# 小步最短路径算法及其存储优化方法

**摘要：**基于松弛操作的最短路径算法通过反复更新节点的路径值来查找最短路径，给计算机系统的功耗、高速缓存访问和内存占用带来了严峻的挑战。本文中的小步最短路径算法不使用松弛操作，通过增加最小的路径值让最小数量的节点一次性地确定最短路径，用1个比特来标记节点是否确定最短路径，以大幅降低被高频访问的数据所占用的存储空间。本文还提出了一种聚类方法，用于提高访问标记节点是否确定最短路径的数据的性能。理论推演和实验结果表明，小步最短路径算法具有更好的可扩展性，其性能显著优于Dijkstra算法，图越大越复杂其性能优势越大。

**关键词：**Dijkstra、松弛操作、最短路径算法、计算机、高速缓存、聚类

## 1引言

图论研究由线连接的点所组成的图形结构及其性质，是离散数学的核心组成部分之一。最短路径问题旨在寻找图中两个节点之间的最短路径，是图论中最经典的算法之一。这个问题在计算机科学、运筹学、社交网络分析、地理信息科学、交通系统等领域广受关注[7,10,18]。

很多实际问题都可以通过抽象，转化为最短路径问题。比如，在道路交通网络中选取出行路线，在计算机网络中确定信息流在路由器间的最佳传输路径，以及在社会关系网络中计算两个陌生人之间的联系紧密度等。在Dijkstra提出最短路径算法[1]和Bellman-Ford[2]针对Dijkstra算法进行的开创性工作之后，大量专家学者围绕最短路径问题进行了深入研究[3-9]。已知有向非负加权图上的最短路径算法[1-9]，如Dijkstra、Bellman-Ford、Johnson和A\*等，都是基于松弛操作的方法，这对计算机系统的功耗、高速缓存访问和内存占用造成了严峻的挑战。研究不使用松弛机制，性能更好的最短路径算法具有重要的理论和实践意义。

在第二部分，本文以Dijkstra算法为例，分析基于松弛操作的最短路径算法（松弛算法）的基本思想和存在的问题。在第三部分，给出了本文使用或定义的概念、定理及其推理。第四部分阐述小步最短路径算法[24]（小步算法）及其基于最短路径算法的聚类方法[25]，以优化小步算法访问计算机高速缓存的效率。第五部分，分析小步算法和Dijkstra算法的时间复杂度和空间复杂度，提出基于可观测量分析算法的时间复杂度。第六部分，在多个公开测试数据集上测试并构建回归模型，分析小步算法与Dijkstra算法的性能。第七部分，总结本文取得的成果，说明存在的问题和未来改进的方向。

如无明确说明，本文中的最短路径算法都是基于最小堆实现的。理论推演和测试表明，小步算法不仅比Dijkstra算法具有更好的时间复杂度和空间复杂度，其通用性和可扩展性也显著优于Dijkstra算法。

## 2相关研究

已有松弛算法通过反复更新节点的路径值来查找最短路径。具体来说，算法维护两个集合：S为已经找到最短路径的节点集，Q为还未找到最短路径的节点集。对最近加入S中的节点u，算法会考察每个与u相连且不在S中的节点v，如果dist[u]+length(u,v)<dist[v]，则用dist[u]+length(u,v) 更新v的权重值dist[v] ，即一次性地处理完u的所有邻接边。

这种方法的一个主要系统开销是路径值的重复读取。特别是当记录节点的路径值需要多个字节的存储空间时，如果使用数组存储路径值会浪费大量内存，如使用哈希表则会引入过多的哈希值计算，成为系统运行的一个瓶颈。

针对松弛算法对节点路径值的重复读问题，本文提出的小步算法，在逐步增加路径长度的过程中，依次确定节点的最短路径，不存在无效的路径值更新。小步算法用1个比特来标记节点是否确定最短路径，可大幅降低被高频访问的数据所占用的存储空间。

为了进一步提高小步算法的执行效率,本文还提出了一种聚类方法。具体来说,该方法通过最短路径算法把节点聚类在一起，将同一聚类内节点相关数据存储在连续内存中，如将标记同一聚类中节点是否确定最短路径的1比特数据，存储在同一系统高速缓存块内，以大幅提高小步算法读系统缓存的性能。

## 3定义与定理

### 3.1 有向非负加权图

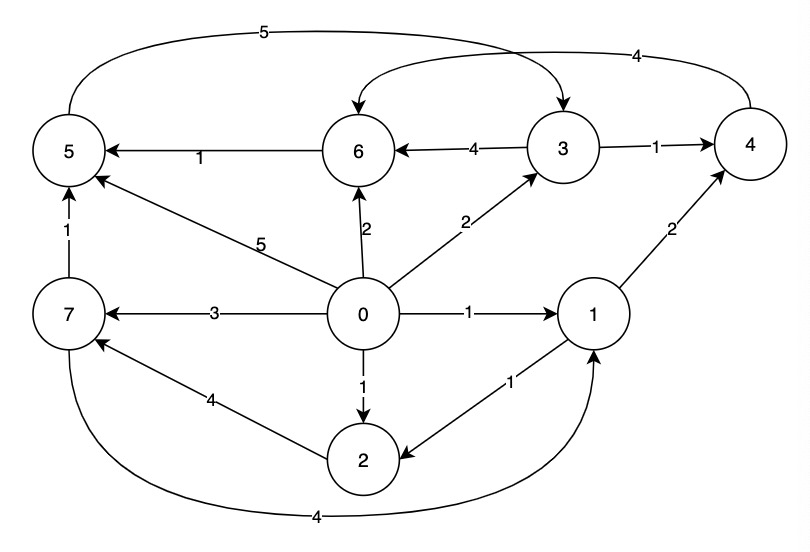
有向非负加权图G=(V,E)由顶点集V和有向边集E组成。有向边始点用from表示，终点用to表示，用(from,to)表示从顶点from指向顶点to的有向边。用|V|表示顶点集V的基数，用|E|表示边集 E 的基数。顶点用非负整数表示，集合[N]={0,1,2,…,N-1}表示顶点集V，V=[N]且|V|=N。权重函数 weights(from, to)表示有向边(from,to)的权重。

