一种基于 Dijkstra 并行线程算法的研究与实现

李　平，李永树

（西南交通大学 地球科学与环境工程学院，四川 成都 610031）

摘　要：针对传统 Dijkstra 算法运行效率的问题，提出了一种基于传统 Dijkstra 并行线程的算法，该算法动态地将交通网络进行子网分割。通过实验测试了不同网络节点数量和弧段数量下传统 Dijkstra 算法和本文算法运行时间，实验结果表明本文算法能够缩减网络节点搜索空间，降低算法的时间复杂度，提高算法的运行效率。

关键词：Dijkstra 算法；GIS；多线程；子网；时间复杂度；运行效率

中图分类号：P208　　**文献标识码：**B　　**文章编号：**1672－5867(2014)09－0050－04

Research and Implementation of One Parallel Thread Algorithm Based on Dijkstra

LI Ping, LI Yong－shu

(Earth Science and Environmental Engineering Academy of Southwest Jiaotong University, Chengdu 610031, China)

Abstract: Aiming at the problem of operating efficiency of traditional Dijkstra algorithm, the paper comes up with one algorithm of parallel thread based on traditional Dijkstra. The algorithm divises traffic network into subnets dynamically. Through experiment, the operating time of traditional Dijkstra algorithm is tested under different number of network junctions and arcs. The experiment has demonstrated that the algorithm can lower searching space of network junctions, reduce Dijkstra time complexity and improve Dijkstra operating efficiency.

Key words: Dijkstra algorithm; GIS; multithreading; subnet; time complexity; operating efficiency

0　引　言

最短路径分析是 GIS 空间分析中最重要的分析应用之一，其在医疗救护、火灾救援等事故以及人们出行导航方面已经成为不可或缺的分析应用[1]。随着城市交通网络的日趋复杂以及用户体验的日益提高，最短路径分析需要在尽可能短的时间分析出结果，加之计算机软硬件日新月异的进步，使得并行处理分析成为可能。

目前，对于最短路径 Dijkstra 算法研究主要有两个大的方向：一个是对于经典串行算法 Dijkstra 的修改、约束和改进，以达到改进经典算法的空间复杂度和时间复杂度，提高算法的运行效率的目的；另一个是修改 Dijkstra 算法为并行算法，合理利用计算机资源，提高算法的运行效率，较普遍的做法是，先根据并行数进行网络节点的平均分配，然后进行分析处理，但也不是并行数越多，效率越高[2]。

经典的 Dijkstra 算法属于串行算法，该算法时间复杂度为 $O(n^2)$[3]，随着网络节点数增加，算法需要的时间也会成倍增加[4]。考虑到没有充分利用现有的计算机资源，仍有较大的提升空间。本文提出一种新的并行算法思路，在经典 Dijkstra 算法的基础上，采用双线程分别从起点和终点同时按照 Dijkstra 算法最短路径的搜索分析，分析过程中采用节点标记法动态地将交通网络进行子网分割，最终将两个子网的搜索结果进行拼接汇总，获取最短路径。

1　最短路径算法简述

传统 Dijkstra 算法思想，被认为是最完备的算法，得出的结果最短，其特点是以起始点为中心，弧段权重为代价，向外层层扩展，直至扩展到终点为止[5]。

设 $G=(V,E)$ 是一个赋权有向图，即对于图中的每一条边 $e=(v_i,v_j)$ 都赋予了一个权值 w_{ij}，$D(v_j)$ 表示点 v_1 和 v_j 之间的最短距离。在图 G 中指定一个顶点 v_1，确定为起点。

收稿日期：2014－01－22

基金项目：高等学校博士学科点专项科研基金(20100184110019)资助

作者简介：李　平(1986－)，男，四川绵阳人，地图学与地理信息系统专业硕士研究生，主要研究方向为地理信息系统及其应用。

1)开始,先给 v_1 标上红色标号,且 $D(v_2)=0$,其余各点 $D(v_j)=+\infty(j\neq 1)$,红点是不可改变 D 值的点,它所具有的 D 值为该点至起始点的最短距离。

