达 v 的孩子顶点很少时，系统需要访问 u 的大量孩子顶点，最终会导致访问大量无用顶点，最坏情况下和基于广度优先遍历来求解 k 步可达性查询的代价一样低效。

建立双区间标签的基本思想是：为每一个顶点 v 附加 2 个区间 $L_v^1=[s_v^1, e_v^1]$和 $L_v^2=[s_v^2, e_v^2]$，其中，

s_v^i 与 e_v^i 分别表示第 i 个区间的开始值与结束值，i=1,2，$s_v^i = \min\{s_x^i \mid x \in outN(v,G)\}$表示 v 可达的顶点在后序遍历时最先访问顶点对应的序号，$e_v^i = post(v)$，表示后序遍历时访问顶点 v 的顺序，直观理解即在深度优先遍历 G 时，顶点 v 被处理结束的顺序。当 $outDeg(v)$=0 时，$s_v^i = e_v^i$。求每个顶点区间标签的过程如下。

1) 从入度为 0 的顶点开始进行深度优先遍历，随机选择一个未处理的孩子顶点进行访问。

2) 当访问到无出度的顶点时，该顶点的 s_v^i 和 e_v^i 为该顶点的后序遍历的顺序值。

3) 当顶点 v 的所有孩子顶点都访问完时，s_v^i 值取其孩子顶点 s_v^i 值的最小值，e_v^i 值等于 v 被处理结束的顺序值。

4) 以上过程递归执行，直到所有的顶点都访问完为止。

第 2 个区间的求解方法如上，这样就得到每个顶点的双区间标签，如图 1 所示。

2.2.2 基于最短路径的方法

该类方法的典型代表是 PLL 方法[16]，其基本思想是为图中的每一个顶点 v 建立 $L_{in}(v)$与 $L_{out}(v)$2 个标签，其中，$L_{in}(v)$中的元素为可达顶点 v 和所对应的最短路径值的二元组集合；类似地，$L_{out}(v)$中的元素为 v 可达的顶点和所对应的最短路径值的二元组集合。其他方法类似，此处不再赘述。

该类方法都是先求出各顶点之间的最短路径，之后再根据查询顶点对的最短路径值和 k 值进行比较，若是最短路径值小于 k 值，则说明 k 步可达，反之则 k 步不可达。该类方法的问题在于当顶点之间不可达时，求解代价太高，并且，最短路径求解算法本身的复杂度远高于可达性查询算法，会致使 k 步可达性的查询效率较低。

2.2.3 基于 k 步索引的方法

文献[17,18]针对 k 步可达性问题，采用最大顶点覆盖的方法求出一个覆盖集 S，即 S 中顶点之间的边覆盖了原始图 G 中所有的边，之后对覆盖集中的每一个顶点求出它 k 步之内可达 S 中的顶点，并求出所对应的距离来建立 k 步可达索引，最后基于该索引回答 k 步可达性查询。如引言部分所述，该类方法的问题在于 k 的取值难以界定。例如，当回答 u 到 v 是否 5 步可达时，如果索引基于 k=3 来构建，则无法回答该查询。同时，当 k=6 时，是否 5 步可达的判断也会变得复杂。如果对所有的 k 值都构建索引，则所需的索引空间会急剧膨胀。另外，由于对 S 中的每个顶点都要建立它和所有 k 步之内可达顶点的边，若覆盖比较稠密时会导致索引规模急剧膨大。虽然作者提出了可扩展的 k 步可达索引来减少索引规模，但查询性能也相应随之降低。