### 3.2 小步最短路径定理

**定理一：**在图G中，设定路径长度为0，从源节点的开始，沿未确定最短路径的邻接边逐步向外移动，路径长度每次递增一个单位，经过足够大的移动次数L，可以找到源节点到所有可到达节点的最短路径。首次到达节点时的路径长度，即为源节点到该节点的最短路径之一。

证明：因为权重值非负，所以从起点首次到达节点的路径就是该顶点的最短路径之一。设，W为所有邻接边权重和，weights(i)是第i号邻接边的权重值。假设存在节点v和正整数K，源节点到节点v存在最短路径，在经过W+K次移动才首次到达v。因为权重值非负，如果移动的次数大于W，则一定存在经过多次的邻接边，对应的from顶点被到达过多次。假设不成立，即在W次移动内得到了起点到可到达节点的所有最短路径信息，W就是一个这样的L值。

**图一**



在图一中，设定源节点0的最短路径长度为0，从源节点0开始，逐步增加一个最小值delt(i)=1，使得二者的和len(i)=len(i-1)+delt(i)到达数量最少节点，即有最小数量的节点确定最短路径值，len(0)=0。具体实施中可用邻接边直接计算len(i)的值。len(1)= len(0)+delt(1)=weights(0,1)=1，确定节点1和2的最短路径；len(2)= len(1)+delt(2)=weights(0,3)=2，确定节点3和6的最短路径；len(3)= len(2)+delt(3)= weights(0,1)+ weights(1,4)=3，确定节点4、5和7的最短路径。

### 3.3图定理及其推论

设图G中节点的出度序列为；为节点i在确定最短路径后需要为邻接节点向堆中加入数据项的比率。在松弛算法中，为需要更新邻接节点路径值的比率。在小步算法中，为需要切实处理的邻接边的比率。

**定理二：**存在有向非负加权图，Dijkstra算法在其上存在无穷个节点使得，小步算法在其上不存在无穷个节点使得。

证明：构造满足条件1、2和3的图，很明显该图就是一个这样的图。

1邻接边(i,i+1)存在，且；

2当i+1<j 时，邻接边(i,j)存在，且；

3当时，邻接边(u,0)存在,且。

在图的上述实例中，如果源节点不是节点0：初始化时（第0次迭代），源节点的路径值更新为0，其它节点的路径值更新为+∞；在第1次迭代中，确定源节点的最短路径值为0，节点0的路径值更新为，编号为源节点+1的路径值更新，其它编号大于源节点的路径值更新为。

上述Dijkstra算法执行步骤表明，节点0和源节点会把图中节点路径值的更新划为两个部分：以节点0为起点的最大值为N的更新序列，和以源节点为起点的最大值为的更新序列。算法不会用N到间的任何值去更新编号大于等于的任何节点的路径值，会大幅降低堆中的数据，显著提高算法的时间和空间复杂度。

对于小步算法而言，当源节点是节点0时，小步算法在其上不存在无穷个节点使得。具体表现为，第i次迭代可确定第i个节点的最短路径值为，只有邻接边(i,i+1)加入到堆中。

**定理三：**存在有向非负加权图，小步算法在其上存在无穷个节点使得，Dijkstra算法在其上不存在无穷个节点使得。

证明：构造满足条件1、2和3的图，很明显该图就是一个这样的图。

1邻接边(0,i)存在且；

2当i+1<j 时，邻接边(i,j)存在，且，；

3当时，邻接边(u,0)存在，且。

在图的上述实例中，如果源节点不是节点0：初始化时（第0次迭代），源节点的路径值更新为0，其它节点的路径值更新为+∞；在第1次迭代中，确定源节点的最短路径值为0，节点0的路径值更新为，编号为源节点+1的路径值更新，其它编号大于源节点的路径值更新为。

上述小步算法执行步骤表明，节点0和源节点会把图中节点路径值的更新划为两个部分：以节点0为首的更新序列和以源节点为首的更新序列，这些更新的值会被权重增加的路径值替换。算法不会为节点j向堆中加入路径值为的数据项，其中源节点小于等于节点j且大于节点i，会大幅降低堆中的数据，显著提高算法的时间和空间复杂度。

对于Dijkstra算法而言，当源节点是节点0时，在其上不存在无穷个节点使得。具体表现为：在第一迭代中，第i个节点路径值更新为；在后续的迭代中，路径值不再更新，即为第i个节点的最短路径值。