2) 如果刚刚得到的红色标号点是 v_i,将从该点进行扩展蓝点集,那么,对于所有这样的点$\{v_j \mid (v_i,v_j)\in E$,而且 v_j 的标号不是红色标号$\}$,将其 $D(v_j)$ 修改为: $\min[D(v_j),D(v_i)+w_{ij}]$,并将 v_j 表上蓝色标号,蓝点是具有蓝色标号的顶点中临时最短距离的点。

3) 从所有标有蓝色标号的顶点中求得 $D(v_i)$ 最小的顶点 v_i,并将其标上红色的标号,转向第 2) 步,直到所有的标有红色标号的顶点全部标成了蓝色标号的顶点,则停止[6]。

从传统 Dijkstra 算法思想可以看出,其采用的图遍历是以顶点数为基数的二次循环[7],依次遍历网络的所有节点,从而得出最短路径。

2　并行线程算法

传统的 Dijkstra 算法是串行算法,不具备并发性,在面对庞大复杂的交通网络的时候,就会显得效率低下,此时,算法的并发性就尤为重要。

2.1　设计思路

参考 A,B 相遇的数学问题,如果存在甲、乙两地,A 在甲地,B 在乙地,A 要遇到 B,在不考虑速度的情况下,假设 A 从甲地到达乙地的时间为 T(如图 1 所示),由 A 一个人走完全程,这就是串行思路; 但如果 A 从甲地出发的同时,B 从乙地出发,这样 A 遇到 B 的时间就会大大缩短,理论上会是 $T/2$, "效率"就提高了。如图 2 所示,这就是本文算法的研究思路: 分派两个线程 A 和 B,分别从甲地和乙地同时相向而行,然后进行"相遇"处理。

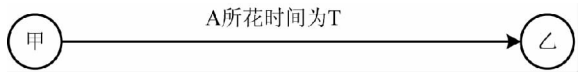

图 1　串行思路

Fig. 1　Serialization

图 2　并行思路

Fig. 2　Parallel

如图 3 所示的无向图 G 中,起点为甲,终点为乙,根据串行 Dijkstra 算法,容易得出,甲到乙的最短路径为: 1 ~ 3 ~ 4 ~ 5。按照本文的并行线程算法的研究思路,线程 A 从甲地出发,线程 B 从乙地出发,开始分析。

假设经过分析处理后,线程 A 搜索处理的节点集为 $\{1,2,3\}$,搜索到的路径集合为$\{1\sim 2,1\sim 3\}$,线程 B 搜索处理的节点集为$\{4,5,6\}$,搜索到的路径集合为$\{5\sim 6,5\sim 4\}$,此时,交通网络就被动态地分割为两个子网,然后根据节点与弧段的拓扑关系进行路径集合的拼接("相遇"问题处理),可以得出拼接路径集合为$\{1\sim 2\sim 4\sim 5,1\sim 3\sim 4\sim 5,1\sim 3\sim 6\sim 5\}$,比较各个路径的累积权重可以得出最短路径为: 1 ~ 3 ~ 4 ~ 5。

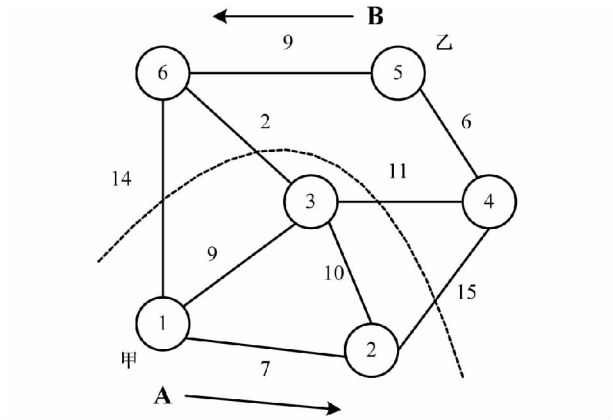

图 3　无向图 G

Fig. 3　Undirected graph G

由此,可以看出本文的并行线程思路在得出最短路径的同时,可以将最短路径搜索的节点数目大大减少,理论上减少了 1/2,同时,将网络动态地划分成了两个子网,达到提高串行 Dijkstra 算法的运行效率的目的,最后,需要花费一定的时间进行后续的路径集合拼接处理。