3 双向搜索方法

本节首先介绍提出的双向搜索算法来减少需要访问的顶点数量，简称 BiRch，然后介绍基于广度和

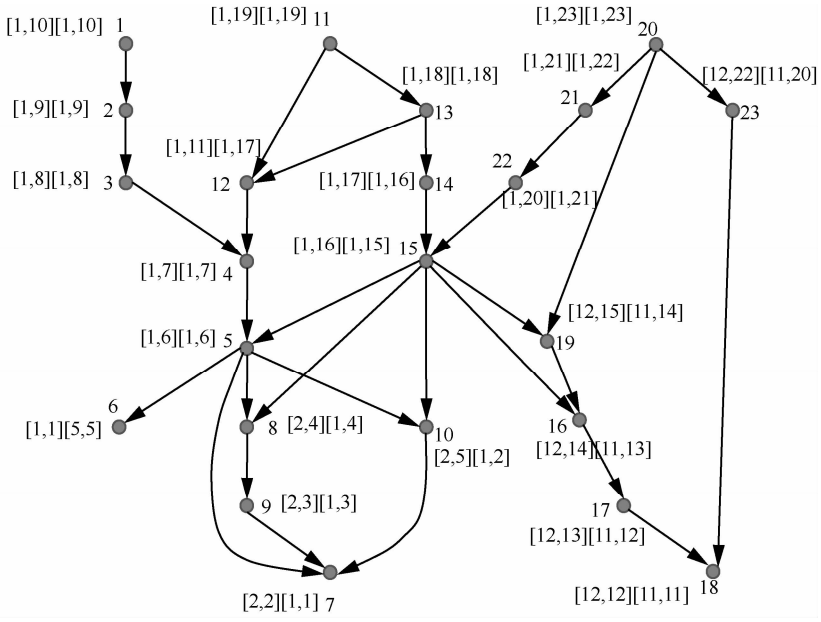


图 1 有向无环图 G 及顶点的区间标签

拓扑层数的 2 种剪枝策略来进一步提升系统性能。

3.1　双向搜索算法 BiRch

和文献[2]类似，本文方法所使用的顶点标签由不同遍历方式所生成的区间构成。和文献[2]不同，当判断 u 是否 k 步可达 v 时，BiRch 的基本思想是：首先判断 v 的 2 个区间是否分别被 u 的 2 个对应区间包含，如果存在一个区间不满足条件，则 u 不可达 v；否则比较 u 的出度和 v 的入度。如果 $outDeg(u,G)\leqslant inDeg(v,G)$，则判断 u 的孩子顶点是否 $k-1$ 步可达 v；否则，判断 u 是否 $k-1$ 步可达指向 v 的顶点（后续讨论中表示为 v 的父亲顶点）。以上操作递归执行直到判断出 u 是否 k 步可达 v，如算法 1 所示。

算法 1　BiRch(u, v, k, G)

1) **if** $u = v$ **then** return true
2) **if** k=0 **then** return false
3) **if** $u.L_1 \not\supseteq v.L_1 \vee u.L_2 \not\supseteq v.L_2$ **then** return false
4) **else**
5)　　**if** $outDeg(u,G)\leqslant inDeg(v,G)$ **then**
6)　　**foreach** $c\in outN(u, G)$ **do**
7)　　**if** BiRch(c, v, $k-1$, G) **then** return true
8)　　**else**
9)　　**foreach** $p\in inN(v, G)$ **do**
10)　　**if** BiRch(u, p, $k-1$, G) **then** return true
11) return false

例 1 考虑图 1 的有向无环图 G。假设查询为 $15\rightarrow_2 17$。由于顶点 15 的出度为 5 而 17 的入度为 1，算法 1 从 17 开始逆向处理顶点 17 的父亲顶点，即判断 $15\rightarrow_1 16$ 是否成立（第 8）和第 9）行）。由于 16 的入度为 2，小于 15 的出度，因此算法 1 继续从顶点 16 开始判断 15 是否在 0 步可达 16 的父亲顶点，即判断 16 的入度顶点中是否存在 15，若有则 2 步之内可达，否则返回 false。由于 16 的入度顶点集包含 15，可知 $15\rightarrow_2 17$ 成立。至此，算法 1 访问的顶点有 17、16、15；如果使用 GRAIL 方法[2]，则需访问的顶点为 15、5、8、10、16、17。假设查询为 $14\rightarrow_2 9$。由于顶点 14 的出度为 1，等于顶点 9 的入度，算法 1 判断 14 的孩子顶点 15 是否满足 $15\rightarrow_1 9$。由于 15 的出度大于 9 的入度，算法 1 转而处理 9 的入度顶点，即判断 $15\rightarrow_0 8$ 是否成立。至此，算法 1 在第 2)行直接返回 false，访问的顶点有 14、15、9、8；而 GRAIL 需要访问的顶点为 14、15、5、8、10、16、19。可见，算法 1 使用的双向搜索方法减少了需要访问顶点的个数。

虽然 BiRch 算法可以避免部分冗余计算，但其最坏情况下的时间复杂度和 GRAIL 算法相同，都是 $O(|V|+|E|)$。同时，索引大小为 $O(|V|)$，构建索引的时间复杂度为 $O(|V|+|E|)$。

3.2　基于双向广度层数的剪枝策略

当 u 的区间包含 v 的区间时，算法 1 需要从 u 或者 v 出发走 k 步才能判断 $u\rightarrow_k v$ 是否成立，本文提出基于双向广度层数的剪枝策略，当从 u 到 v 的最短路径的长度大于 k 时，直接终止，无需额外处理。

定义 1　（正向广度层数）给定有向无环图 G，顶点 v 的正向广度层数 $bre(v)$定义如下

$$bre(v)=\begin{cases}1, inN(v,G)=\phi\\ \min\{bre(u)+1\}u\in inN(v,G), \text{其他}\end{cases} \tag{1}$$

直观来看，入度为 0 的顶点其正向广度层数为 1，否则顶点的正向广度层数等于它的所有入度顶点的正向广度层数加 1 之后取最小值，即顶点 v 的正向广度层数等于对 G 进行广度优先遍历时 v 的父亲顶点的层数加 1。图 1 中每个顶点的正向广度层数如图 2 中每个顶点标签中第一个数字所示。对于顶点 12 来说，它的正向广度层数为 2，是根据顶点 11 的正向广度层数加 1 得来的。基于正向广度层数，有如下结论。

