基于并行计算的快速Dijkstra算法研究

叶颖诗,魏福义,蔡贤资
华南农业大学 数学与信息学院,广州 510000

摘 要:通过分析经典Dijkstra算法的思想和执行流程,对多标号的Dijkstra算法给出新证明,以此作为理论依据对Dijkstra算法进行了多标号的串行与并行优化。对于正则树,给出了经典Dijkstra算法、串行多标号Dijkstra算法和并行多标号Dijkstra算法的时间复杂度排序。针对优化算法的特点,设计出四种实验,采用运行时间和并行加速比作为优化指标,考核三种算法的效率。仿真实验表明:对顶点数大于6 000的稠密图和稀疏图(正则树),多标号并行算法优于串行算法,且优化效果明显;对于正则树,优化效果分别与深度、出度成正相关。

关键词: Dijkstra算法;并行计算;最短路径;正则树;时间复杂度;仿真实验
文献标志码:A **中图分类号:**TP312 doi:10.3778/j.issn.1002-8331.1903-0119

叶颖诗,魏福义,蔡贤资.基于并行计算的快速Dijkstra算法研究.计算机工程与应用,2020,56(6):58-65.
YE Yingshi, WEI Fuyi, CAI Xianzi. Research on fast Dijkstra algorithm based on parallel computing. Computer Engineering and Applications, 2020, 56(6):58-65.

Research on Fast Dijkstra Algorithm Based on Parallel Computing

YE Yingshi, WEI Fuyi, CAI Xianzi
College of Mathematics and Information, South China Agricultural University, Guangzhou 510000, China

Abstract: In order to make optimized to Dijkstra algorithm, this paper gives a proof of multi-label Dijkstra algorithm and uses it as a fundamentals to make the optimized improvement, which is the classical Dijkstra algorithm and the multi-label Dijkstra algorithm with serial and parallel way. Facing the regular graph, this paper calculates three algorithm time complexity. After analysis, this paper comes up with multi-label parallel computing for the Dijkstra optimized algorithm. This optimized algorithm designs four experiment to verify degree of optimization by running time and parallel acceleration ratio. The experiment result shows the multi-label parallel computing algorithm is more effective than the classical Dijkstra algorithm and the multi-label serial computing algorithm facing the dense graph which point number are more than 6 000 and sparse graph(regular tree). For the regular tree, it exists positive correlation between the effectiveness and regular tree's depth and out degree.

Key words: Dijkstra algorithm; parallel compute; shortest path; regular tree; time complexity; simulation experiment

1 引言

最短路径问题是一个经典的计算问题,它是计算一个节点到其他节点的最短路径,在很多领域具有广泛的应用,譬如物流线路优化,GPS导航、社交网络等。快速计算最短路径,满足其应用的实时性是这些应用的基本要求。常见的最短路径算法有Dijkstra算法、Floyd算法、A*算法等。

Dijkstra算法[1]是求解单源最短路径算法中的经典算法。对Dijkstra的优化引起了海内外学者的广泛关注。构造特殊的数据结构是常见的优化手法,如堆[2]和最短路径树[3]等。文献[4]提出了当临时性标号t相同时,将其同时标记为永久性标号p。随着多核时代的到来,并行计算是解决此问题的关键。文献[5]首次提出并行异步更新标号t的优化算法,其主要思想是使用多

基金项目:广东省联合培养研究生示范基地人才培养项目;华南农业大学2018质量工程项目;广东省教育厅特色创新类项目(No.2017KTSCX020)。
作者简介:叶颖诗(1994—),女,硕士研究生,研究领域为图论,最短路径;魏福义(1964—),通信作者,男,教授,研究领域为代数图论,E-mail:weifuyi@scau.edu.cn;蔡贤资(1973—),男,副教授,研究领域为数据挖掘,计算机应用技术。
收稿日期:2019-03-11 **修回日期:**2019-07-11 **文章编号:**1002-8331(2020)06-0058-08
CNKI **网络出版:**2019-07-24, http://kns.cnki.net/kcms/detail/11.2127.TP.20190724.1129.006.html

个队列在共享储存系统上实现更新。文献[6]将并行算法通过OpenMP引入到Dijkstra算法中;文献[7]将多线程运用于交通网络进行子网分割中;文献[8]通过Hadoop云平台下采用MapReduce并行编程提高算法效率。文献[9-13]均为前人对最短路径算法的并行优化。

本文将多标号的Dijkstra算法和并行计算有机结合,给出了文献[4]所提出的多标号Dijkstra算法新的证明;得到赋权正则树对于三种算法的时间复杂度排序;采用Java编程进行仿真实验,分析多标号串行算法与并行算法的优劣。实验结果表明,Dijkstra多标号并行算法能更好地提高计算最短路径的效率。

