复杂网络中近似最短路径问题①

刘 微, 肖华勇

(西北工业大学 理学院, 西安 710072)

摘 要: 随着网络规模的不断增大, 经典算法(如 Dijkstra 等)效率越来越低. 针对这一问题, 研究者们提出了许多近似搜索算法, 但如何既能提高搜索效率又能保持准确性一直是一大难点. 本文根据复杂网络的结构特性引入区域划分,同时改进树分解的构造, 将图构造成一棵树进行搜索, 得到了一个新的适合于复杂网络的最短路径近似算法. 此外通过实例验证, 该算法不仅在一定程度上降低了计算复杂性, 而且保持了较高的近似准确性.

关键词: 复杂网络; 树分解; 树宽; 树; 最短路径近似算法

Approximate Shortest Path Problem in Complex Networks

LIU Wei, XIAO Hua-Yong

(Science of School, Northwestern Polytechnical University, Xi'an 710072, China)

Abstract: With the increasing of data in complex networks, the efficiency of classic algorithms such as Dijkstra algorithm is very low. To solve this problem, the researchers put forward a number of approximate search algorithms, but how to reduce the computational complexity and also keep the accuracy has been a big difficulty. According to the structural characteristics of complex networks, this article introduces regional division, improves the structure of the tree decomposition, and constructs graph into a tree to search the target vertex, getting a new the shortest path approximate algorithm for complex networks. In addition, the proposed algorithm not only reduces the computational complexity but also remains the high accuracy of approximation by examples.

Key words: complex network; tree decomposition; tree-width; tree; shortest path approximate algorithm

1 引言

在图论中, 计算一对顶点之间的最短路径是一个基本的图算法问题. 本文将解决的就是复杂网络中点与点之间的(近似)最短路径查询问题. 在最短路径问题研究的发展中, 出现了各式各样的最短路径查询方法, 最经典的方法是 Dijkstra 算法[1], 广度优先搜索算法. 这些方法计算精确, 但对于大规模复杂网络花费时间太长. 近年来, 研究们也提出了不少新的算法. Matthew J. Rattigan 等人[2]提出了基于图结构索引(network structure indices,简称 NSI)的 *DTZ* 算法,其花费时间 $O(ekd)$. 该算法利用保存的预处理结果来快速近似节点间的距离. Wei[5]提出了一种基于树分解的最短路径索引和查询处理方法. 其预处理时间为 $O(n^2)$, 而查询时间为 $O(w^2h)$, 很显然, 其树宽和树高都对复杂度有影响. 在此基础上, Takuya[4]等人提出了针对于小树宽的复杂网络的新的最短路径查询处理方法, 其预处理时间最坏的情况为 $O(nm+bw^2)$, 查询处理时间 $O(w^2\sqrt{h})$, 改进后的混合近似算法查询处理时间 $O(w^2\sqrt{h}+wd)$, 但网络的稀疏程度的变化会大大影响 Takuya 的效率. 其中 w 为树分解的宽度参数, n 为节点数, m 为边数, b 为 *bag* 数.

本文算法的基本思想是将图构造成一棵树来搜索目标顶点. 其文章布局为: 本文的第二部分, 基础知识. 第三部分, 详细的最短路径算法和复杂度分析. 第四部分, 实验结果与分析.

① 基金项目:国家磁约束核聚变能发展专项

收稿时间:2015-08-26;收到修改稿时间:2015-11-02

2 基础知识

令图 $G=(V,E)$,其中 V 为图 G 的点集, E 为图 G 的边集, 设 u,v 属于 V, u 到 v 的最短距离即为 u 到 v 的路径中最短路径的路径长度. 设点 v 在 G 中的邻居节点集合为 $N_G(v)$, 即就是, 若点 v 在 G 中的邻居节点为 $v_1,v_2,...v_n$, 则 $N_G(v)=\{v_1,v_2,...v_n\}$.