定理 1　给定有向无环图 G 中任意 2 个顶点 u 和 v，若 $bre(v)-bre(u)>k$，则 $u\rightarrow_k v$ 不成立。

证明　假设 $bre(v)-bre(u)>k$ 且 $u\rightarrow_k v$ 成立。根据 k 步可达的定义，若 $u\rightarrow_k v$ 成立，则从 u 到 v 存在长度小于等于 k 的路径，因而从 u 出发广度优先遍历，可知 v 的正向广度层数最多比 u 的正向广度层数大 k，即 $bre(v)-bre(u)\leqslant k$。这和假设矛盾。根据广度优先遍历的性质可知，若 $bre(v)-bre(u)>k$ 成立，则从 u 到 v 的最短路径长度一定大于 k，可知从 u 到 v 不可能 k 步之内到达。

证毕。

正向广度层数可用于快速过滤 u 可达 v，但不满足 k 步条件的查询。考虑查询 $2\rightarrow_2 9$ 是否成立。由于 2 可达 9，因而算法 1 需要访问 2 到 9 的路径上部分顶点才能确定查询 $2\rightarrow_2 9$ 不成立。如果使用正向广度层数，可以在 $O(1)$时间确定 $2\rightarrow_2 9$ 不成立，这是因为顶点 2 的正向广度层数为 2，顶点 9 的正向广度层数

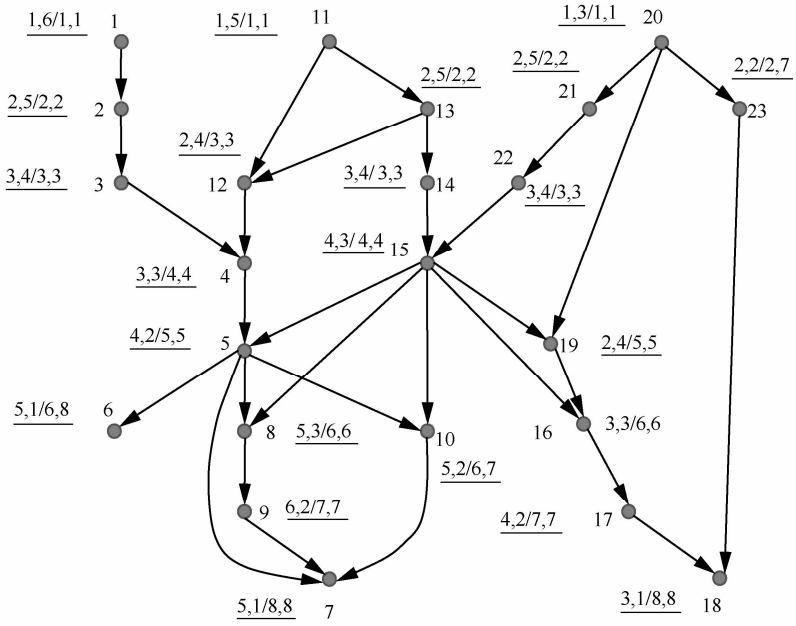


图 2 正向、逆向广度层数/正向、逆向拓扑层数

为 6，6-2>2，因此 $2\rightarrow_2 9$ 不成立。然而，当查询顶点的正向广度层数满足 k 步条件时，单纯使用正向广度层数并不能排除所有不满足条件的情况。例如，考虑查询 $2\rightarrow_2 5$，由于顶点 2 和顶点 5 的正向广度层数差不大于 2，因此不能确定 $2\rightarrow_2 5$ 不成立。但是从图 2 可知，显然 $2\rightarrow_2 5$ 不成立。针对该问题，提出使用逆向广度层数来进行过滤，如定义 2 所示。

定义 2 （逆向广度层数）给定有向无环图 G，顶点 v 的逆向广度层数 $rebre(v)$定义如下

$$rebre(v)=\begin{cases}1, & outN(v,G)=\phi \\ \min\{bre(u)+1 \mid u\in inN(v,G)\}, & \text{其他}\end{cases} \tag{2}$$

直观来看，顶点 v 的逆向广度层数就是将 G 的有向边反转，然后执行正向广度优先遍历得到的正向广度层数。对图 1 进行处理后得到所有顶点的正向和逆向广度层数，分别对应图 2 中每个顶点标签中“/”前面的 2 个数字，其中第 1 个数字表示正向广度层数，第 2 个数字表示逆向广度层数。对于顶点 12 来说，它的正向广度层数和逆向广度层数分别为 2 和 4，其中，正向广度层数是根据顶点 11 的正向广度层数加 1 得来的，逆向广度层数是根据顶点 4 的逆向广度层数加 1 得来的，而顶点 4 的逆向广度层数由顶点 5 的逆向广度层数加 1 得来的。对于顶点 5 来说，其逆向广度层数由顶点 6、7、8 或者 10 的逆向层数加 1 后取最小值得到。由于顶点 6 和 7 的逆向广度层数都是 1，因而可知顶点 5 的逆向广度层数为 2，最终可知顶点 12 的逆向广度层数为 4。基于逆向广度层数，有如下结论。

定理 2 给定有向无环图中的任意 2 个顶点 u 和 v，若 $rebre(u)-rebre(v)>k$，则 $u\rightarrow_k v$ 不成立。

证明 根据广度优先遍历的性质可知，若 $rebre(u)-rebre(v)>k$ 成立，则从 u 到 v 的最短路径长度一定大于 k，可知从 u 到 v 不可能 k 步之内到达。证毕。