**定理四：**存在有向非负加权图，小步算法和Dijkstra算法在其上都存在无穷个节点使得。

证明：构造满足条件1、2和3的图，很明显该图就是一个这样的图。

1邻接边(i,i+1)存在，且；

2当i+1<j 时，邻接边(i,j)存在，且；

3当时，邻接边(u,0)存在，且。

**推论一：**在定理二、定理三和定理四中，条件3对节点0限制的可替换为：存在有限K个节点，对于其中任一节点至少存在一个节点，使得邻接边(u,w)存在，且。

证明：有限K个节点会将N个节点划分为K+1份，其中必然存在一个划分中包含无穷多的节点。对于任意源节点，在确定有限个节点的最短路径后，一定会到达一个包含无穷多的节点的分区。在该分区内存在无穷个节点使得。

**推论二：**在定理二、定理三和定理四中，如存在无限个节点w，则不存在无穷个节点使得。

在众多的实际应用中，即便节点规模可不断地增加，但图中特定节点的邻接边数量总有上限。没有综合考虑松弛算法读取路径值的复杂度与需要最小堆操作的数据项复杂度，割裂了小步算法确定节点是否确定最短路径的复杂度与需要最小堆操作的数据项的复杂度。对于小步算法和松弛算法而言，存在无穷个节点使得，在很大概率上只在理论上是可能的。

**定理五**：基于最小堆的Dijkstra算法，在确定第j个节点的最短路径后，堆中存留的数据项的数量，大于等于，基于最小堆的小步算法，在确定第j个节点的最短路径后，堆中存留的数据项的数量，其中，。

证明：假设存在一个节点，在确定该节点的最短路径后，Dijkstra算法在堆中存留的数据项的数量小于小步算法在堆中存留的数据项的数量。如果该假设成立，在确定该节点的最短路径后，则必然存在一条邻接边(m,n)，使得：节点m已经确定最短路径，节点n还未确定最短路径；节点n只在小步算法中在堆中存留的数据项的数量，在Dijkstra算法中没有在堆中存留的数据项。这与Dijkstra算法使用的松弛操作相矛盾。

### 3.4 可观测量

设，j为节点确定最短路径的序号，为最短路径算法确定两个节点的最短路径之间向堆中加入的数据项数量。 设为j对应的节点i的出度，为对应的。在Dijkstra算法中，在小步算法中。根据定理二和定理三可知，Dijkstra算法中的可大于、等于或小于小步算法中的，二者的大小关系无法确定。

根据定理五，可知在每一步的迭代中，Dijkstra算法中的大于等于小步算法中的。因此，理论上Dijkstra算法中最小堆中元素数量，并不会明确地显著大于小步算法中最小堆中元素数量，二者的差异由节点的出度和邻接节点的重复程度决定。

是一个非常好的估量算法的主要操作和时间复杂度的可观测量。当较小时，算法确定邻接节点是否确定最短路径的操作是算法的瓶颈，即Dijkstra算法对路径值读取，小步算法对标记节点是否确定最短路径标记的读取。随着值的增加，算法中的堆操作将会对算法的性能有着越来越重要的影响，会成为算法的性能瓶颈之一。当很大时，尤其是当|V|、甚至 也很大时，算法中的堆操作会成为算法的唯一瓶颈。

### 3.5复点

如上述图中的w节点和源节点，本文把降低Dijkstra算法更新路径值的节点和降低小步算法向堆中加入数据项的节点，称为**复点**。根据图、和及其对复点的分析可知，不同的最短路径算法在同一图上的效率可能截然不同。诸如图、和，只有一个复点或少量复点的图是很少见的，图中节点互为复点更为普遍。因此，在现实世界中的多数图上，Dijkstra算法和小步算法确定邻接节点是否确定最短路径的操作是瓶颈所在。

复点越多则越小，复点会显著降低Dijkstra算法和小步算法的时间复杂度。复点会打断Dijkstra算法更新节点路径值的链条，给予Dijkstra算法把更优的路径值尽早地传递到节点的能力，能够大幅降低Dijkstra算法对节点路径值的更新次数及其向堆中加入的数据项的频率。Dijkstra算法更新节点路径值的次数越多算法的性能越差。Dijkstra算法更适用于树高相对低的最短路径树的计算。

复点会让小步算法尽早发现更多的邻接节点已经确定最短路径，能够显著降低小步算法需要处理的邻接边数量，会大幅降低小步算法向堆中加入的数据项的频率。小步算法向堆中加入的数据项越多算法的性能也会越差。图中复点及其邻接节点越多，小步算法在大规模问题上的适用性和效率就越显著。

## 4小步算法

### 4.1小步算法

在确定一个节点的最短路径后，松弛算法会立即处理该节点的所有邻接边。这既是松弛操作的本质，也是松弛算法的最小堆中数据项过多和路径值访问效率低等问题的根源。在每次迭代中，小步算法只为一个邻接节点对应的邻接边在最小堆中加入一个键值对，完全克服了松弛操作带来的各种效率问题，显著提高了最短路径算法的效率。

S1.初始化源节点的最短路径值为0，其它节点的最短路径值为+∞；

S2.将已确定最短路径且有邻接节点未确定最短路径的节点作为第一节点组成第一节点集；

S3.从第一节点集中的第一节点的未确定最短路径的邻接节点中，选择距离源节点最近的邻接节点确定最短路径并更新第一节点集；

S4.重复上述步骤S2-S3，直至找到所有期望最短路径为止。

在具体实施中，可用最小堆维护已确定最短路径且有邻接节点未确定最短路径的节点的相关数据。用S3中的邻接节点距离源节点的路径长度作为相关数据的键值。可用线性表来加速距离源节点最近的邻接节点的查找过程。