对于无权图, 两点之间的最短距离为其最短路径步数之和, 而对于加权图, 其两点间的最短距离为最短路径上的权值之和. 本文先考虑无权无向图.

定义 1. 树分解和树宽[6][7]

图 G 的树分解表示为 (T,X),其中 T 为一树, $X=\{X_t\in V(T)\}$ 是 $V(G)$ 的子集族. 树节点 X_i 称为 *bag*. 树分解具有以下特性:

① $\bigcup_{t\in V(T)} X_t=V(G)$

② 对于任意 $\forall(u,v)\in E(G)$, 存在 $t\in V(T)$ 使得 $u,v\in X_t$.

③对于所有 $u\in V(G)$, 集合 $\{t\mid u\in X_t\}$ 生成 T 的子树.

树分解的宽度定义为 $\max_{t\in V(T)}\{|X_t|-1\}$, 所有图 G 的树分解的宽度的最小值称为 G 的树宽, 用 $w(G)$ 表示, 简写为 w.

定理 1[5]. 给定一图 G 的树分解 $T_G=(\{X_i\mid i\in I\},T)$, T_u, T_v 分别是包含 u,v 的 *bag* 构成的子树, 其中 R_u R_v 分别是这一子树的根. L_{uv} 是 R_u,R_v 的最近共同祖先 *bag*. 从顶点 u 到 v 的任意路径必至少经过 L_{uv} 中的至少一个顶点.

定理 2. 给定一图 G 的树分解 $T_G=(\{X_i\mid i\in I\},T)$, 其中 T 为一树, 树宽为 w, 需要时间 $O(w^2\log(h))$ 即可查询 s 到 t 的最短距离.

证明: 根据树的结构特点, 在树分解 T_G 上自下而上来进行查询, 以得到从 s 到 t 的最短路径和距离. 对于每个节点 $t\in V(T)$,令 *depth(t)*为 t 的深度.

由定理 1 可知,

$$d_G(s,t)=\min_{v\in X}\{d_G(s,v)+d_G(v,t)\}$$

其中 $d_G(s,t)$ 为图 G 中 s 到 t 的最短距离. 假设 T_G 上的 *bag* 中的每对顶点的最短路径和最短距离已预计算出.

首先, 从 s 开始查询, 设 s 所在的 *bag* 为 B_s, 对于任意节点 $u\in B_s$ 由预计算可得最短距离 $d(s,u)$. 接下来, 我们视 B_s 作为孩子节点, 查找它的父节点, 直到执行至最近共同祖先 bag L_{st}. 同理可得 t 的查询过程.

在每一步处理过程中,从 s 到在当前 *bag* 中所有顶点的最短距离计算需要 w^2, 其中 w 为当前 *bag* 的宽度. 假设 B_c 是当前 *bag* 节点, B_p 是它的父 *bag* 节点, 对于每个 $u,v\in B_c$ 我们已经得到 $d(u,v)$. 若已知从 s 到 B_c 的每一个顶点的最短距离, 计算从 s 到 B_p 中顶点的最短距离如下:

1) 对于每个顶点 $v\in B_c\cap B_p$, 由预计算即可的 $d(s,v)$.

2) 对每个顶点 $v\in B_p\setminus B_c$, 即有 $d(s,v)=\min(d(s,u)+d(u,v))$, 其中 $u\in B_p\cap B_c$.

由以上查询可以看出, 若自下而上查询, 每一步只查询其父节点, 则最多需要 $depth(s)+depth(t)$ 步; 但若预处理中计算出从 R_s 中的所有节点到它的第 k^l 祖先 *bag* 的最短路径(其中 k 为常数, $s\in V(T)$, $0\le l\le\log_k(\text{depth}(s))$, $depth(s)$ 可被 k^l 整除), 那么我们就能够从每个 *bag* 跳到它的第 k^l 祖先, 重复跳直到最高祖先 *bag* L_{st}, 则步数最多为 $\log_k h$. 由此可见, 若在构造树分解中每一非终端树节点有至少 k^l 个孩子, 每一步等价于 k^l 跳, 即就是一次查询 *bag* 数为 k^l 到 b/k^l (一般取 $k=2$, $0\le l\le 2$, 当 $l=0$ 时, 即为至少有 1 个孩子节点, 所有非终端树节点都满足), 查询过程就最多有 $2\log_k h$ 步, 而每一步更新最短路径花费为 $O(w^2)$, 因此总的查询时间为 $O(w^2\log h)$.