之所以使用逆向广度层数，是因为 2 个顶点之间的最短路径即使长度大于 k，也可能存在正向广度层数差较小但逆向广度层数差较大，或者正向差大而逆向差小的情况。需要注意的是，正向和逆向层数可以通过一遍广度优先遍历得到。其具体做法是在广度优先遍历时，算法初次碰到一个顶点时得到的是正向广度层数，当访问完该顶点后得到其逆向广度层数。基于定理 1 和定理 2 所得到的改进算法如算法 2 所示。

算法 2 BiRch-BL(u, v, k, G)

1) **if** $u = v$ **then** return true

2) **if** k=0 **then** return false

3) **if** $u.L_1 \not\supseteq v.L_1 \vee u.L_2 \not\supseteq v.L_2$ **then** return false

4) **if** $bre(v)$-$bre(u)>k$ or $rebre(u)$-$rebre(v)>k$ **then return** false

5) **else**

6) **if** $outDeg(u,G)\leqslant inDeg(v,G)$ **then**

7) **foreach** $c\in outN(u, G)$ **do**

8) **if** BiRch-BL(c, v, $k-1$, G) **then**

return true

9) **else**

10) **foreach** $p \in inN(v, G)$ **do**

11) **if** BiRch-BL(u, p, k−1, G) **then** return true

12) return false

例 2　考虑图 2 的顶点关系。假设查询为 $2 \rightarrow_2 10$，根据二者的正向广度层数差，算法 2 在第 4)行返回 false，因而 $2 \rightarrow_2 10$ 不成立；若查询为 $13 \rightarrow_2 5$，根据 2 个顶点的逆向广度层数差，算法 2 在第 4)行返回 false，因而 $13 \rightarrow_2 5$ 不成立。

3.3　基于双向拓扑层数的剪枝策略

虽然基于双向广度层数的剪枝策略可以快速判断部分顶点之间的不可达性，但是当给定的有向无环图变得稠密时，顶点之间的正向广度层数差和逆向广度层数差都会变小，这时其剪枝效果随之降低。例如给定查询为 $14 \rightarrow_2 3$ 时，通过区间标签无法判断查询是否成立，也不能通过正向广度层数和逆向广度层数来进行过滤，算法 1 和算法 2 都需要通过深度优先遍历的方式来确定 $14 \rightarrow_2 3$ 不成立。针对该问题，提出基于双向拓扑层数的剪枝策略。

定义 3　（正向拓扑层数）给定有向无环图 G，顶点 v 的正向拓扑层数 $topo(v)$定义如下

$$topo(v)=\begin{cases}1, inN(v,G)=\phi \\ \max\{topo(u)+1 \mid u \in inN(v,G)\}, \text{其他}\end{cases} \tag{3}$$

直观来看，如果 v 的入度为 0，则其正向拓扑层数为 1，否则 v 的正向拓扑数为它的所有入度顶点的拓扑层数加 1 之后的最大值，即正向拓扑层数是所有入度为 0 的顶点向上对齐所产生的层数。图 2 中，顶点的正向拓扑层数为其标签中“/”之后的第一个数字，例如顶点 12 有 2 个入度顶点 11 和 13，其对应的正向拓扑层数分别为 1 和 2，因此顶点 12 的正向拓扑层数为 3。

定理 3　给定有向无环图 G 中的 2 个顶点 u 和 v，如果 $topo(u) \geqslant topo(v)$，则 $u \rightarrow_k v$ 不成立。

证明　假设 $topo(u) \geqslant topo(v)$时 $u \rightarrow_k v$ 成立。则从 u 到 v 至少存在一条路径，因此拓扑排序时，v 的访问次序必然在 u 之后，即 $topo(u) < topo(v)$。这和假设矛盾。

证毕。

假设查询为 $12 \rightarrow_2 3$。由于顶点 12 和 3 的正向拓扑层数均为 3，满足 $topo(12) \geqslant topo(3)$，根据定理 3，可知 $12 \rightarrow_2 3$ 不成立。然而，单纯使用正向拓扑层数并不能排除所有不满足条件的情况。例如，考虑查询 $23 \rightarrow_2 16$。由于顶点 23 的正向拓扑层数为 2，而顶点 16 的正向拓扑层数为 6，根据定理 3 无法判断顶点 23 是否 2 步之内可达顶点 16。为此，引入逆向拓扑层数来解决该问题。

定义 4　（逆向拓扑层数）给定有向无环图 G，顶点 v 的逆向拓扑层数 $retopo(v)$定义如下

$$retopo(v)=\begin{cases}d_{\max}, outN(v,G)=\phi \\ \min\{retopo(u)-1 \mid u \in outN(v,G)\}, \text{其他}\end{cases} \tag{4}$$

其中，$d_{\max}$ 表示 G 中最长路径的长度。

直观来看，顶点的逆向拓扑层数是所有出度为 0 的顶点向下对齐所产生的拓扑层数。根据定义 4，所有出度为 0 的顶点其逆向拓扑层数都为 G 中最长路径的长度，其余顶点的逆向拓扑层数为它的所有出度顶点的逆向拓扑层数减 1 之后取其最小值。图 2 中每个顶点的逆向拓扑层数如其标签中最后一个数字所示。如顶点 23 的逆向拓扑层数为 7。对于顶点 15 来说，其出度顶点有 5 个，根据定义 4 可知其逆向拓扑层数为 4。基于逆向拓扑层数的定义，有如下结论。