小步算法中的节点的最短路径值是一次确定的，可用一个比特来标记一个节点是否确定最短路径，每个第一节点在堆中有一个键值对。通过查询邻接节点是否确定最短路径，可优化掉与邻接边关联的很多计算和堆操作，避免了计算哈希值带来的瓶颈。小步算法会间断性地访问邻接边，这引入了少量的缓存开销。

### 4.2存储优化

如以连续的64比特为一个观察单元，小步算法在访问同一观察单元中的各个一比特数据时，分布的十分广泛。也就是说，在同一观察单元中的64比特，小步算法从首次访问和最后一次访问之间，访问了大量其它观察单元。小步算法的这种数据访问方式，给计算机缓存系统带来巨大的冲击，难以充分发挥出小步算法的优势，是算法的一个重要瓶颈。

可用聚类方法，使得小步算法对同一观察单元中比特的访问尽量集中在一定时段内。已有图节点聚类方法，如标签传播算法(LPA)[11]，是根据特征的符合程度，把不同的节点放入同一类别中，以保证同一类别内部节点是相似的，不同类别间的界限越大越好。

本文计算节点数量有上限的最短路径树，把树中的节点划为一个等价类，把同一等价类中节点关联的数据存储在连续的存储空间内。该方法是以系统访问数据的统计特征为根据来划分等价类的，提高同一观察单元内数据访问的集中程度，提高计算机缓存的命中率。

1. 选择预设数量的未划分等价类的节点；
2. 将选择的节点作为根节点并确定最短路径树，其中最短路径树中的节点都未划分等价类，等价类的基数具有上限，最短路径树中的节点数量小于等于等价类的基数上限；
3. 根据最短路径树中的最短路径信息及节点信息，确定一个最优树，并将其中节点划分为一等价类；
4. 重复上述步骤，直至所述有向图中所有节点均已划分等价类；
5. 编码所述等价类及等价类中的节点，得到各等价类的二元组；
6. 根据所述等价类的二元组，将所述等价类中节点的相关数据在按系统缓存行大小对齐的存储介质中存储。

选择未划分等价类的节点作为根节点，计算最短路径树，使最短路径树的节点集的基数不大于一个上限，把它们划归为一个等价类。进一步地，根据该等价类中节点信息，连续存储与其中节点相关数据，以更集中地访问同一观察单元，提高缓存系统的效率。

存在多种根节点的选择策略，如随机选择，或选择（最近）已划分等价类节点邻接节点，或选择距离（最近）已划分等价类节点邻接节点一定距离的节点。前两种选择策略比较简单，但是选择出的根节点附近的部分节点可能已经划分了等价类，这会严重限制树的形状。

最后一种根节点选择策略，会在一定程度上避免或降低树的形状受到限制。可继续迭代一定长度的路径，对当前等价类对应的最短路径树进行边缘的扩展，从扩展后最短路径树的叶子节点中选择根节点。可使用适当的最短路径算法，如Dijkstra算法或小步算法，为选择出的根节点计算最短路径树。

合并基数小于等价类的基数上限的等价类，充分利用存储空间，提高系统缓存块的利用率。可按照等价类确定顺序拆分、合并形成新的基数等于等价类的基数上限的等价类，或合并等价类使其基数等于等价类的基数上限。

如果用1个bit来标记节点是否确定最短路径，在64字节大小的缓存行的运行环境中，用64字节的连续存储空间存储一个等价类中节点是否确定最短路径，64字节可标记512（等价类的基数上限为512）个节点。用类别ID寻址不同的64字节的连续存储空间，用节点在类别内部的编号寻址等价类中每个节点对应比特。等价类的基数上限要根据CPU缓存的大小和图自身的复杂性来确定，其最大值为CPU缓存块中比特的数量。

在计算最短路径时，如果确定了某个节点的最短路径，则设置对应1比特数据为1。在处理其它邻接边时，先查询邻接节点to是否已经确定最短路径，若是，跳过对该邻接边的处理，若否，继续处理该邻接边。

### 4.3配对堆优化

当很大时，基于最小堆的Dijkstra算法和小步算法的主要操作都是插入和删除元素。尤其是当|V|、甚至 非常大时，最小堆所需要的内存空间会显著地增加，给系统的运行带来很大的不确定因素。在这种情况下，可采用空间复杂度为的配对堆[19-22]，用降低键值操作代替大量数据项的插入和删除。

## 5复杂度分析

### 5.1 Dijkstra算法

Dijkstra算法更新邻接节点的路径值的次数不大于|E|，堆高度不大于log|V|。Dijkstra算法的主要操作由决定：当非常大时，堆操作是算法的主要操作；当较小时，读取节点的路径值是算法的主要操作。目前Dijkstra算法的公认时间复杂度为，它脱离了算法更新节点路径值的数量、节点数量|V|和边的数量三者的关系，是个很宽泛的结果，实践指导意义较小。

