移动目标单源最短路径树更新的近似算法*)

董 彬 李全龙 徐晓飞 宿 陆
(哈尔滨工业大学计算机科学与工程系 哈尔滨 150001)

摘 要 提出一种更新移动目标最短路径树的近似算法来避免重新生成整棵路径树。算法使用了局部图的思想,使每次迭代更新尽量少的节点来减少代价。实验证明算法具有良好的效率、近似度和可伸缩性。分析了如何调整算法,以便在近似度和效率之间实现平衡。

关键词 移动目标,单源最短路径树,近似算法,局部图

Approximate Algorithms for Updating Single-Source Shortest Path Tree of Moving Target

DONG Bin LI Quan-Long XU Xiao-Fei SU Lu
(Dept. of Computer Science&Engineering, Harbin Institute of Technology, Harbin 150001)

Abstract This paper presents an approximate algorithm for updating the shortest path tree of moving target to avoid re-generate the whole tree. Based on the concept of Local map, the algorithm tries to update as few nodes as possible to reduce cost. The experimental results demonstrate that the new algorithm deserves good efficiency, approximation and scalabilities. Furthermore, we show how to reach the trade-off between efficiency and approximation.

Keywords Moving target, Single-source shortest path tree, Approximate algorithms, Local map

在平面有向图上构造一棵以某个目标为根的最短路径树是一个典型问题,并且有着广泛的应用背景(如移动通信和军事应用等)。而在求解移动的追踪者和目标之间的最短路径问题中,如何构造移动目标的最短路径树是一个更为复杂的问题。当目标移动到一个新的位置时,需要构造一棵以目标新位置所在节点为根的最短路径树,其计算复杂度很高。另外,目标的每一步行动并不是总能被发现,有可能在目标从当前节点经过几条边后才被发现。这样,新位置所在节点不一定和上一次被发现位置所在节点相邻。

构造新的最短路径树的最直接解决办法是利用图的最短路径生成树算法,对整个图进行重新计算。图的最短路径树问题是一个在计算机科学领域的基本问题,目前对于边代价为正实数的一般图 $G=(V,E)$,最快算法是用 Fibonacci heaps 实现的 Dijkstra 算法[1],其运行时间复杂度为 $O(m+n\log n)$,其中 $n=|V|$,$m=|E|$。而对于非负边代价的平面图,文[2]基于图分解理论得到了 $O(n)$时间的算法来解决最短路径树问题。

Dijkstra 算法不能利用之前计算的最短路径树结果。对于移动目标来说,目标每次移动都要重新计算最短路径树,代价非常大,而且是不必要的,因为每次移动前后最短路径树只有部分发生了变化。如果能够通过更新前一次计算的结果来计算新的最短路径树,就能够避免对全部图进行重新计算。

在目标移动到相邻节点时,Stefan Edelkamp 提出了一种动态 Dijkstra 算法来更新最短路径树,解决目标追踪问题,最坏情况下的时间复杂度为 $O(n\log n+m)$[3]。但是对于本文提出的问题,当目标在不相邻节点被发现时,该算法不能求解。

S. Nguyen 等人[4]给出了解决此问题的一个时间复杂度为 $O(n^2)$的对偶算法,但是由于时间复杂度过大,很难实际应用。

本文基于局部图[5]的思想提出了一个更新移动目标的最短路径树的近似算法。在一般情况下,算法具有很好的性能和近似程度。

1 基于局部图的移动目标最短路径树更新算法

首先,我们让 u 和 v 分别为第 i 次移动前后目标的位置,第 $i-1$ 次迭代后,我们有以 u 为根节点的最短路径树 T_{i-1},以及以 v 为根节点的子树 T_v,经过第 i 次迭代得到最短路径树 T_i,如图 1 所示。

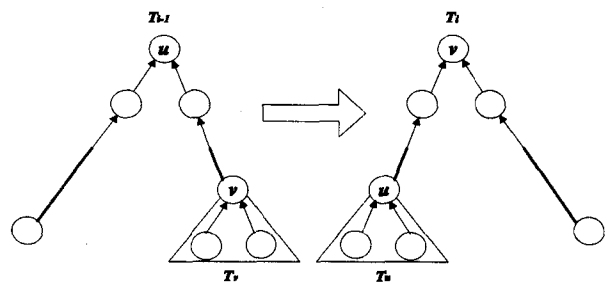

图 1 最短路径树 T_{i-1} 经过第 i 次迭代后得到 T_i

F. Chimura 和 M. Tokoro 在解决追踪移动目标的问题中提出了抽象图的概念[4],基本思想是把图分解成若干个局部图,而把每个局部图看成一个节点,就组成了一张抽象图。随着目标移动,在大部分情况下只需要更新其所在的局部图就可以了,从而减少了更新代价。在解决本文问题的过程中,我们也利用了更新局部图的思想来降低更新代价。