定理 4　给定有向无环图 G 中的 2 个顶点 u 和 v，如果 $retopo(u) \geqslant retopo(v)$，则 $u \rightarrow_k v$ 不成立。

证明　证明过程和定理 3 相同，此处略。

基于定理 3 和定理 4，结合双向拓扑层数剪枝的算法如下。

算法 3　BiRch-BTL(u, v, k, G)

1) **if** $u = v$ **then** return true

2) **if** k=0 **then** return false

3) **if** $u.L_1 \not\supseteq v.L_1 \vee u.L_2 \not\supseteq v.L_2$ **then** return false

4) **if** $bre(v)$-$bre(u)$>k or $rebre(u)$-$rebre(v)$>k **then** return false

5) **if** $topo(u) \geqslant topo(v)$ or $retopo(u) \geqslant retopo(v)$ **then** return false

6) **else**

7) **if** $outDeg(u,G) \leqslant inDeg(v,G)$ **then**

8) **foreach** $c \in outN(u, G)$ **do**

9) **if** BiRch-BTL(c, v, k−1, G) **then** return true

10) **else**

11) **foreach** $p \in inN(v, G)$ **do**

12) **if** BiRch-BTL(u, p, k−1, G) **then** return true

13) return false

例 3 假设查询为 $23\rightarrow_2 16$。由于通过正向拓扑层数不能确定 $23\rightarrow_2 16$ 是否成立，算法 3 在第 5)行比较顶点 23 和 16 的逆向拓扑层数(分别为 7 和 6)，满足定理 4 的要求，因此确定 $23\rightarrow_2 16$ 不成立。

直观来看，双向广度层数通过拉近顶点之间的层数差，可以过滤掉部分可达但最短路径长度大于 k 步的顶点对；而对于广度层数差在 k 步之内的顶点对，双向拓扑层数则可以过滤掉部分不可达的顶点对。

需要注意的是，虽然以上介绍的 2 种剪枝策略能够加速查询处理的速度，但最坏情况下的时间和空间复杂度都没有变化。

4 实验

4.1 实验环境

所用机器的基本配置包括 Intel(R) Pentium(R) 2.9 GHz CPU，4 GB 内存，500 GB 硬盘和 Windows 7 Professional OS。用于比较的算法包括：1）基于区间标签求可达性的算法 GRAIL[2]，和原文一样，每个顶点的区间个数为 2 个；2）直接处理 k 步可达性查询算法 k-reach[17,18]，和原文一样，k 值为所有顶点对的最短路径的中位数；3）处理最短路径距离查询的当前最好的算法 PLL[16]。所有算法都基于 Microsoft Visual C++实现。为了比较不同算法查询处理的整体性能，实验中对每个数据集随机选取 1×10^5 个查询顶点对进行测试。所有的运行时间(1×10^5 个查询顶点对的总时间)都是算法执行 10 次的平均值。

实验数据由 19 个实际数据集组成，包括 Agrocyc[注1]、Anthra、Ecoo[注1]、Human[注1]、Mtbrv[注1]、Vchocyc[注1]、Amaze[7]、Kegg[7]、Nase[8]、Xmark[8]、Arxiv[注2]、Citeseer[注3]、Pubmed[注4]、Go[注5]、Yago[注6]、citeseerx05p[17,18]、cit-patent05p[17,18]、go-uniprot01p[注7]、go-uniprot[注7]。这些数据集的相关统计信息如表 1 所示，其中最后 4 个数据集规模较大，$|V|$表示给定有向无环图中的顶点数量，$|E|$为边的数量，$indeg_{\max}$ 为顶点的最大入度，$outdeg_{\max}$ 为顶点的最大出度。

表 1 数据集统计信息

数据集	$\|V\|$	$\|E\|$	$indeg_{\max}$	$outdeg_{\max}$
Agrocyc	12 684	13 657	571	5 487
Amaze	3 710	3 947	1 237	1 860
Anthra	12 499	13 327	486	5 400
Arxiv	6 000	66 707	695	173
Citeseer	10 720	44 258	184	57
Ecoo	12 620	13 575	542	5 434
Go	6 793	13 361	6	70
Human	38 811	39 816	553	28 570
Kegg	3 617	4 395	1 234	2 048
Mtbrv	9 602	10 438	467	4 004
Nasa	5 605	6 538	11	30
Pubmed	9 000	40 028	432	87
Vchocyc	9 491	10 345	486	3 916
Xmark	6 080	7 051	192	803
Yago	6 642	42 392	2 370	55
citeseerx05p	1 457 057	3 002 252	53 452	2 774
go-uniprot01p	665 850	4 584 419	1 186 281	170
cit-patent05p	1 671 488	3 303 789	139	128
uniprot22m	1 595 444	1 595 442	1 539 898	1