Dijkstra算法涉及6种操作：计算路径长度、比较操作，访问邻接边和读节点路径值的复杂度都为；更新节点路径值的时间复杂度为；最小堆插入和删除元素操作的时间复杂度为。其中，计算路径长度和比较操作的效率很高；Dijkstra算法会连续访问节点的邻接边，能高效地利用系统的高速缓存系统；通过哈希表读节点的路径值是Dijkstra算法的主要操作；更新节点路径值操作的主要问题在于，系统的高速缓存与内存的数据同步，当很大时，也将是Dijkstra算法的一个重要操作；当较大时，最小堆插入和删除元素会成为Dijkstra算法的性能瓶颈。这里值得强调的是，如不考虑存储成本和分布式系统中的通信成本，如选择使用数组存储节点的路径值，最小堆插入和删除元素就是Dijkstra算法的主要操作，与无关。因此，使用哈希表时Dijkstra算法的时间复杂度为。

随着问题规模N的增长，有两个可能值，即，或。（1）如， ， 。（2）如，当N趋于无穷时如存在有限个使得， ， = 。（3）如，当N趋于无穷时如存在无穷多个i使得， = ， = 。

因此，当较小时，算法的时间复杂度为；当很大时，尤其是当|V|、甚至 也很大时，算法的时间复杂度为；当较大时，算法的时间复杂度介于二者之间，为。其中，为访问路径值的复杂度，为堆操作的复杂度，K为一个较大的数。

Dijkstra算法不具备较大的提升和改进空间。虽然可充分利用硬件资源完成松弛操作引入的冗余计算，但是这一方面会消耗大量的能源，另一方面也会占用进一步的算法优化和编译优化所需的资源。判断一个节点是否确定最短路径只需要1个比特即可，不需要读取路径值。生成大量中间结果、频繁地访问路径值、大量比较两个路径值的大小及其计算大量的哈希值等，都是松弛操作思想强行引入到Dijkstra 算法的，不是最短路径问题自身的要求。

### 5.2小步算法

小步算法同样使用6种重要的操作：访问邻接边、访问标记和比较标记操作的时间复杂度都为；更新标记的时间复杂度为；计算路径长度的时间复杂度为，最小堆插入和删除元素操作的时间复杂度为 。其中，邻接边的访问是间断性地进行的，会有一定量的缓存开销，是小步算法目前无法克服的问题；比较操作是针对1比特的标记数据，比Dijkstra算法简单；可用数组存储节点的标记数据，把类别号和类别中节点编号编码到一个整数中，可用循环展开非常高效地通过移位操作来读取类别标号，当较小时，是小步算法的主要操作；每个节点只需要更新一次标记数据，与无关，优于Dijkstra算法更新数据的时间复杂度；小于Dijkstra算法计算路径长度的时间复杂度；当较大时，最小堆插入和删除元素会成为Dijkstra算法的性能瓶颈。使用数组存储标记数据时，小步算法的时间复杂度为。

随着问题规模N的增长，有两个可能值，即，或。（1）如， = ， = 。（2）如，当N趋于无穷时如存在有限个使得， = ， = 。（3）如，当N趋于无穷时如存在无穷多个i使得， = ， = 。

因此，当较小时，算法的时间复杂度为；当很大时，尤其是当|V|、甚至 也很大时，算法的时间复杂度为；当较大时，算法的时间复杂度介于二者之间，为。其中，为访问标记节点是否确定最短路径的复杂度，为堆操作的复杂度，K为一个较大的数。

Dijkstra算法与小步算法的时间复杂度公式是一样的。但是，小步算法的时间复杂度对应的基础操作及其高频访问的存储空间的复杂度都显著优于Dijkstra算法。Dijkstra算法中增长率要远大于小步算法，所以理论上，当堆操作成为小步算法的主要操作时，小步算法的性能也会显著优于Dijkstra算法。

### 5.3空间复杂度

算法中频繁访问的数据结构或变量的存储空间的复杂度，是算法非常重要的性能指标。如果这些高频访问的存储空间复杂度过高，将会严重影响算法的效率和可扩展性。算法应该尽量使用空间复杂度较低的数据结构，来存储和管理这些高频访问的数据。

如果使用8字节来存储节点的路径值，Dijkstra算法高频访问的内存的空间复杂度为。对这 8 字节的频繁读及哈希值的计算，是Dijkstra算法的一个重要操作。小步算法会一次性地确定节点的最短路径值，如果使用1比特标记节点是否确定最短路径，那么小步算法高频访问的内存的空间复杂度为。

当较小时，小步算法访问的空间需要的时间复杂度为，Dijkstra算法访问的空间需要的时间复杂度也是。小步算法显著优于Dijkstra算法。

与松弛算法相比，小步算法具有巨大的存储优势。小步算法的高频访问数据只为松弛算法的1/64，可节省下几十、几百GB、甚至数TB的内存，即使是具有4000亿节点的网络[16,17]也只需要47GB的标记数据。因此，小步算法可用数组而不是哈希表来存储高频访问的标记数据，用移位运算来代替大部分松弛算法所需的乘法运算，大幅提高算法的效率。

这种存储优势对于硬件的更新和电能的消耗都是一个非常大的优化。尤其是在使用多台服务器分布式地计算最短路径时，保持多台服务器之间节点路径值的同步是一个复杂的问题，小步算法这种存储的优势会带来巨大的通信优势。