2 Dijkstra算法及其优化方向

本章给出了将Dijkstra算法优化为串行算法和并行算法的理论基础,给出了基于并行计算的Dijkstra算法。

2.1 Dijkstra算法思想

Dijkstra算法是由荷兰科学家Dijkstra于1959年首次提出[1],此算法被叙述为初始化、求标号p和修改标号t三步。它可以有效地解决单源最短路径问题。该算法的主要特点是以起点为中心点向外延展,直至延展到终点为止,在此过程中获得两点间的最短路径。

给定一个简单赋权图 $G(V,E,W)$,其中 V 为图 G 的顶点集,E 为图 G 的边集,W 为图 G 的边权集。对于任意 $v_i,v_j\in V$,若存在 $e_{ij}\in E$,则边 e_{ij} 的权 w_{ij} 为非负正实数;否则,$w_{ij}=\infty$。

首先给出描述Dijkstra算法的基本定义。

定义1 设 $l_i^{(r)*}$ 为顶点 v_1 到顶点 v_i 的最短路径的权值,如果顶点 v_i 在第 r 步获得了标号 $l_i^{(r)*}$,则称顶点 v_i 在Dijkstra算法的第 r 步获得了永久性标号p,其中 $r\geqslant 0$。

定义2 设集合 $P_r=\{v|l_i^{(r)*}\neq 0\}$,$P_r$ 称为永久性标号集合。其中任意 $v_i\in P_r$,v_i 获得永久性标号p;设集合 $T_r=V/P_r$,其中 T_r 中的元素为在第 r 步尚未获得永久性标号p的点集,T_r 称为临时性标号集合。

定义3 $l_i^{(r)}$ 为顶点 v_0 到顶点 v_i 的最短路径的上界,如果顶点在第 r 步获得 $l_i^{(r)}$,则称顶点 v_i 在Dijkstra算法的第 r 步获得了临时性标号t,其中 $r\geqslant 0$。当 v_0 与 v_i 通过集合 T_r 不可达或 $i=0$,则 $l_i^{(r)}=\infty$。

2.2 优化方向

文献[4]提出,当选取标号p时,面对多个顶点同时在第 r 步拥有最小的标号t时,应该同时拥有标号p。令这p个顶点同时拥有标号p,可以在一定程度上减少集合 T_r 历遍次数,缩短计算时间,优化算法计算效率。

定理1 Dijkstra标号法中,在同一步将多个拥有最小标号t的顶点标记上标号p,最短路的计算结果不变。

证明 计算 v_0 到图 G 的其他顶点的最短路径。

当 $r>0$,若存在 $v_i,v_j\in T_{r-1}$,使得

$$l_i^{(r-1)}=l_j^{(r-1)}=\min_{v_k\in T_{r-1}}(l_k^{(r-1)}),i\neq j$$

即 v_i,v_j 在第 r 步中同时拥有被p标记的机会。现在将顶点 v_i 标记为标号p。接着进行Dijkstra算法的修改标号t。

在此步不改变 v_j 的标号t值。若需修改,则 $e_{ij}\in E$。否则,由于 $l_i^{(r-1)}+w_{ij}<l_i^{(r-1)},w_{ij}=0$。但 $l_i^{(r-1)}=l_i^{(r-1)}$,所以在此步不可能修改 v_j 的标号t值,$l_i^{(r)}=l_i^{(r-1)}$。接下来Dijkstra算法找寻新的标号p。

假设在此 r 步内选取不到 v_j 标记为标号p,则存在 v_m,使得 $l_m^{(r)}=\min_{v_k\in T_r}(l_k^{(r)}),l_m^{(r)}<l_j^{(r)}$。

当 $l_m^{(r)}$ 在第 r 步被修改过,则

$$l_m^{(r)}=l_m^{(r-1)}+w_{im}<l_j^{(r)}=l_j^{(r-1)}$$

所以 $l_m^{(r-1)}<l_i^{(r-1)}=l_j^{(r-1)}$,此与选取 v_i 为标号p矛盾。

当 $l_m^{(r)}$ 在第 r 步被未修改过,$l_m^{(r)}=l_m^{(r-1)}$,但 $l_m^{(r-1)}<l_j^{(r-1)}$,矛盾。

综上,在第 r 步可以选到 v_j 作为标号p,选择的次序并不影响后面的操作。所以不影响计算结果。

运用此定理,可减少选取标号p的次数,从而缩短算法的运行时间,达到提高算法效率的目的。本文将此思想加入到优化算法中,称该算法为多标号的Dijkstra算法。

2.3 多标号的Dijkstra并行算法

分治法,就是"分而治之",将一个复杂的问题分为若干个相似的子问题,而子问题能用简单的方式直接求解。并行计算正是使用分治的思想,使用多个处理器来协同解决同类问题,分别对原问题的子问题同时求解,以增加算法效率。使用多线程并行计算的目的是充分使用CPU资源,使其在较短的时间内完成计算,进而提升整体处理性能。如图1所示,图中的 Δt 则为节省的时间。