4.2 索引构建及大小

表 2 和表 3 分别给出不同方法的索引构建时间和索引大小。如表 2 所示，在索引构建时间上，提出的双向搜索方法 BiRch 和 GRAIL 方法一样，对于所有的数据集来说索引时间都是最少的。虽然 BiRch-BL 和 BiRch-BTL 算法在索引上都要比 BiRch 算法的索引时间要长，但是都比 k-reach 方法在大部分数据集上要快一个数量级，比 PLL 方法快大约 2 个数量级。其中，PLL 方法在后面的 4 个大数据集上没有在 12 h 之内出来，本文忽略其时间。索引大小方面，如表 3 所示，BiRch 和 GRAIL 有相同的索引大小，和 k-reach 相比，它们在某些数据集对应的索引比 k-reach 的索引要小，如 Amaze、Arxiv、Pubmed、Citeseer、Go、Kegg、Nasa、Xmark 以及 go-uniprot01p。对其他数据集来说，本文方法的索引比 k-reach 方法的索引大，但是相差不大，并且都比 PLL 方法的索引小。需要说明的是：虽然从每个节点标签大小来看，BiRch-BL 和 BiRch-BTL 是 BiRch 的 1.5 倍和 2 倍，但由于每个节点的广度层数和拓扑层数非常小，可以用一个字节表示，因此与 BiRch 相比，总的索引大小几乎没有增加。

注1 ecocyc.org
注2 arxiv.org
注3 citeseer.ist.psu.edu
注4 pubmedcentral.nih.gov
注5 www.geneontology.org
注6 mpi-inf.mpg.de/yago-naga/yago
注7 www.uniprot.org

表 2　　索引构建时间/ms

数据集	GRAIL	k-reach	PLL	BiRch	BiRch-BL	BiRch-BTL
Agrocyc	0.21	10.43	384.28	0.21	1.25	2.26
Amaze	0.13	24.92	59.27	0.13	0.42	0.78
Anthrax	0.22	8.80	367.91	0.22	1.19	2.25
Arxiv	1.63	1 346.71	2 396.37	1.63	2.32	3.97
Citeseer	2.38	253.32	509.18	2.38	2.86	5.40
Ecoo	0.24	14.21	384.10	0.24	1.27	2.45
Go	0.94	52.60	212.77	0.94	1.45	2.43
Human	0.54	41.91	2 532.34	0.54	3.56	6.64
Kegg	0.21	29.72	58.23	0.21	0.49	0.90
Mtbrv	0.20	8.96	234.67	0.20	0.99	1.87
Nasa	0.36	38.08	131.15	0.36	0.91	1.33
Pubmed	1.39	100.90	517.63	1.39	2.25	4.21
Vchocyc	0.20	9.27	232.18	0.20	0.97	1.74
Xmark	0.27	37.01	141.35	0.27	1.10	1.57
Yago	1.00	13.83	222.66	1.00	1.25	2.36
citeseerx05p	135.49	4 358.08	—	135.49	428.84	784.50
go-uniprot01p	71.85	553 799	—	71.85	178.69	385.30
cit-patent05p	298.31	656.05	—	298.31	815.96	1 402.06
uniprot22m	71.66	359.23	—	71.66	223.88	409.93

表 3　　索引大小/MB

数据集	GRAIL	k-reach	PLL	BiRch	BiRch-BL	BiRch-BTL
Agrocyc	0.32	0.15	0.63	0.32	0.37	0.42
Amaze	0.08	0.18	0.17	0.08	0.09	0.11
Anthrax	0.32	0.15	0.62	0.32	0.36	0.42
Arxiv	0.12	19.00	2.85	0.12	0.14	0.17
Citeseer	0.24	0.84	1.15	0.24	0.28	0.33
Ecoo	0.32	0.16	0.63	0.32	0.36	0.43
Go	0.16	0.23	0.57	0.16	0.18	0.22
Human	1.12	0.46	1.95	1.12	1.23	1.42
Kegg	0.08	0.22	0.17	0.08	0.09	0.11
Mtbrv	0.23	0.12	0.48	0.23	0.27	0.34
Nasa	0.13	0.20	0.35	0.13	0.15	0.15
Pubmed	0.21	1.20	1.25	0.21	0.24	0.28
Vchocyc	0.23	0.12	0.47	0.23	0.26	0.31
Xmark	0.14	0.30	0.37	0.14	0.16	0.19
Yago	0.13	0.09	0.61	0.13	0.15	0.18
citeseerx05p	52.09	44.75	—	52.09	57.78	65.11
go-uniprot01p	17.04	147.46	—	17.04	19.64	22.82
cit-patent05p	60.26	23.51	—	60.26	66.78	74.78
uniprot22m	41.92	19.19	—	41.92	46.60	52.83

4.3 k 步可达性查询的性能

4.3.1 查询时间和访问顶点数量

如表 4 所示，BiRch-BTL 方法几乎在所有数据集上都有最快的响应时间，其中在 Amaze、Arxiv 以及 Kegg 这 3 个数据集上比 GRAIL 快 20~50 倍，在最后 4 个数据集上比 k-reach 快 40 倍以上，在几乎所有数据集上比 PLL 快 10 倍以上。具体原因可以从表 5 的访问顶点数量来解释。从表 5 可知，BiRch-BTL 在几乎所有的数据集上访问顶点数量是最少的，这也进一步证明 BiRch-BTL 的高效性。由于 PLL 方法基于顶点标签求最短距离，无需访问路径中的顶点，因此表 5 中不存在 PLL 方法的访问顶点数量。另外，从表 4 也可以看出，本文方法在 Arxiv 数据集上的运行时间远长于 k-reach，原因在于 Arxiv 数据集非常稠密，导致本文的剪枝效果不够明显，需要访问的顶点数量急剧增加，因而所需时间也较长。