令 $v.parent$ 表示节点的先辈节点,$v.dist$ 表示到根节点的代价上界,$v.oldDist$ 保存更新前的 $v.dist$。定义局部图 $G'=(V',E')$,其中 $V'=\{x|x\in V, x.dist\leqslant range\}$,$E'=\{(x,y)|x,y\in V'\}$,如图 2 所示。

*)国家 863 计划(No. 2002AA413310, No. 2003AA4Z2170, No. 2003A413021)。董 彬 硕士研究生。

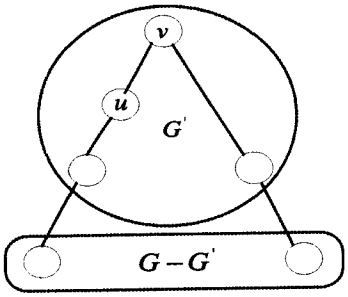

图 2　图划分为局部图 G′和非局部图 G−G′

一般情况下，由于每次 u 和 v 之间的最短路径代价相对于整个图来说相当小，我们可以认为更新后的最短路径树只是在包含 u 和 v 的局部图 G' 内有较大变化，而 $G-G'$ 内的子树没有明显改变。为求近似解，$G-C'$ 内的子树改变可以忽略。

这里 *range* 是局部图中节点到 u 最短路径代价的上界，$range=v.dist\times r$，$r\geqslant 1$ 为常数。其中 r 称之为范围比例因子。通过调节 r 的大小，能够控制算法在效率和精确度之间的平衡。如果 r 越大，算法就越精确，同时效率降低，反之效率提高，而算法精确度下降。

算法开始前，$v.dist$ 所保存的是 u 和 v 之间的最短路径代价。由于 $r\geqslant 1$，因此 $range\geqslant v.dist$，得到 $v\in V'$。

算法分为三步。

(1)初始化

算法的初始化需要完成：

- 局部图的划分
- 更新局部图内节点的 *dist* 值

更新 *dist* 值的目的在于尽量降低最短路径代价的上界，使计算局部以 v 根的最短路径树时尽量少的更新节点路径。我们就用已知到 v 最短路径代价给局部图的节点初始化。

```
procedure initialize
    Local←∅
    range=v.dist×r
    Divide(u)
```

Divide 过程需要找到所有与 u 的路径代价不大于 *range* 的节点 z，并根据上一次迭代已知的路径初始化 z 与 v 之间的最短路径代价上界。代价上界的初始化分为两种情况：

①$z\in T_v$，则 z 与 v 之间的最短路径已知，即 T_v 上 z 到 v 的路径。

②$z\notin T_v$，则上一次迭代已知的 z 与 v 之间的最短路径是 T 上 z 经 u 到达 v 的路径。

这个过程从节点 u 开始深度优先搜索，其中 $\Gamma(z)$ 是 z 的子节点集合。

```
Divide(z)
    if(z.dist≤range)
        Local←Local∪{z}
        z.oldDist=z.dist
        if(z∈T_v)
            z.dist←z.dist-v.dist//T_v 上 z 与 v 之间
            的最短路径代价
        else
            z.dist←z.dist+v.dist //T 上 z 经 u 到达
            v 的最短路径代价
    ∀z'∈Γ(z)−Local //遍历不属于 Local 的 z 的子
        节点
        Divide(z')
```

(2)计算局部图的最短路径树

这个过程使用的是 Fibonacci heaps 实现的 Dijkstra 算法[3]，其中 Q 是用 Fibonacci heaps 实现的优先权队列。

```
Procedure localSPT
Q←Local
While(Q≠∅)
z←ExtractMin(Q)
∀z'∈{x|(x,z)∈E'}
    If(z'.dist>z.dist+w(z,z'))
        z'.dist←z.dist+w(z,z')
        z'.parent←z
        DecreaseKey(Q,z',z'.dist)
```