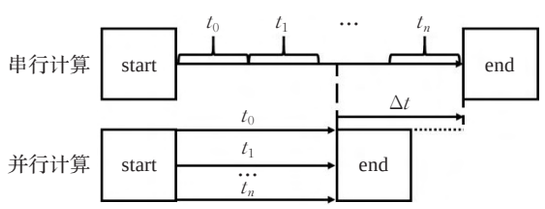
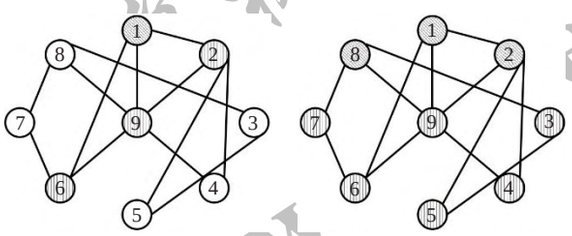


图1 并行计算效率提升示意图

随着研究问题规模的扩大,传统的串行计算技术难以满足对其计算能力的需求。文献[4]中的多标号Dijkstra算法将多个拥有最小 $l_i^{(r)}$ 的顶点同时标记上永久性标号p时,多标号的Dijkstra算法(3)中修改标号t变得复杂

了。由于此步仅修改获得永久性标号p顶点的邻点,导致多标号的Dijkstra需要修改的点数增加。

如图2所示,经典的Dijkstra算法将一个点(点1)选为永久性标号p,它的邻点(点2、点8、点9)将需要修改临时标号t;然而多标号的Dijkstra同时将三个点(点1、点2、点3)选为永久性标号,则它需要修改这些点所有邻点的临时性标号(点3、点4、点5、点6、点7、点9)。由此可得多标号的Dijkstra修改临时性标号的工作量被增加了。因此考虑将并行算法融入到多标号的Dijkstra算法之中。



(a)经典Dijkstra修改临时性标号t　(b)多标事情Dijkstra修改临时性标号t

图2　两种算法修改临时性标号工作量对比

当有多个线程同时运行并试图访问同一个公共资源时,会出现资源争夺的问题,从而影响程序执行的效率和正确性,这种现象被称为非线程安全。为了避免非线程安全,需要保证这些公共资源只有一个线程在使用它。在多标号的Dijkstra算法中,储存标号t的数组作为此并行算法的共享数据。在此根据所在实验平台的CPU的线程数,分配各个线程的工作量,让线程固定处理若干个顶点的标号t值,从而做到数据独立,达到线程安全。如图3所示,假设数组长度为8,线程数为4,则每个线程为其分配两个顶点进行工作。

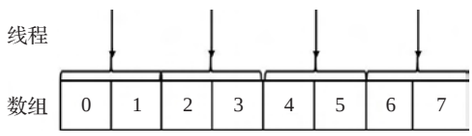


图3　线程分配示意图

假设计算 v_1 到其他点的最短路径,多标号Dijkstra并行算法流程图如图4所示。

3　三种算法的时间复杂度

算法的效率常用时间复杂度作为评判指标。针对赋权正则树,分析了经典Dijkstra算法,多标号的Dijkstra串行算法和多标号的Dijkstra并行算法的时间复杂度,并且给出其排序。

由于并行算法所节约的遍历次数与各实验平台处理器性能有关,计算其算法复杂度比较困难。本文试构造特殊图,在求标号p步骤中,可以使得多个顶点同时拥有永久性标号p。在计算时间复杂度时,算法优化程度能达到极限。

若图中任意两个顶点之间都有边相连,此图称为完全图,记为 B。若它的顶点数为 n,则边数为 C_n^2,即 $n(n-1)/2$。当图的边数远远少于完全图,这样的图称为稀疏图;反之,则称为稠密图。

无圈的连通图称为树,记为 T。给定一个树 $T(m,h;w_1,w_2,\cdots,w_h)$,在此树中任意非叶子节点的出度都相同,记为 m;从根节点到叶子节点依次经过的节点(含根、叶节点)形成树 T 的最长路径称为深度,记为 h;当根节点到两个节点的的路径长度相同,则这两个节点处于同一层,第 $r(r=1,2,\cdots,h)$ 层的节点的出度上的权数相同,记为 w_r。此种树 T 称为赋权正则树。赋权正则树 $T(3,2;1,2)$,见图5。

对于赋权正则树 $T(m,h;w_1,w_2,\cdots,w_h)$,设它的顶点数为 n,则 $n=\sum_{r=1}^{h}m^r+1$。

传统的Dijkstra算法每步修改临时性标号的工作量为1,最多 $n-1$ 步,因此总工作量为 $n-1$;而多标号的Dijkstra算法第 r 步修改临时性标号的工作量则为 m^r(r 为步数,$1\leqslant r\leqslant h$),总工作量是 $\sum_{r=1}^{h}m^r=n-1$,共 h