4.3.2 不同 k 值的查询处理性能

表 6 给出了 BiRch-BTL 算法在不同特征的数据集上执行 1×10^5 个查询时对应于不同 k 值的查询时间。其余数据集得到的趋势和这 6 个数据集中的趋势类似，在此不一一列举。表 6 的结果显示，对于稀疏数据集来说（前 5 个数据集），k 值的变化对算法性能影响不大，原因在于这时的剪枝效果较好。当数据集变得稠密后（如 Arxiv 数据集），算法性能会随着 k 值的增大而下降，原因在于剪枝效果变差，需要访问的顶点数据变多，从而导致性能下降。需要注意的是，当 k 值变化时，同一个查询的返回值可能会发生变化。

5 结束语

针对现有方法在处理 k 步可达性查询时索引规模庞大、效率低的问题，首先提出一种双向搜索算法 BiRch，当判断顶点 u 是否满足 k 步可达顶点 v 时，优先处理度小的顶点，从而减小了索引规模、提升了系统效率、避免了由于 u 的出度过大所带来的效率下降问题；进而提出了基于双向广度层数和双向拓扑层数的剪枝策略来提升系统响应速度。基于大量真实数据集的实验结果验证了本文方法的高效性。

表 4 **查询时间/ms**

数据集	GRAIL	k-reach	PLL	BiRch	BiRch-BL	BiRch-BTL
Agrocyc	1.71	1.47	21.11	0.84	0.81	0.78
Amaze	82.42	2.83	12.36	2.59	2.30	2.16
Anthrax	1.58	1.32	21.46	0.80	0.79	0.77
Arxiv	11 115.90	36.72	212.76	612.38	606.12	447.07
Citeseer	14.36	12.27	50.04	8.25	8.04	6.01
Ecoo	1.90	1.68	20.34	0.85	0.81	0.80
Go	3.12	6.72	27.53	2.81	2.59	2.49
Human	1.94	1.56	30.57	0.81	0.81	0.81
Kegg	107.15	3.41	12.51	2.68	2.54	2.41
Mtbrv	1.56	1.62	18.10	0.82	0.80	0.78
Nasa	2.48	4.34	18.20	1.96	1.90	1.87
Pubmed	13.43	10.48	58.31	6.56	6.48	5.39
Vchocyc	1.58	1.49	17.94	0.81	0.80	0.80
Xmark	14.71	3.94	17.74	2.31	2.28	2.24
Yago	3.61	2.87	31.27	2.37	2.32	1.43
citeseerx05p	10.71	903.82	—	10.37	10.03	6.24
go-uniprot01p	7.88	17 205.10	—	8.29	7.95	4.87
cit-patent05p	1.96	80.75	—	2.63	2.52	2.16
uniprot22m	3.34	109.22	—	2.56	2.47	0.98

表 5　　**访问顶点数量**

数据集	GRAIL	*k*-reach	BiRch	BiRch-BL	BiRch-BTL
Agrocyc	142 386	110 749	100 345	100 250	100 238
Amaze	16 351 674	149 480	138 656	133 410	132 312
Anthrax	138 568	109 644	100 308	100 206	100 200
Arxiv	1 929 333 067	499 104	80 738 948	75 887 342	55 008 203
Citeseer	1 294 447	330 228	479 353	470 275	328 549
Ecoo	142 846	110 792	100 368	100 301	100 290
Go	248 098	208 329	135 048	125 114	115 006
Human	170 257	103 908	100 031	100 014	100 014
Kegg	21 569 254	163 295	146 498	137 430	136 138
Mtbrv	143 470	112 730	100 724	100 535	100 505
Nasa	134 389	159 186	114 462	113 665	112 312
Pubmed	1 507 629	349 759	412 416	410 508	312 118
Vchocyc	143 824	113 120	100 465	100 349	100 344
Xmark	1 904 054	157 893	120 736	119 064	117 099
Yago	372 986	172 977	139 888	133 178	125 328
citeseerx05p	841 455	1 589 887	349 634	349 465	244 080
go-uniprot01p	366 756	731 755	254 863	247 256	180 385
cit-patent05p	118 752	183 404	114 805	114 793	106 802
uniprot22m	151 027	2 018 312	101 077	101 077	100 027

表 6　　**BiRch-BTL 在不同 *k* 值的查询时间/ms**

数据集	*k*=2	*k*=3	*k*=4	*k*=5	*k*=6	*k*=7	*k*=8	*k*=9	*k*=10
uniprot22m	0.82	0.81	0.81	0.81	0.81	0.81	0.81	0.81	0.82
Ecoo	0.82	0.82	0.81	0.80	0.80	0.80	0.80	0.80	0.80
cit-patent05p	1.58	1.55	1.51	1.53	1.50	1.53	1.56	1.55	1.56
Pubmed	3.49	4.54	5.41	5.81	5.84	5.91	5.90	5.93	5.93
Yago	1.47	1.54	1.55	1.54	1.55	1.57	1.57	1.57	1.57
Arxiv	14.59	57.79	177.61	436.53	734.74	779.13	934.02	980.15	1061

参考文献：

[1] AGRAWAL R, BORGIDA A, JAGADISH H V. Efficient management of transitive relationships in large data and knowledge bases[A]. Special Interest Group on Management of Data Conference(SIGMOD)[C]. 1989. 253-262.