(3)最短路径树的传播

我们得到了一棵 G' 上的以 v 为根的最短路径树。为了得到 G 的最短路径树，需要对 $G-G'$ 中的节点进行更新。

```
procedure Broadcast
    ∀z∈V−Local
        Update(z)
```

我们认为新的最短路径树在 $G-G'$ 内变化足够小，以致于 $G-G'$ 内节点的先辈关系改变引起的代价变化可以忽略。*Update* 方法用于更新节点的 *dist* 值。

```
procedure Update(z)
    if(z∈Local)
        return z.dist−z.oldDist
    else
        z.dist←z.dist+Update(z.parent)
        Local←Local∪{z}
        retrun dist
```

这个递归过程在 z 沿着双亲节点的指针找到第一个 *Local* 集合的节点时，计算该节点到根节点的最短路径代价的改变，然后更新其所有后辈节点的 *dist*。

2　算法分析

下面我们来分析此算法的时间复杂性。

在 initialize 过程中，需要把图划分为 G' 和 $G-G'$，这个过程使用树的先深搜索，找到所有与 u 的路径代价不大于 range 的节点，共需要 $|V'|$ 次划分，每次划分所花费的时间是常数，所以这个过程需要 $O(|V'|)$。

在 localSPT 过程中，V−Loval 通过用 Fibonacci heaps 实现的优先权队列，我们得到 $O(|V'|\log|V'|+|E'|)$ 的时间复杂度。

在 Braodcast 过程中，V−Loval 中的每一个节点都被遍历一次。每次遍历到某个节点时，需要更新其到根节点的路径代价，这个过程需要常数时间，所以 Braodcast 过程需要 $O(|V|-|V'|)$。

我们得到算法的时间复杂度为 $O(|V'|)+O(|V'|\log|V'|+|E'|)+O(|V|-|V'|)=O(|V|+|V'|\log|V'|+|E'|)$。当 $|V'|<<|V|$ 时，$O(|V|+|V'|\log|V'|+|E'|\approx O(|V|)$。

在最坏情况下，$range\geqslant\max(\{z.dist|z\in V\})$，则 $V'=V$，$O(|V|+|V'|\log|V'|+|E'|)=O(|V|\log|V|+|E|)$。

3 实验验证

实验中我们通过比较 Fibonacci heaps 实现的 Dijkstra 算法重新计算最短路径树，来分析算法的效率和准确度。实验数据采用随机图[6]，每次迭代目标经过的边的个数不超过 8，边代价取 1～200 的随机数。当 $i=0$ 时，我们用 Dijkstra 算法生成初始的最短路径树，$i=1$ 时开始用更新算法进行迭代。

为了描述结果的近似程度，我们定义近似度 $\rho=\sum_{z\in T} z.dist/\sum_{z\in T'} z.dist$，其中 T 是通过更新算法生成的最短路径树，T' 是通过 Dijkstra 算法生成的最短路径树。由于 T 是近似解，因此 $\rho\geqslant 1$。

为了描述不同条件下，整个迭代过程的运行时间和近似度，我们定义平均运行时间 $t=\sum t_i$（t_i 是第 i 次迭代的运行时间），及平均近似度 $p=\sum p_i$，p_i 是第 i 次迭代的近似度。通过计算平均运行时间和平均近似度来描述整个迭代过程。

下面我们通过实验从 3 个方面来分析本算法的运行效率和近似度。

3.1 目标移动过程中运行时间的比较和近似度的变化

我们采用 300×300 的随机图来进行实验，范围比例因子 r=5,7,9。

如图 3 所示，从运行时间上来看，更新算法由于避免了全部图的重新计算，相对于 Dijkstra 算法得到了较高的效率，运行时间随选取 r 值的不同而变化，较小的 r 值对应更高的效率。

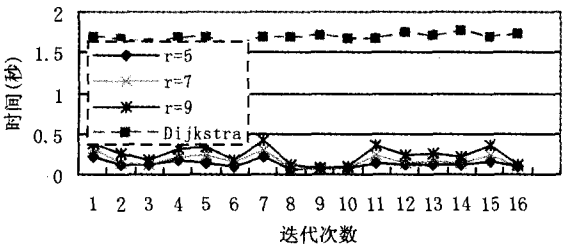

图 3 运行时间的比较

近似度随着迭代次数的增加的变化如图 4，更新算法得到了接近于 1 的较好的近似度。由于较大的 r 值使更新节点数目增加，得到的结果会更近似于精确解，近似度更接近 1。