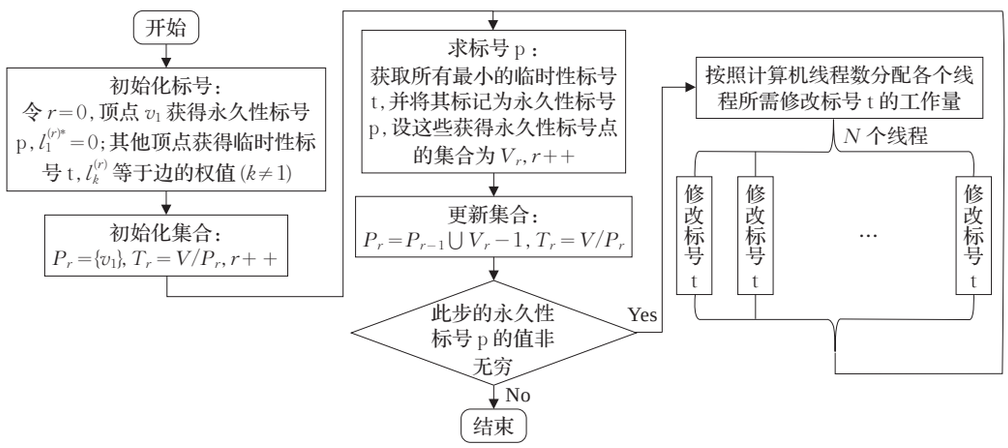


图4　多标号Dijkstra并行算法流程图

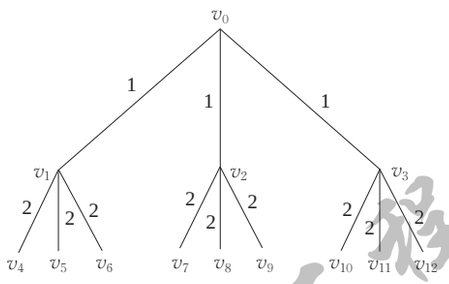


图5 正则树示意图

步。总工作量没有减少,但步骤少,时间缩短。设开设的线程数为 l。并行多标号的Dijkstra算法第 r 步修改临时性标号的工作量为 $\left\lceil \frac{m^r}{l} \right\rceil$,共 $\sum_{r=1}^{h}\left\lceil \frac{m^r}{l} \right\rceil$ 步,则总工作量不大于 $\sum_{r=1}^{h}\left\lceil \frac{m^r}{l} \right\rceil \leqslant \left\lceil \frac{n-1}{l} \right\rceil + h$。因此引入并行计算可减少修改临时性标号的工作量和总步数,达到提高算法效率的目的。

3.1 时间复杂度排序

下面针对赋权正则树 $T(m,h;w_1,w_2,\cdots,w_h)$ 的结构对时间复杂度进行分析。

定理2 对于赋权正则树 $T(m,h;w_1,w_2,\cdots,w_h)$,传统的Dijkstra算法,多标号的Dijkstra串行算法,多标号的Dijkstra并行算法时间复杂度依次降低。

证明

(1)传统的Dijkstra

使用线性搜索计算最大标号p的Dijkstra算法,其时间复杂度为 $O(n^2)$ [1]。文献[14]通过斐波那契堆储存标号p,其时间复杂度下降到 $O(n\lg n+m)$,其中 n 表示顶点数,m 表示边数。

(2)串行多标号的Dijkstra

串行算法外循环的时间复杂度为 $O(h)$,使用二分法选取标号p的时间复杂度为 $O(\lg n)$,修改标号t的时间复杂度为 $O(n)$,因此其总的时间复杂度为 $O(h(n+\lg n))$。

(3)并行多标号的Dijkstra

并行算法假设实验平台支持 l 条线程同时工作,在理想的状况下,修改标号t的时间复杂度从 $O(n)$ 下降到 $O(n/l)$。因此其总的时间复杂度为 $O(h(n/l+\lg n))$。

$$O(n^2)>O(h(n+\lg n))>O(h(n/l+\lg n))$$

定理成立。

3.2 准确性分析

时间复杂度计算是基于理想状态下的算法效率衡量,下面对此时间复杂度进行准确性分析。

针对并行多标号的Dijkstra算法,并行计算中线程的切换需要耗费一定的时间成本,在同一操作平台下,此时间成本记为 T。由定理2可知:理想状态下并行多标号的Dijkstra算法时间复杂度为 $O(h(n/l+\lg n))$。则现实状态与理想状态的时间复杂度比值为:

$$\frac{h\left(\frac{n}{l}+\lg n\right)+T}{h(n/l+\lg n)}$$

当 n 较小时,从理论上说,由于T的存在,此值大于1,优化提速与线程开销抵消,运用传统算法计算时间相对于两种优化,计算时间也较短,优化效果不明显。由于

$$\lim_{n\to\infty}\frac{h\left(\frac{n}{l}+\lg n\right)+T}{h(n/l+\lg n)}=1 \tag{1}$$