3 最短路径算法和复杂度分析

3.1 最短路径算法

本文基于树分解算法提出了新的“根子树区域式”树分解搜索算法. 由于在一颗树中, 除了根节点外, 每个子节点可以分为多个不相交的子树, 本文利用树的这一特性, 应用树分解, 构造一个新的以 *bag* 为树节点的树结构.

首先以中心点集合为树根节点, 将图 G 划分为若干个子图, 然后子图通过树分解构成树根节点的子树(在本文中称为“根子树”). 即就是, 各“根子树”区域图的树分解为树根节点的子树. 这样, 经过“根子树”区域划分和树分解后, 形成了新的树结构: 以中心点作为树根节点, 而中心点对应的“根子树”区域网络的树分解形成的树构成互不相交的子树.

与以往算法类似, 其算法也分为两个阶段: 预处理阶段和在线查询阶段. 所谓的预处理算法是指计算某些信息(数据结构等)为下一阶段做准备. 预处理阶

段后的在线查询阶段即为尽快搜索源点到目标顶点的最短路径或距离.

3.1.1 预处理阶段处理过程

预处理阶段分为以下几部分

① 选择中心顶点 $c_1,c_2...c_k$ 作为树根节点 G_0, $G_0=\{c_1,c_2...c_k\}$.

② 构造子图

("根子树"区域划分算法)以中心顶点开始用广度优先遍历将图 G 划分为 k 个"根子树"区域, 记为 $Z_1,Z_2...Z_K$. $Z_1,Z_2...Z_K$ 其对应的节点集为子树的顶点集(子树). 同时记录每个"根子树"区域中顶点到其对应中心顶点的距离, 并存入矩阵 $D_i(i=1,2...k)$.

③ 树分解

对每个"根子树"区域图 $Z_1,Z_2...Z_K$ 进行收缩, 得到收缩图 $G_1,G_2...G_K$. $G_1,G_2...G_K$ 分别作为树根节点 G_0 的直接孩子节点. 树分解算法则采用顺序消去启发式思想, 首先设定一个阀值(如删除顶点度 $<m$), 将"根子树"区域 i 中顶点按度由小到大顺序排列, 依次判断.

如果顶点满足这一阀值条件, 则邻居顶点之间添加边, 使其与邻居顶点构成完全子图, 并删除该顶点 u 和它的所有的边. 直到得到收缩图 G_i (如图 1($i\to iv$), 收缩图由顶点 1 3 5 构成). 收缩图 G_i 作为树分解的根节点 M_i, 收缩过程中构造的每个完全子图所对应的节点和边构成一个 bag, 作为树节点

设 u 为删除顶点, 对应图 G_i 中, 树节点 $(u,u_1,u_2...u_n)$, $(u_1,u_2...u_n)$ 则为 u 的邻居顶点集 $N_{G_i}(u)$. 而包含 $N_{G_i}(u)$ 且不包含 u 的树节点为树节点 $(u,u_1,u_2...u_n)$ 的直接父节点, 重复这一过程即得其树分解(如图 1). 我们将 i "根子树"区域所构成的树分解称为根节点的 i 子树, 用 $R_T(i)$ 表示.