[2] YILDIRIM H, CHAOJI V, ZAKI M J. Grail: scalable reachability index for large graphs[J]. PVLDB Journal, 2010, 3(1): 276-284.

[3] CHENG J, HUANG S, WU H. TF-label: a topological-folding labeling scheme for reachability querying in a large graph[A]. Special Interest Group on Management of Data Conference(SIGMOD)[C]. New York, 2013. 193-204.

[4] YANO Y, AKIBA T, IWATA Y. Fast and scalable reachability queries on graphs by pruned labeling with landmarks and paths[A]. International Conference on Information and Knowledge Management (CIKM)[C]. San Francisco, CA, USA, 2013.1601-1606.

[5] CHEN Y, CHEN Y. An efficient algorithm for answering graph reachability queries[A]. IEEE 24th International Conference on Data Engineering(ICDE)[C]. 2008.893-902.

[6] JIN R, RUAN N, DEY S. SCARAB: Scaling reachability computation on large graphs[A]. Special Interest Group on Management of Data Conference (SIGMOD)[C]. Scottsdale. 2012.169-180.

[7] TRIßL S, LESER U. Fast and practical indexing and querying of very large graphs[A]. Special Interest Group on Management of Data Con-

ference(SIGMOD)[C]. Beijing, China. 2007. 845–856.

[8] JIN R, XIANG Y, RUAN N. Efficiently answering reachability queries on very large directed graphs[A]. Special Interest Group on Management of Data Conference(SIGMOD)[C]. Vancouver, 2008. 595-608.

[9] WANG H, HE H, YANG J. Dual labeling: answering graph reachability queries in constant time[A]. International Conference on Data Engineering(ICDE)[C]. Atlanta, GA, USA, 2006.

[10] CHENG J, YU J X, LIN X. Fast computing reachability labelings for large graphs with high compression rate[A]. International Conference on Extending Database Technology (EDBT)[C]. 2008. 193-204.

[11] VAN SCHAIK S J, DE MOOR O. A memory efficient reachability data structure through bit vector compression[A]. Special Interest Group on Management of Data Conference (SIGMOD)[C]. Athens, 2011. 913-924.

[12] ZHU L, CHOI B, HE B. A uniform framework for ad-hoc indexes to answer reachability queries on large graphs[A]. International Conference on Database Systems for Advanced Applications(DASFAA)[C]. Brisbane, Australia, 2009.138-152.

[13] CHEN Y, CHEN Y. Decomposing DAGs into spanning trees: a new way to compress transitive closures[A]. International Conference on Data Engineering(ICDE)[C]. Hannover, Germany, 2011. 1007-1018.

[14] 李艳, 孙乐, 朱怀忠. 网树求解有向无环图中具有长度约束的简单路径和最长路径问题[J]. 计算机学报, 2012, 35(10): 2194-2203.
LI Y, SUN L, ZHU H Z. A nettree for simple paths with legngth constraint and the longest path in directed acyclic graphs[J]. Chinese Journal of Computer, 2012, 35(10):2194-2203.

[15] CHEN L, GUPTA A, KURUL M E. Stack-based algorithms for pattern matching on DAGs[A]. International Conference on Very Large Data Bases(VLDB)[C]. Trondheim, Norway, 2005. 493-504.

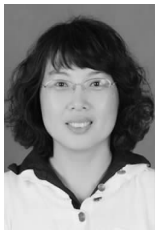
[16] AKIBA T, IWATA Y, YOSHIDA Y. Fast exact shortest-path distance queries on large networks by pruned landmark labeling[A]. Special Interest Group on Management of Data Conference(SIGMOD)[C]. New York, 2013. 349-360.

[17] CHENG J, SHANG Z, CHENG H. *K*-reach: who is in your small world[J]. PVLDB, Journal, 2012, 5(11):1292-1303.

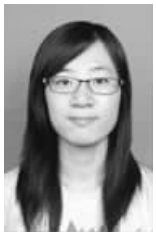
[18] CHENG J, SHANG Z, CHENG H. Efficient processing of *k*-hop reachability queries[J]. VLDB Journal, 2014, 23(2):227-252.

作者简介：

周军锋（1977-），男，陕西西安人，博士，燕山大学教授，主要研究方向为 XML 数据库、字符串相似匹配等。



陈伟（1980-），女，河北秦皇岛人，燕山大学博士生，主要研究方向为数据库理论。



费春苹（1990-），女，河北秦皇岛人，燕山大学硕士生，主要研究方向为图数据管理。

陈子阳（1973-），男，黑龙江五常人，博士、燕山大学教授、博士生导师，主要研究方向为数据库理论与系统等。