2.2　数据结构设计

GIS 中网络图是用弧段和节点表示,通过拓扑构建可以得到图中弧段与弧段、弧段与节点、节点与节点之间的邻接关系。图的存储方式很多,常用的有邻接矩阵、邻接表、邻接多重表、十字连表等[8]。为了节省存储空间,避免算法的效率受到影响,本文采用邻接表存储网络图[9]。弧段与节点的拓扑存储邻接关系见表 1、表 2。

表 1　节点与弧段的邻接关系

Tab. 1　Adjacent relationship between knot and curve segment

VertexId	ArcId
V1	E1,E2,E3…
⋮	⋮
Vn	Ei,Ej,Ek…

表 2　弧段与节点的邻接关系

Tab. 2　Adjacent relationship between curve segment and knot

ArcId	VertexId
E1	V1,V2
⋮	⋮
En	Vi,Vj

由拓扑关系可知,一个节点可能会与多条弧段相邻接,一条弧段只可能与首尾两个节点相邻接。

节点存储结构

```
class Junction{
int JunctionId; //节点 Id
Point Junction; //节点对象
List < Edge >  AdjacentEdges; //节点邻接弧段对象集合
```

bool beProcess; //当节点所有的邻接弧段均被分析后进行标记

bool beCapture; //当节点被 A 或 B 线程占领扩展时进行标记,防止 A,B 线程重复访问相同节点。

}

弧段的存储结构

class Edge{

intEdgeId; //弧段 Id

List < int > AdjacentPoints; //弧段的邻接节点 Id(弧段起始点)集合

double weight; //该弧段的权重值

}

路径的存储结构

class Path{

int JunctionId; //路径的终点 Id

double SumWeight; //路径的累积权重

List < int > passedJunctionsId; //路径所经过的节点 Id 集合

}

2.3 多线程问题[10]

本算法的实现平台采用 C#. net4.0 进行实现。

1) 多线程同步

由于算法中分为 *A*,*B* 线程进行并行处理,其中会涉及到对公共资源的访问与处理,此时必须进行两个分析线程的同步处理,否则如果 *A* 正在访问资源,执行读操作,*B* 却在修改资源,执行写操作,资源不同步,就会出现不可预知的致命错误。

采用 C#中的 lock 关键字可以对于公共资源的访问进行锁定,实现线程间的同步。核心代码如下:

```
class Example{
object threadLock = new object( ) ; //锁标识符
void Example( ) {
lock( threadLock) {
//对公共资源进行访问和处理的代码
}
}
}
```

2) 多线程搜索

算法中,*A*,*B* 线程需要不断地查询搜索相应的节点对象以及扩展的路径对象,在庞大的数据面前,查询搜索会消耗掉大部分的时间。采用 C#中 Parallel 多线程遍历方法,提高搜索效率。核心代码如下:

```
Parallel. For( fromIndex, toIndex, ( Index, loopState)
= >{
if ( //条件) {
. . . . . //处理逻辑
loopState. Break( ) ; //退出多线程搜索循环
}
} )
```