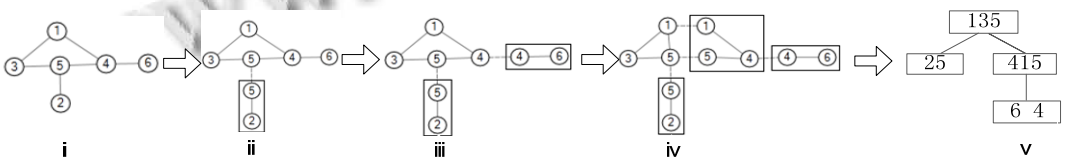

图 1 i 图的树分解过程

④ 局部距离求解

首先, 计算每一个"根子树"区域的树分解 $R_T(i)$ 中每个 bag 的顶点间的距离. 由以上结构可知, 删除的节点和增加的边使 bag 中顶点构成完全子图, 且 bag 中的其它节点是删除节点的邻居节点, 则只需计算邻居节点之间的关系.

本文求解 bag 内局部距离方法为:

首先定义若 v 在第 i 个"根子树"区域, 则其所在的 bag 位置, 用 $bag_i(v)$ 表示. $bag_i[n]$ 表示第 i "根子树"区域中第 n 个 bag. 设 u,v 为同一 bag 节点 $bag_i[n]$ 中, $bag_i[n]$ 中的删除顶点为 t, 且在 G 中 u,v, 没有直接相连, 则

$$d(u,v)=\min_{u,v\in bag_i[n]}\{(d(u,t)+d(t,v)\}$$

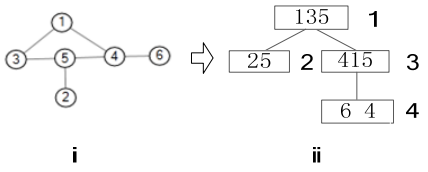

图 2 i 图的树分解结果

如图 2 中求 $bag[3]$ 中 1 到 5 的距离, 则由图 1 树分解过程可看出,

$$d(1,5)=\min_{1,5\in bag[3]}\{(d(1,4)+d(4,5)\}=2$$

而对于计算任意两个中心顶点之间的最短距离, 本文用"根子树"区域间关系来判断"根子树"区域之间的距离. 当"根子树"区域直接相邻, 即中心顶点 c_i,c_j 所在区域 Z_i,Z_j 相邻, 则

$$\begin{aligned}&d(c_i,c_j)\\&=\min\{(D(c_i,v)+d(v,t)+D(t,c_i)\}\\&=\min\{(D(c_i,v)+D(t,c_i)\}+1\end{aligned}$$

其中 v 在区域 Z_i 中, t 在区域 Z_j 中, 且 v 和 t 相连. 因为相邻"根子树"区域的边是中心点之间的"必经之路"; 对于未直接相邻的"根子树"区域的中心点则以相邻"根子树"区域中心顶点作为桥梁来估计中心点间的关系, 若 c_{m_0},c_{m_n} 所在区域不相邻, 即有

$$\begin{aligned}d(c_{m_0},c_{m_n})=&\min\{(D(c_{m_0},c_{m_1})+D(c_{m_1},c_{m_2})\\&...+D(c_{m_{n-1}},c_{m_n})\}\end{aligned}$$

其中, 中心顶点 $c_{m_i},c_{m_{i+1}}$ 所在区域相邻.

中心点选择方法: 将顶点度由小到大排序, 选择

度最大的前 k 个顶点为中心顶点. 原因在于复杂网络的无标度性[8,9], 表明其绝大多数节点的度相对很低, 且存在少数度相对较高的节点. 其度大的节点为复杂网络的 hub 点[10], 作为中心顶点.

注意: m 用于限制每个 bag 的大小, 各“根子树”区域节点结构不同, 因此需要对不同“根子树”区域设定不同的 m 值. 由文献[3]可知, 根节点的大小随 m 的增大, 迅速减小, 但当 m 达到一定值时, m 对于根节点大小影响微弱.

当然, 若预处理阶段步骤 2 中的“根子树”区域点到中心顶点距离不超过 1, 即 $\max(D_i)\le 1$, 那么将自组成 bag , 即就是无须再树分解. 这将大大提高预处理效率, 且精确度未减小.

3.1.2 在线查询处理过程

1) u,v 都存在于树根节点 G_0 中, 预处理中查询即可得到 $d(u,v)$.