## 6测试

### 6.1数据集

基于SNAP[12]，测试系统使用四核Intel Core i7处理器，基本时钟速度为 2.2 GHz，配备 16 GB 的 DDR4 RAM, 运行频率为1600 MHz。本文使用8个生物网络公开测试数据集[13]，(i)bio-CE-CX，(ii)bio-CE-GN，(iii)bio-DM-CX，(iv)bio-HS-CX，(v)bio-SC-HT，(vi) bio-WormNet-v3，(vii)bio-mouse-gene，(viii) bio-human-gene1，其中节点为基因，边为基因之间的链接。本文还使用7个生成的公开测试数据集[14], (i) datagen-7\_5-fb，(ii) datagen-7\_6-fb，(iii) datagen-7\_7-zf，(iv) datagen-7\_8-zf，(v) datagen-7\_9-fb，(vi) datagen-8\_0-fb，(vii) datagen-8\_1-fb，其中节点数量达到1000万级，边数量达到亿级别。

### 6.2实验设定

stats.blue[15]是一个免费的在线统计工具，提供回归分析等多种统计分析功能。本文使用stats.blue提供的Multiple Linear Regression Calculator功能（https://stats.blue/Stats\_Suite/multiple\_linear\_regression\_calculator.html）来构建线性回归模型，分析Dijkstra算法和小步算法的性能关系。

为了探索构建的线性回归模型图像的特性,本文使用在线 3D 绘图计算器 Desmos[18]。该工具提供了强大的三维绘图功能,可以直观地展示各类三维函数图像。

在非负加权图中， Dijkstra算法是性能最好的单源最短路径算法，它是现有各种加权图最短路径算法的基础。设为Dijkstra算法所用时间与小步算法所用时间的比值-1，为节点数量，为节点的平均邻接边数量，， 为小步算法为每个节点加入堆中的平均数据项数量，表示Dijkstra算法为每个节点加入堆中的平均数据项数量。

### 6.3数据分析

在上述15个非负加权图上，多次各随机个抽取10个、100个和1000个节点，查询其到其它节点的最短路径。实验结果表明，，小步算法在99.7%的样本上优于Dijkstra算法，其中性能1.5倍以上的占比85.8%，性能2倍以上的占比61.5%，性能3倍以上的占比42.2%。当遍历的节点数量大于11000时，小步算法100%优于Dijkstra算法。

**表一：样本空间y值统计**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 150样本 | 1500样本 | 15000样本 |
| 异常数据 | 0 | 18 | 178 |
| Y<=0 | 0 | 4 | 35 |
| 0<Y<=0.5 | 19 | 206 | 2056 |
| 0.5<y<=1 | 40 | 351 | 3604 |
| 1<Y<=1.5 | 19 | 219 | 2207 |
| 1.5<Y<=2 | 3 | 15 | 166 |
| 2<Y<=2.5 | 1 | 75 | 485 |
| 2.5<Y<=3 | 20 | 177 | 1040 |
| 3<Y<=3.5 | 11 | 187 | 1902 |
| 3.5<Y<=4 | 15 | 116 | 1600 |
| 4<Y<=4.5 | 13 | 77 | 982 |
| 4.5<Y<=5 | 2 | 38 | 467 |
| 5<Y<=5.5 | 2 | 16 | 212 |
| 5.5<Y | 1 | 1 | 66 |

在这些样本上，小步算法中的均值范围为(1,3.5)，Dijkstra算法的均值范围为(1,3.2)。在这些数据集上，是很小的，复点是常见节点，的均值与邻接边数量并不存在有效的函数关系。实验表明，当图邻接边数量增加时，Dijkstra算法对应的会缓慢地增加，小步算法对应的则增加的更为缓慢。在邻接边数量小的图中，后者均值大体上小于前者，随着图邻接边数量的增加，前者均值大体上将小于后者。

因此，在这些公开数据集上，小步算法和Dijkstra算法的时间复杂度都是。 在某些源节点上Dijkstra 算法会优于小步算法，而小步算法会在99.7%的概率地优于 Dijkstra 算法。

**图八：散点图**

在上述15个非负加权图上，各随机个抽取10个节点，查询其到其它节点的最短路径。去除异常数据项后，用Stats.Blue[15]构建自变量多元线性回归预测模型以分析、、之间的关系，构建因变量多元线性回归预测模型以分析小步算法与Dijkstra算法的性能关系。

### 6.4自变量线性回归模型

#### 6.4.1 回归模型

在实验数据上构建[15]回归模型（1）分析、、之间的关系。

（1）

， 递增； ， 时 递增， 时 递减。根据本文对的分析，随着的增加，和的整体趋势是递增的。 时 递减，这是因为和的取值受 取值的制约，即时，当有增量h时，如，则在大于99.7%的概率上，有最小增量t，使得。在小步算法和Dijkstra算法性能对比中，点在大于99.7%的概率上没有实际意义。