2.4 算法描述

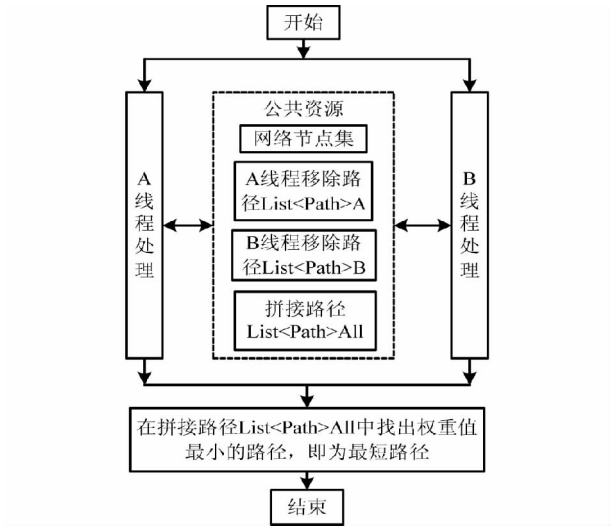

图 4 算法总流程图

Fig. 4 Overall algorithm

如图 4 所示的算法总流程图中,程序一开始,就分别启动 *A*,*B* 两个线程,按照串行 Dijkstra 算法思路进行局部最短路径搜索和扩展,其中 *A* 线程由起点开始分析,*B* 线程由终点逆向进行分析。*A* 线程需要更新维护 *A* 线程的移除路径表 List < path > *A*,*B* 线程需要更新维护 *B* 线程的移除路径表 List < path > *B*,拼接路径表 List < path > All 与网络节点集则需要由 *A*,*B* 线程共同维护更新,因此,在对公共资源进行读写操作时,*A*,*B* 线程必须同步。最终在拼接路径集合(全路径) List < path > All 中获取累积权重最小的路径,即为最短路径。

A,*B* 线程的处理逻辑大致一样,现以 *A* 线程处理逻辑为例,*A* 线程流程图如图 5 所示。

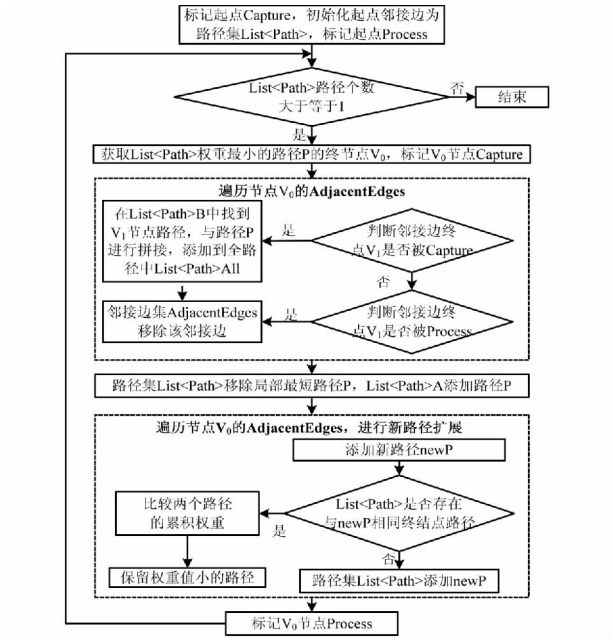

图 5 A 线程处理流程图

Fig. 5 Processing procedures of thread A

流程图中,节点的 Process 属性用于标记线程内节点是否被处理分析,即传统 Dijkstra 算法的蓝、红点状态标记。节点的 Capture 属性用于线程间的节点是否被访问占领,即本文特有的排它状态标记,每个线程需要在访问某个节点的第一时间标记节点的 Capture 状态,其他线程遇到被标记为 Capture 的节点时,不予扩展分析,只进行线程间的路径拼接分析。这样做的目的是保持 A,B 线程处理分析的节点集互相没有交集,使得程序逻辑清晰,方便路径拼接分析,给出正确的分析结果。

3　实验验证

3.1　与传统 Dijkstra 算法比较

本文实验基于 ArcGIS 平台,实验数据来自于 ArcGIS Desktop 10.0 Tutorial Data 中 Paris 城市交通网络数据,通过对该交通网络的不同程度的删减处理,测试了在不同网络节点数量和弧段数量下,传统串行 Dijkstra 算法和本文算法运行分析时间对比见表 3。

表 3　算法性能统计

Tab. 3　Functional statistics of algorithm

节点数	弧段数	串行 Dijkstra 算法时间	并行 Dijkstra 算法时间
582	901	0.078s	0.198s
1 798	2 846	0.495s	0.691s
5 810	9 068	4.674s	3.278s
12 072	18 624	18.163s	8.784s

从实验分析结果数据可以看出:

1) 当交通网络的节点数和弧段数较少时,即交通网络相对简单,如第一、二组数据,串行的 Dijkstra 算法的算法时间少于本文并行 Dijkstra 算法的算法时间,原因在于并行线程在对公共资源进行处理时需要进行同步处理以及后续路径的拼接和权重比较,消耗了部分时间。