2) u,v 在同一 $R_T(i)$ (“根子树”区域 i)中.

① u,v 在同一 $R_T(i)$ 的同一 bag 中, 预处理中查询即可得到 $d(u,v)$.

② u,v 在同一 $R_T(i)$ 中, 但在不同 bag 中. 在这种情况下, 分以下两种情况.

如果 u 或 v 在 $R_T(i)$ 的根树节点 M_i 中. 设 u 不属于 M_i , v 属于 M_i .

$$\begin{aligned} d(u,v) &= \min\{\min_{t_i\in M_i}(d(u,t_i)+d(t_i,v)),\\ &\qquad D(u,c_i)+D(v,c_i)\}\\ &\le \min_{t_i\in M_i\cap bag_i(v),v\in\min_{bag_i(v)\in T_i} bag_i(v)}\\ &\quad \{(d(u,t_i)+d(t_i,v)),D(u,c_i)+D(v,c_i)\} \end{aligned}$$

其中 $bag_i(v)$:表示 v 所在第 i 个“根子树”区域的 bag 位置; $bag_i[u]$:表示 i “根子树”区域中第 n 个 bag . 例, 图 2 中 3 到 4 的最短路径查询: 由图可知(取 5 为中心点) $bag_i(3)=\{1\}$, $bag_i(4)=\{3,4\}$,

$$\min(bag_i(4))=3,\ bag_i[3]\cap bag_i[1]=\{1,5\}$$

$$\begin{aligned} d(3,4) &= \min_{t_i\in M_i\,4\in\min_{bag_i(4)\in T_i} bag_i(4)}\{d(3,t_i)\\ &\qquad +d(t_i,4)\}\\ &= \min\{(d(3,5)+d(5,4)),\\ &\qquad (d(3,1)+d(1,4))\} \end{aligned}$$

则 $d(3,4)=2$.

如果 u 和 v 都不在 M_i 中, 则先获取最近祖先(least common ancestors[3]) bag L_{uv} . 这一过程中, 我们对文献[4][5]中树分解查询进行改进.

$$\begin{aligned} d(u,v) = \min_{t_i,t_j\in L_{uv}}\{&(d(u,t_i)+d(t_i,t_j)+d(t_j,v)),\\ &D(u,c_i)+D(v,c_i)\} \end{aligned}$$

在相距较远两节点最近共同祖先查询中, 往往查询到中心节点为最近共同祖先. 因为中心节点为网络 hub 节点, 根据 hub 节点性质, 网络最短路径通过 hub 节点的比率最大, 因此未影响其精度, 同时算法已预计算出 $D(u,c_i)$ 和 $D(v,c_i)$, 显然这一改进大大提高查询效率.

3) u,v 在不同“根子树”区域时, u 与所在“根子树”区域中心点 c_i 的距离 $D(u,c_i)$, v 与所在“根子树”区域中心点 c_j 的距离 $D(v,c_i)$,

$$\begin{aligned} d(u,v) &\le \min\{D(u,c_i)+d(c_i,c_j)+D(c_j,v)\}\\ &= d(c_i,c_j)+\min\{D(u,c_i)+D(c_j,v)\} \end{aligned}$$

其中 $D(u,c_i)$ 和 $D(v,c_i)$ 由预处理中的步骤 2 可得, 而 $d(c_i,c_j)$ 在预处理阶段步骤 5 中已经求得. 如果“根子树”区域相邻则直接遍历, 若“根子树”区域不相邻, 则遍历到相邻“根子树”区域中心点, 再遍历到目标中心点, 即

$$d(c_i,c_j)\le \min\{d(c_i,c_m)+d(c_m,c_j)\}$$

其中 c_m 为相邻“根子树”区域中心点.