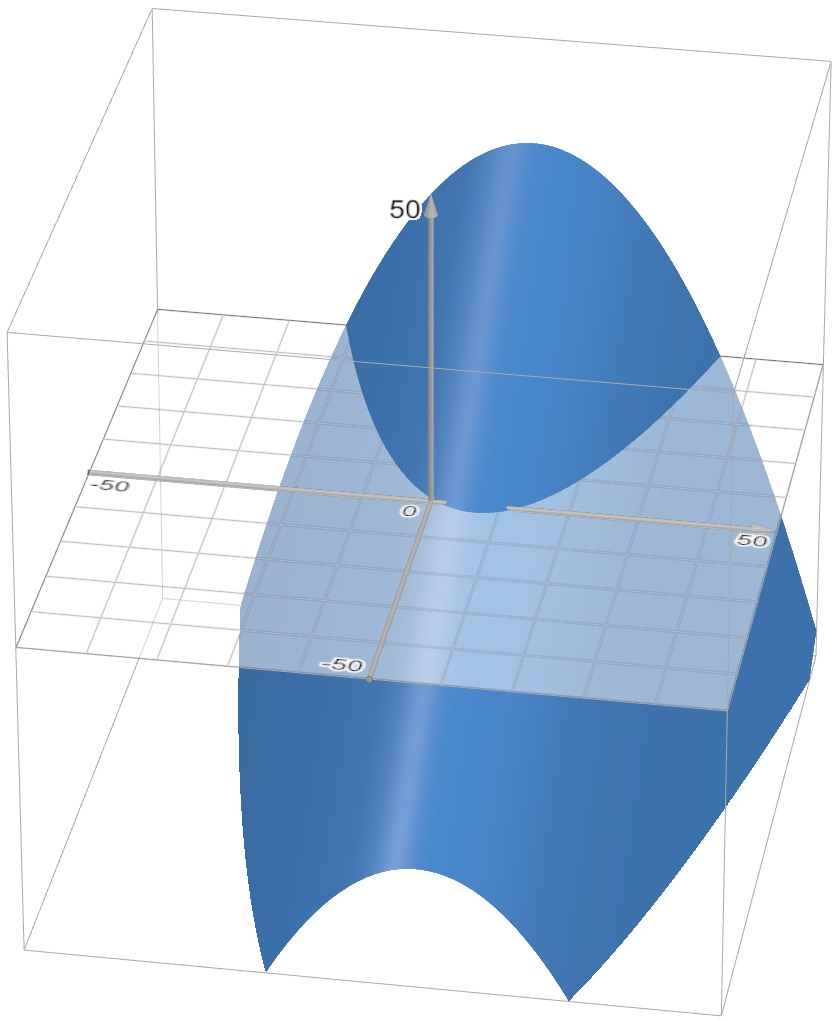
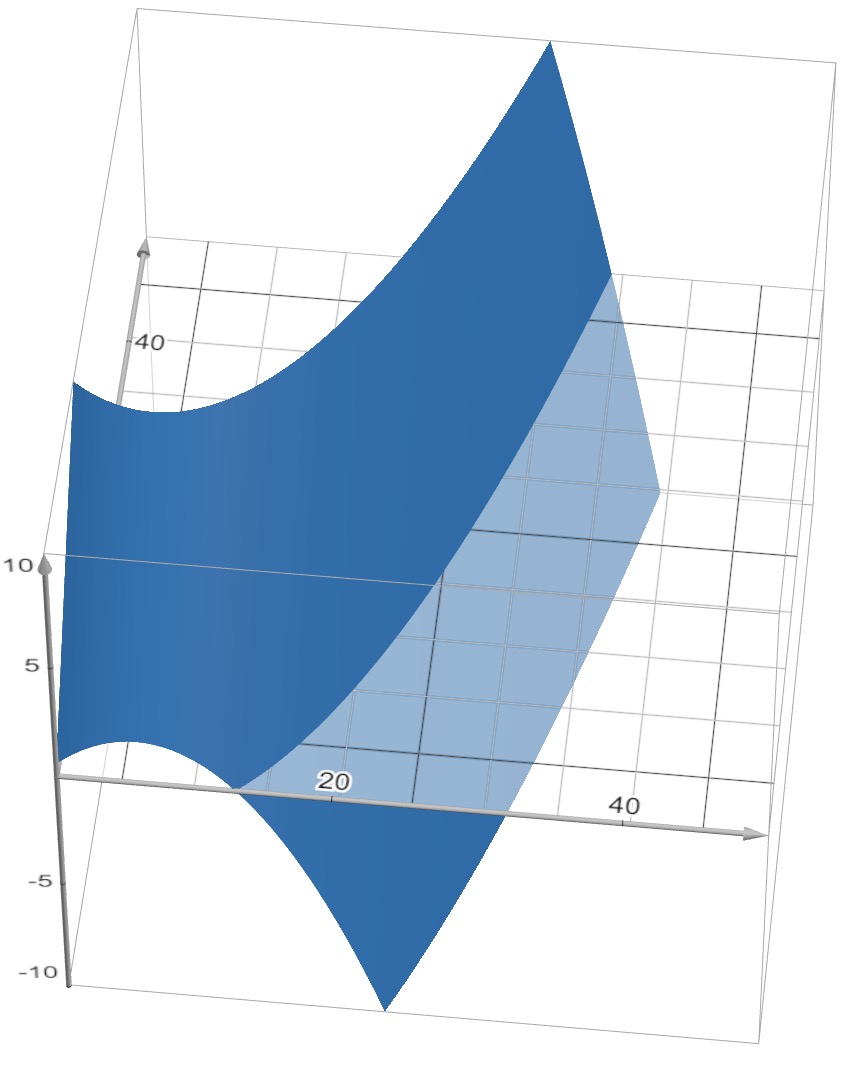
**表三：参数表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Predictor** | **Coefficient** | **Estimate** | **Standard Error** | **𝑡-statistic** | **𝑝-value** |
|  |  |  |  |  |  |
| **Constant** | ***𝛽0*** | ***-0.5884*** | ***0.1148*** | ***-5.1256*** | ***0*** |
|  |  |  |  |  |  |
|  | ***𝛽1*** | ***0.5524*** | ***0.0303*** | ***18.2085*** | ***0*** |
|  |  |  |  |  |  |
|  | ***𝛽2*** | ***0.6328*** | ***0.0942*** | ***6.7215*** | ***0*** |
|  |  |  |  |  |  |
|  | ***𝛽1,1*** | ***-0.0417*** | ***0.0023*** | ***-17.9493*** | ***0*** |
|  |  |  |  |  |  |