2) 当交通网络的节点数和弧段数较多时,即交通网络相对复杂,如第三、四组数据,本文的并行 Dijkstra 算法的算法时间少于串行的 Dijkstra 算法,并且随着网络节点数和弧段数的继续增加,本文并行算法的时间相对于串行的算法时间将大大缩短。

通过以上分析可发现,对于中大规模复杂的交通网络,本文的并行 Dijkstra 算法就单个线程而言,缩小了网络节点的搜索空间,降低算法的时间复杂度,能够很大程度上提高传统串行 Dijkstra 算法的分析效率。

3.2　与常规并行 Dijkstra 算法比较

常规并行 Dijkstra 算法,不管是采用多线程技术,或是多计算机技术,先要确定并行数目,然后进行网络节点的纯数学意义上的平均分配,划分成多个子网,接着进行分析处理。此时就有一个问题出现: 这种分配是事先分配好的,属于静态的,节点数的平均,并不意味着与节点相关连的弧段数也是平均的,有些计算机或者线程处理快,有些则处理慢,处理快的就闲置了,无法去帮助处理慢的,但最终需要等待所有网络子网都处理完毕,才会进行汇总分析,一定程度上造成了资源的浪费,影响了分析的效率。其次,多计算机或者多线程同时处理一个任务,其间的相互同步协调,数据通信必然频繁,效率也会大打折扣。

本文的并行 Dijkstra 算法,虽然仅仅扩展到两个线程,但易于线程间的协调控制和数据通信,两个线程在处理过程中根据节点的状态标记,将交通网络动态地进行两个子网的分割,一个线程处理慢了,另一个线程可以进行"帮忙",直到处理完毕,两个子网才划分结束。

不会造成资源的浪费,提高了分析的效率。

综上所述,常规并行算法支持多个线程(≥ 2) ,在线程数目上大于本文算法,在超大规模的交通网络中必然效率高于本文算法,但其对软硬件的要求也较高; 在中大规模的交通网络中,本文算法相对更有优势,简单明了,易于实现。

4　结束语

本文研究了传统 Dijkstra 串行算法的单线程的不足,并进行了基于传统 Dijkstra 并行线程的实现方法研究,利用计算机的多线程技术,将传统的串行算法改为两个线程首尾并行进行分析的算法,缩小了单个线程的节点搜索空间,减少了算法的时间复杂度。同时,也讨论了本文算法与现有并行算法的优势与区别,实验证明: 在中大规模复杂的交通网络中,能大幅度提高算法的运行效率。该算法沿用了经典的 Dijkstra 算法,将交通网络动态地分成了两个子网,同时进行分析,思路简单明了,易于实现。

参考文献:

[1] 古凌岚. GIS 最短路径分析中 Dijkstra 的算法的优化[J]. 计算机与数字工程,2006,34(12):53 - 55.

[2] 卢照,师军,于海蛟,等. 城市路网的最短路径并行求解[J]. 计算机技术与发展,2010,20(1):82 - 84.

[3] 卢照,师军. 并行最短路径搜索算法的设计与实现[J]. 计算机工程与应用,2010,46(3):69 - 70.

[4] 张渭军,王华. 城市道路最短路径的 Dijkstra 算法优化[J]. 长安大学学报: 自然科学版,2005,25(6):62 - 63.

[5] 刘刚,李永树,杨骏. 一种 Dijkstra 算法改进方法的研究与实现[J]. 测绘科学,2011,36(4):233 - 235.

[6] 司连法,王文静. 快速 Dijkstra 最短路径优化算法[J]. 测绘通报,2005(8):15 - 16.

[7] 周培德. 交通道路网络中任意两点之间最短路径的快速算法[J]. 计算机工程与科学,2002,24(2):35 - 37.

[8] 严蔚敏,吴伟民. 数据结构[M]. 北京: 清华大学出版社,1997.

[9] 王臣杰,毛海城,杨得志. 图的节点 - 弧段联合结构表示法及其在 GIS 最短路径选择中的应用[J]. 测绘学报,2000,29(1):47 - 51.

[10] Andrew Troelsen. C#与. NET4 高级程序设计[M]. 朱晔,肖逵,姚琪琳,等译. 北京: 人民邮电出版社,2011.

[编辑: 胡　雪]