3.2 复杂度分析

在 TEDI 算法中, 其预处理的时间复杂度为 $O(n^2)$, Takuya 其预处理时间最坏的情况为 $O(ne+bw^2)$,本文预处理时间包括构造树结构和局部距离. 其中广度遍历算法划分区域并计算区间节点与中心节点的局部距离最坏情况下需要时间 $O(n)$, 其中, 中心点间距离需要 $O(k^2)$, 图收缩过程中需要 $O(n_i+b_iw_i^2)$, 树结构即为在前一阶段基础上计算每个 bag 的父节点, 需时间 $O(b_iw_i)$, 其中 $n_i(1\le i\le k)$ 为每个“根子树”区域的节点数, k 为“根子树”区域个数, 且 $\sum n_i=n$. 由于复杂网络的无标度性, 则 $k\ll n$, 所以 $O(k^2)$ 可以忽略, 则本文预处理时间为 $O(n)+\sum_{i=1}^{k}O(b_iw_i^2)$. 又 $k>1$, 显然, 本文对 TEDI 和 Takuya 算法在预处理时间复杂度上有了一定的减小.

而对于查询节点时间复杂度, TEDI 查询时间复杂度为 $O(w^2h)$, Takuya 查询处理方法 $O(w^2\sqrt{h})$, 改进后的混合近似算法查询处理时间 $O(w^2\sqrt{h}+wd)$, 由定理 2 可知, 每个“根子树”区域时间复杂度为 $O((w_i^2\log h_i))$, 则本文的查询时间复杂度为 $O(\max_{i=1}^{k}(w_i^2\log h_i))$, 实际上最坏情况也要小于 $O(\max_{i=1}^{k}(w_i^2\log h_i))$.

4 实验结果与分析

本文使用一些实际网络: Power[11]是一个美国西部的国家电力网络, Erdos[3]为社会网络, hep[12]为学术引用图, dip[13]为蛋白质相互作用数据库(详细数据信息见表 1),利用本文提出的算法, 与 TEDI 算法及 Takuya 的精确算法和混合近似算法通过仿真实验进行比较分析, 所有算法通过 Matlab 编程实现.

首先本文应用实际网络来探究 r 取值与搜索时间的关系. r 取值与预处理时间的关系如图 3, 由图可见, r 取值不同, 搜索时间就不同, 且大致呈现 U 型曲线, 很显然存在极小值点.

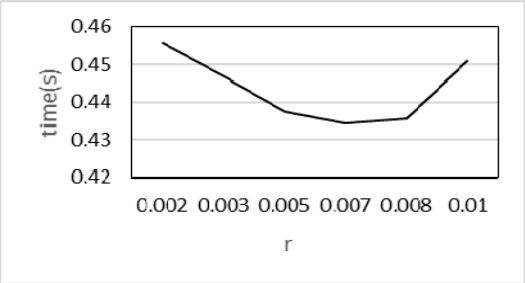

图 3 r 取值与预处理时间的关系

表 1 数据信息

Dataset	Vertices	Edges	r
Power	4941	6594	0.007
Erdio	6927	11850	0.005
Dip	19928	82406	0.0035
Hep	27400	705084	0.0007

结果比较见表 2.

表 2 预处理时间比较

	预处理时间(s)			
		Takuya		
Dataset	Ours	Exact	Hybrid	TEDI
Power	0.8987	1.6108	0.7262	2.3996
Erdio	0.1272	0.8192	0.2310	0.2641
Dip	12.2060	27.9688	10.5477	21.2673
Hep	21.5053	46.9638	21.3894	50.8147

为了取得更好的查询结果, 我们对每个网络取其 r 的极小值的同时, 在网络的每个“根子树”区域中取最优的树宽参数. 其预处理时间的比较见表 2. 在查询过程中, 对于每个网络, 随机抽取 10%的节点对, 得到的查询时间取平均值作为查询时间. 查询处理时间的比较结果见表 3.