当 n 逐渐增大时,此值逐渐接近1,算法出现正优化,时间复杂度与算法速率呈正相关,说明此复杂度计算的准确性。

当固定图规模 n 时,非叶子节点 m 越大,深度则越小,则并行多标号的Dijkstra算法的时间复杂度越小,计算速率越高。

4 仿真实验

本文搭建了单机版的实验平台。实验平台的硬件环境为Intel® Core™,且有四个3.6 GHz的内核CPU,在硬件上支持八线程并行,机器上运行Windows 10专业版(x64)的操作系统,内存为8 GB,利用Java语言编程实现。

4.1 实验指标和目的

针对稠密图和特殊类型的非稠密图(正则树)两类图,共设计四种实验,采用运行时间和并行加速比两个指标,刻画Dijkstra算法,多标号的Dijkstra串行算法和多标号的Dijkstra并行算法的运行效率。设计实验显示对Dijkstra算法的优化效果。

本文用于衡量算法优化程度的指标为运行时间和并行加速比。

(1)运行时间

运行时间作为判断程序效率的常见指标,能较好地体现优化效果。

(2)并行加速比

并行加速比是用于衡量程序并行化的性能和效果的一个常见指标。其计算方法为串行计算程序和并行计算程序中的运行时间比率,公式如下:

$$S_p=T_1/T_p$$

其中 p 为CPU数量,T_1 为传统Dijkstra算法的执行时间,T_p 为当有 p 个处理器时,并行算法的执行时间,S_p 称为当有 p 个处理器时的加速比。当 $S_p=p$ 时,称此加速比为线性加速比,又称为理想加速比。所以,当并行加速比的数值越接近处理器数量,程序的并行优化程度越好。

设计下列实验,验证基于多标号的串行算法与并行算法对Dijkstra算法的优化效果。

4.2　仿真设计与数据

在数据结构中,常见的表示结构有两种:邻接矩阵和邻接表。

邻接矩阵是使用二维数组模拟线性代数中矩阵的结构。此种数据结构所使用的储存空间较大,但其使用较为便捷。

邻接表使用单链表数组数据结构实现。数组的下标为顶点的标号,每一个数组元素指向一个单链表。单链表中的元素则是顶点下标所指顶的邻点。链表中的元素数据结构如图6所示,每个元素代表其所在数组下标所表示的顶点与链表元素中顶点编号所示顶点有边相连,“边权值”代表该边的权值,“下一个元素地址”表示下一个链表元素以此形成链表的结构。

图6　单链表元素数据结构

将所有链表以数组的形式存放到一起,具体转换形式如图7。

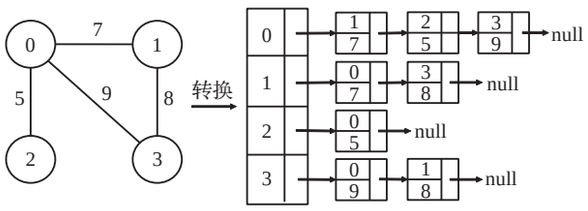


图7　图结构与邻接表的转换示意图

当图的顶点数较大时,节省内存空间选择邻接表表示无向图。设计两类数据图进行实验。

(1)权值满足泊松分布的随机稠密图

为验证一般图与本优化算法的贴合度,特设计一般随机图进行实验,图的权值符合泊松分布[15]。通过观察三种算法在此图中两种指标的状况,研究图的顶点规模与两种算法的优化性能。本实验将稠密图顶点数与边数的比例控制在1:5左右,即将此数据集的点边比控制0.2左右,令边的权值分布符合泊松分布,以此控制顶点数观察图规模与优化性能之间的关系点边规模如表1所示。

表1　泊松分布随机图边点规模

编号	顶点数	边数	编号	顶点数	边数
1	2 000	11 478	6	12 000	69 689
2	4 000	22 846	7	14 000	79 857
3	6 000	34 279	8	16 000	91 247
4	8 000	45 797	9	18 000	102 687
5	10 000	57 969	10	20 000	114 789

(2)非稠密图-正则树

多标号的Dijkstra串行算法对于Dijkstra算法的优化着重于在每一步选取永久性标号p,当每一步选取的节点越多,此算法的优化效率应越明显。

计算最短路径一般面向稀疏图。本实验通过正则树探求两种优化算法所适应的图的一般情况。研究树的非叶子节点的出度、深度与本算法优化之间的关系,设计出三种类型的正则树进行实验。

① 固定深度,以非叶子节点的出度为变量。

为考察非叶子节点出度与优化效果之间的关系,此实验固定深度以非叶子节点的出度作为变量进行实验。固定图的深度为4,令任意非叶子节点的出度逐渐增大,出度与节点数量的关系如表2所示。

表2　深度为4的出度与节点数关系

编号	出度	节点数	编号	出度	节点数
1	16	4 369	7	22	11 155
2	17	5 220	8	23	12 720
3	18	6 175	9	24	14 425
4	19	7 240	10	25	16 276
5	20	8 421	11	26	18 279
6	21	9 724			