由于每次迭代都是通过上一次迭代得到的近似解来更新计算，导致误差逐步累加，因此近似度随着迭代次数增加而上升。解决这个问题的办法是迭代过程中记录目标移动的总距离，在目标移动的总距离超过了某一个门限值时，对其重新计算，从而使近似度保持在某个范围之内。

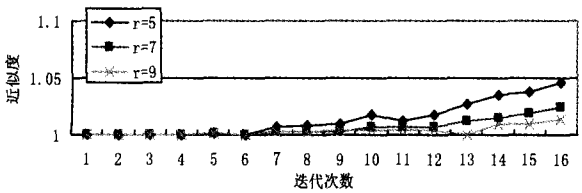

图 4 不同 r 值的近似度变化曲线

3.2 不同规模节点数的图上平均运行时间和平均近似度的变化

不同规模节点数的图上平均运行时间如图 5。如我们所知，Dijkstra 算法运行时间随着节点数目增加而显著增加。而更新算法的运行时间随着节点数目增加没有显著变化。

不同规模节点数的图上平均近似度如图 6。表明随着节点数的增加，结果近似度并没有明显变化，始终接近 1。

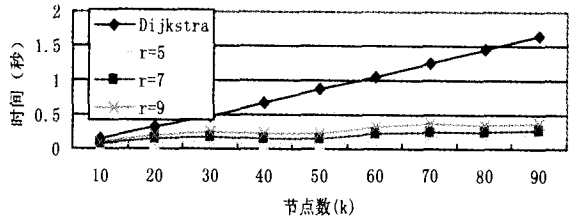

图 5 不同节点数的运行时间

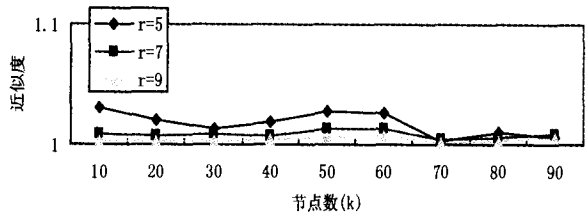

图 6 不同节点数的近似度

通过计算不同节点数目的图上平均运行时间和平均近似度，表明本文所给出的移动目标最短路径树更新算法具有较好的可伸缩性，能够适应不同规模图。

结论 对于以移动目标为根的最短路径树来说，目标每次移动都要对其重新生成的代价非常之大，利用已有的计算结果来计算是一条有效的途径。为了解决计算移动目标的最短路径树问题，本文提出了一个有效生成移动目标最短路径树近似算法，算法中我们使用了局部图的思想，使每次迭代更新尽量少的节点，从而使算法具有较好的效率和近似度。通过实验证明了算法的效率、近似程度和可伸缩性。

本算法的不足是随着迭代次数的增加，近似度呈上升趋势。可以通过对目标移动的总代价设置门限值，当总代价超过门限值时，对整个图重新计算。未来要解决的问题是找到一种估计当前迭代近似度的方法，以便能根据近似度判断是否需要重新计算，从而使近似度控制在某个范围之内。

参 考 文 献

1 Fredman M L, Tarjan R E. Fibonacci heaps and their use in improved network optimization algorithms[J]. Journal of the ACM, 1987,34:596～615

2 Klein P N, Rao S, Rauch M, et al. Faster shortest-path algorithms for planar graphs[A]. In: Proceedings of the ACM Symposium on Theory of Computing[C], Montreal, 1994. 27～37

3 Edelkamp S. Updating Shortest Paths[A]. In: European Conference on Artificial Intelligence (ECAI)[C], Brighton, England, 1998. 655～659

4 Nguyen S, Pallottino S, Scutella M G. A new dual algorithm for shortest path reoptimization. In: Gendreau M, Marcotte P, eds. Transportation and Nework Analysis - Current Trends, Kluwer, 2002. 259～265

5 Sasaki T, Chimura F, Tokoro M. The trailblazer search with a hierarchical abstract map. In: Mellish C S, ed. Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence[C], 1995. 259～265

6 Frigioni D, Ioffreda M, Nanni U, et al. Experimental Analysis of Dynamic Algorithms for the Single-Source Shortest-Path Problem [J]. ACM Journal of Experimental Algorithms, 1998,3(5)