表 3 查询时间比较

	查询时间(us)			
		Takuya		
Dataset	Ours	Exact	Hybrid	TEDI
power	0.298	1.825	0.286	1.697
Erdio	0.127	1.482	0.087	2.037
dip	0.708	1.462	1.320	1.772
hep	3.504	5.081	2.926	6.984

由上述实验仿真结果可知, 相较于 TEDI 算法和 Takuya 准确算法, 本文算法对预处理和查询过程的速度都有一定程度的加快, 尤其是对于较大规模的无标度性网络效率有明显提高.

4.2 算法的准确性分析

为了全面分析算法的性能, 我们对算法的准确度进行评测分析. 本文在 Dijkstra 算法作为精准值得基础上, 使用常用的距离近似方法的正确性的评测方法-平均路径比(PathRatio)[2] 对算法准确度进行评估. 其定义如下

$$P=\sum_{i=1}^{n}P_{f_i}\Big/\sum_{i=1}^{n}P_{o_i}$$

其中 n 表示随机抽取的节点对数, 对于任意节点对 i, P_{f_i}、P_{o_i} 分别表示精确距离和近似距离. 由定义可知, P 值为 1, 即为精确值; P 越靠近 1, 精确度越高; 反之, 精确度越低.

使用这一标准对近似方法准确性进行评测, 得到近似方法在以下四种网络上的精度计算结果如下表所示

表 4 近似算法在不同网络中的平均路径比值

Algorithm	Power	Erdio	Dip	Hep
Ours	1.012	1.009	1.012	1.010
Takuya	1.046	1.062	1.130	1.097

由表 2、表 3 可以看出, 本文提出的最短路径近似算法相较于 TEDI 算法和 Takuya 准确算法的时间效率有很大提升, 而相较于 Takuya 混合近似算法提升不明显, 但由表 4 可见, 近似算法虽不能达到绝对精确度, 显然相较于 Takuya 混合近似算法准确性却又很大的提升.

综合实验结果可以看出, 该算法在复杂网络上能够大幅度地降低计算复杂性的同时, 还保持了较高的近似准确性.

5 结论与展望

由以上的理论基础和实验验证可知, 本文在保持

较高精度情况下得到了一个效率较高的算法. 我们是在无向无权图的情况下提出的该算法. 当然, 本文也可以扩展到有向图. 推广到有向图需要对每个区域节点计算两个最短路径——源点到中心顶点的距离,和中心顶点到源点的距离. 但对于加权图还需要进一步研究.

参考文献

1 Dijkstra EW. A note on two problems in connexion with graphs. Numerische Mathematik, 1959,1(1): 269−271.

2 Rattigan MJ, Maier M, Jensen D. Using structure indices for efficient approximation of network properties. Proc. of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM. 2006. 357−366.

3 Tretyakov K, Armas-Cervantes A, García-Bañuelos L, et al. Fast fully dynamic landmark-based estimation of shortest path distances in very large graphs. Proc. of the 20th ACM International Conference on Information and Knowledge Management. ACM. 2011. 1785−1794.

4 Akiba T, Sommer C, Kawarabayashi K. Shortest-path queries for complex networks: Exploiting low tree-width outside the core. Proc. of the 15th International Conference on Extending Database Technology. ACM. 2012. 144−155.

5 Wei F. TEDI: Efficient shortest path query answering on graphs. Proc. of the 2010 ACM SIGMOD International Conference on Management of data. ACM. 2010. 99−110.

6 Amir E. Approximation algorithms for treewidth. Algorithmica, 2010, 56(4): 448−479.

7 高文宇,李绍华.图的树分解及其算法应用研究进展.计算机科学,2012,39(3):14−18.

8 汪小帆,李翔,陈关荣.复杂网络理论及其应用.北京:清华大学出版社. 2006, 4.

9 Barabási AL, Albert R. Emergence of scaling in random networks. Science, 1999, 286(5439): 509−512.

10 汪小帆,李翔,陈关荣.网络科学导论.北京:高等教育出版社,2012

11 http://cdg.columbia.edu/cdg/datasets.

12 http://www.cs.cornell.edu/projects/kddcup/index.html.

13 http://dip.doe-mbi.ucla.edu/dip/Main.cgi.