② 固定图顶点规模,以非叶子节点的出度为变量。

优化算法优化程度与顶点规模有关。顶点规模与运行时间成负相关。固定图顶点规模,以非叶子节点的出度为变量,观察其优化效率。固定图顶点数为6 000,令任意非叶子节点的出度逐渐增大,使得深度随之变浅。非叶子节点的出度与深度之间的关系如表3所示。

表3　顶点数为6 000的深度与出度之间的关系

编号	出度	深度	编号	出度	深度
1	2	12	6	7	5
2	3	8	7	8	5
3	4	7	8	9	4
4	5	6	9	10	4
5	6	5			

③ 固定非叶子节点的出度,以深度为变量。

深度作为正则树的层数,最大程度地影响正则树的顶点规模,此实验研究深度与优化算法效率之间的关系。固定图的非叶子节点的出度为2,令其深度逐渐增大,深度与节点数之间的关系如表4所示。

表4　出度为2的深度与节点数关系

编号	深度	节点数	编号	深度	节点数
1	3	7	7	9	511
2	4	15	8	10	1 023
3	5	31	9	11	2 047
4	6	63	10	12	4 095
5	7	127	11	13	8 191
6	8	255	12	14	16 383

通过两种实验数据设计,应用运行时间和并行加速比两个指标,分析其优化程度,获得其算法优越性。

4.3　实验结果与分析

4.3.1　权值满足泊松分布的随机稠密图实验分析

程序运行时间如图8所示,多标号p的Dijkstra算法对传统的Dijkstra算法有明显的改进。且当图的顶点数越多,多标号的Dijkstra串行算法对Dijkstra算法的效率改进越明显,最高在顶点数为20 000上节约995 ms(见图8点A、B)。

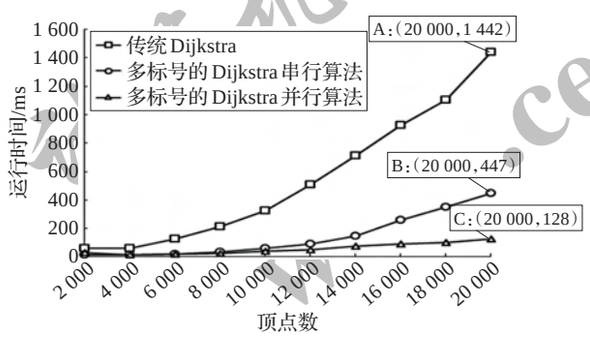


图8　泊松分布随机图运行时间

基于并行计算的Dijkstra算法又在多标号p的Dijkstra算法优化的基础上,进一步提高了计算效率,优化程度见图8。可见两种优化都比传统的Dijkstra的计算速度快,且优化程度随着图规模的增大而增大。多次重复实验也同样显示此规律。

实验表明本方法对传统Dijkstra算法的优化效果明显。下面以并行加速比评判并行算法对多标号p的Dijkstra算法的优化程度。

从并行加速比的数据可得:随着图顶点规模的扩大,加速比呈增长的趋势。由于线程开销的问题,图顶点规模6 000以下的加速比小于1。此现象表明当图顶点的规模较小时,使用并行的优化的效果不明显。但当数据顶点规模到达6 000点时,加速比已经达到1.05(见图9点A),6 000以上的图加速比都能处于较优的位置。当数据顶点规模为20 000点时,加速比为3.49(见图9点B),此时已经较为接近理想加速比。

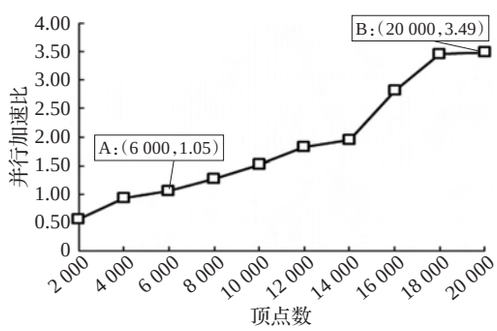


图9　正态数据的并行加速比

上述两个指标的数据表明,Dijkstra算法的多标号p优化对于稠密图有较好的优化效果,计算效率有明显的提升。对于大数据集合,算法的并行优化比串行优化效率更高。说明本文的并行算法优化对稠密图模型的计算有较好的效果。

另一方面,对于节点数较小的图,其优化效果不明显。原因是由于并行计算中线程的切换需要耗费一定的时间成本,算法的优化程度被此时间成本一定程度地抵消一部分,优化效果不明显。因此本算法更适用于规模较大的模型。

4.3.2　正则树实验数据分析

(1)固定深度,以非叶子节点的出度为变量。

在相同的深度下(固定深度为4),不同的非叶子节点出度的程序运行时间如图10所示。在深度为4的情况下,非叶子节点的出度为14时(见图10点A、B、C),顶点数为2 955。非叶子节点的出度小于14时,三种算法的运行时间相差不大,甚至出现有负优化的现象,这是优化效果被线程开销抵消的结果。当非叶子节点的出度大于14时,多标号p的Dijkstra算法相对于传统Dijkstra算法的计算效率明显提高,优化效果大于线程开销,使得优化效果呈现正面效果。而另一方面,基于并行计算的Dijkstra算法对比于多标号p的Dijkstra算法更加高效。

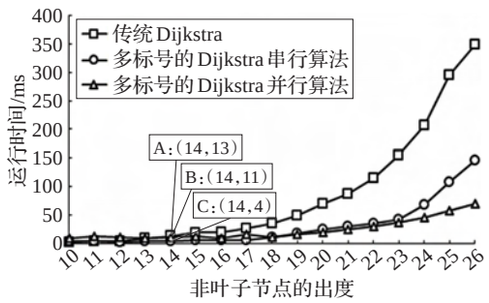


图10　正则树中不同出度的运行时间

并行加速比的变化如图11所示。在深度为4的情况下,非叶子节点出度等于18的情况下,并行加速比达到1(见图11点A),往后随着非叶子节点出度的增大,并行加速比逐渐增大,且在非叶子节点出度达到26时,并行加速比达到2.12(见图11点B),处于较优的状况。

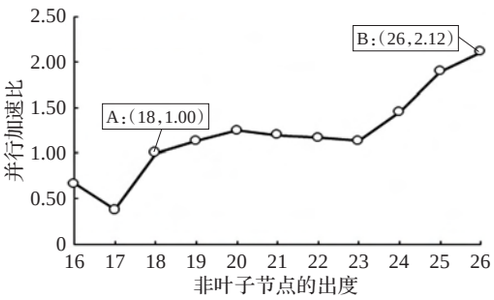


图11　正则树中不同出度的并行加速比

观察并行加速比指标,说明了并行优化对于多标号Dijkstra算法的优化明显。

(2)固定图顶点规模,以非叶子节点的出度为变量。

在固定顶点规模的图中,不同的非叶子节点出度的运行时间如图12所示。当顶点出度为2时,优化效果较差,但当非叶子节点出度逐渐增加,优化效果逐渐变好;当非叶子节点的出度达到4时,其呈现正相关的优化效果(图12点A、B、C)。且随着非叶子节点的出度的增加,优化效果越来越好。对于并行加速比,如图13所示,在非叶子节点出度大于3时,并行计算对此算法有正面影响,且随着非叶子节点的增加而增加。

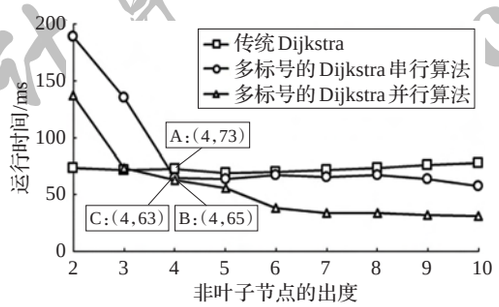


图12　正则树同样顶点规模下不同出度的运行时间

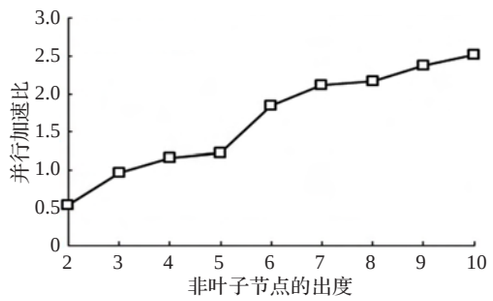


图13　正则树同样顶点规模下不同出度的并行加速比

实验表明:基于并行计算的Dijkstra算法的优化效果与非叶子节点的出度有关,但当出度较小时,效果并不明显。

(3)固定非叶子节点的出度,以深度为变量。

在非叶子节点出度同时为2的情况下,不同深度类正则图的程序运行时间如图14所示。当深度小于12时,优化算法对传统Dijkstra算法效率的提升并不明显。在非叶子节点的出度为2的情况下,深度为12时,

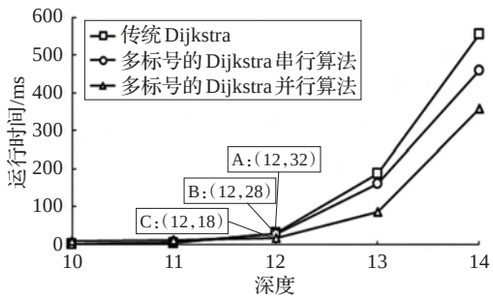


图14　正则树中不同深度的运行时间

顶点数为4 095,顶点规模较小,说明优化效果不理想的原因为图的顶点规模较小。当深度大于12,两种优化算法的运行时间明显短与传统Dijkstra算法(见图14点A、B、C)。

在非叶子节点出度同时为2的情况下,深度大于12类正则图的并行加速比都能大于1(见图15点A)。此实验说明顶点规模大于4 095时,三种算法都有明显不同,优化效果较好。

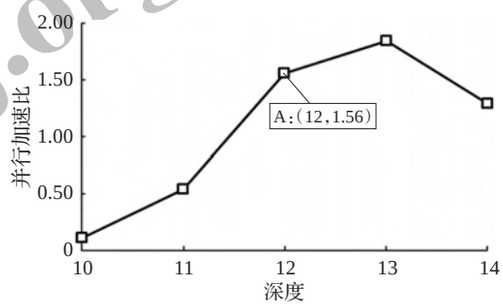


图15　正则树中不同出度的并行加速比

4.4　算法推广实验

此算法已被深圳微达安公司用于改进基于栅格模型的室内导航系统。基于栅格模型的室内导航模型是将平面图划分为若干大小相同的栅格,通过相邻栅格之间的关系构建的无向图模型。它的任意点都拥有若干条距离相同的邻边。下面通过实验验证优化算法的有效性。

实验环境为 $30\,\mathrm{m}\times40\,\mathrm{m}$ 的室内空间,一共八个房间,平面图按照一定规则分为4 096个栅格,相邻栅格定义边相连,构建具有4 096个顶点的无向图模型,具有指定起点和终点运用。三种算法的运行时间如表5所示。

表5　栅格模型图中两种算法运行时间对比

算法	运行时间/ms
传统的Dijkstra算法	34
多标号的Dijkstra串行算法	18
多标号的Dijkstra并行算法	11

多标号的Dijkstra串行和并行算法计算速率均高于传统的Dijkstra,且使用并行计算速率更高。可见优化算法在此现实场景中有较好的计算效率。

5　结束语

对于赋权正则树,证明了多标号Dijkstra串行算法时间复杂度为 $O(h(n+\lg n))$;多标号的Dijkstra并行算法时间复杂度为 $O(h(n/l+\lg n))$;都优于传统Dijkstra的时间复杂度 $O(n^2)$。通过仿真实验,也验证了时间复杂度排序的正确性。多标号p的Dijkstra算法对于较为稠密的一般图,具有较好的优化效率;对于并行优化,在顶点规模较大的(顶点数大于6 000)图有较优的优化效

率。对于正则树,非叶子节点出度和深度对于两种优化算法的优化效率都有影响。当固定正则树的深度为4时,非叶子节点的出度超过14时,两种优化算法的优化效率都与非叶子节点的出度成正相关;当固定非叶子节点为2时,当正则树的深度大于12时,两种优化算法对Dijkstra算法有正优化的效果;在同样的图顶点规模下,当非叶子节点的出度大于3时,两种优化算法的优化效率都与非叶子节点的出度成正相关。相对于文献[2]的并行优化,本文的并行优化的并行加速比高于其最高的2.06。

由公式(1)可知,本文的优化算法对于顶点数较大的稠密图都有很好的优化效果。对于非稠密图,当图的顶点向外邻点数基本一致时,此时图与正则树相似,也适用于本优化算法。

参考文献:

[1] Dijkstra E W.A note on two problems in connexion with graphs[J].Numerische Mathematik,1959,1(1).

[2] 张翰林,关爱薇,傅珂,等.Dijkstra最短路径算法的堆优化实验研究[J].软件,2017,38(5):15-21.

[3] 杨晓花,武继刚,史雯隽,等.稳定的最短路径树及其构造算法[J].计算机工程与科学,2016,38(3):418-424.

[4] 王树西,吴政学.改进的Dijkstra最短路径算法及其应用研究[J].计算机科学,2012,39(5):223-228.

[5] Bertsekas D P,Guerriero F,Musmanno R.Parallel asynchronous label-correcting methods for shortest paths[J].Journal of Optimization Theory and Applications,1996,88(2):297-320.

[6] 逄淑玲,王晓升.Dijkstra算法的并行实现[J].微型机与应用,2017,36(9):25-27.

[7] 李平,李永树.一种基于Dijkstra并行线程算法的研究与实现[J].测绘与空间地理信息,2014,37(9):50-53.

[8] 于方.MapReduce下的Dijkstra并行算法研究[J].阴山学刊(自然科学版),2018,32(1):66-71.

[9] 孙文彬,谭正龙,王江,等.最短路径算法的并行化策略分析[J].地理与地理信息科学,2013,29(4):17-20.

[10] 黄跃峰,钟耳顺.多核平台并行单源最短路径算法[J].计算机工程,2012,38(3):1-3.

[11] 徐唐剑.A~*寻路算法的并行化设计及改进[J].现代计算机(专业版),2018(21):44-49.

[12] 范炯,朱志宇.基于MapInfo的Dijkstra最短路径算法研究[J].江苏科技大学学报(自然科学版),2017,31(1):79-83.

[13] 彭定旭,冀肖榆.Dijkstra算法的Java实现方式及优化[J].黑龙江科技信息,2017(4):166-167.

[14] Fredman M L,Tarjan R E.Fibonacci heaps and their uses in improved network optimization algorithms[C]//25th Annual Symposium on Foundations of Computer Science,1984.

[15] 柯忠义,蒋辉.概率论与数理统计[M].北京:科学出版社,2012:1-205.