**表四：总体适合度**

|  |  |
| --- | --- |
| R-Squared: | =0.8408 |
|  |  |
| Adjusted R-Squared: | =0.8376 |
|  |  |
| Residual Standard Error: | 0.1683 on 146 degrees of freedom. |
|  |  |
| Overall F𝐹-statistic: | 257.0818 on 3 and 146 degrees of freedom. |
|  |  |
| Overall p𝑝-value: | 0 |

**图三：为横轴，为斜轴，为纵轴 ­**



**原点(1,1,0) 原点(0,0,0)**

本文使用 Desmos 3D[18] 绘图计算器，绘制函数的三维图像，如图三所示。从图像中可以清楚地观察到，该三维曲面是一个马鞍型曲面。在，区域内，增加h时，至少要增加，才能保证。越大，增加h时，会增加的量越大。

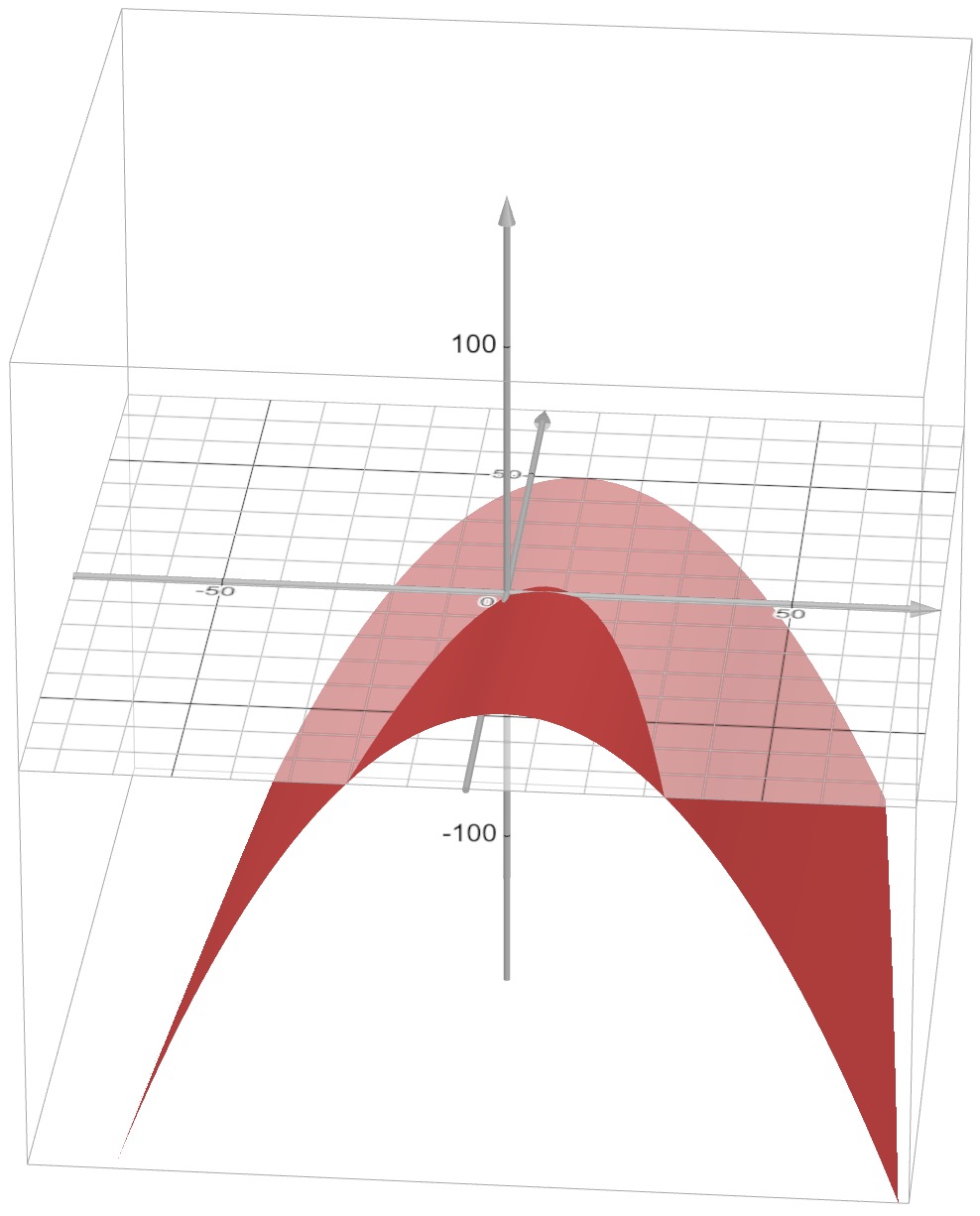
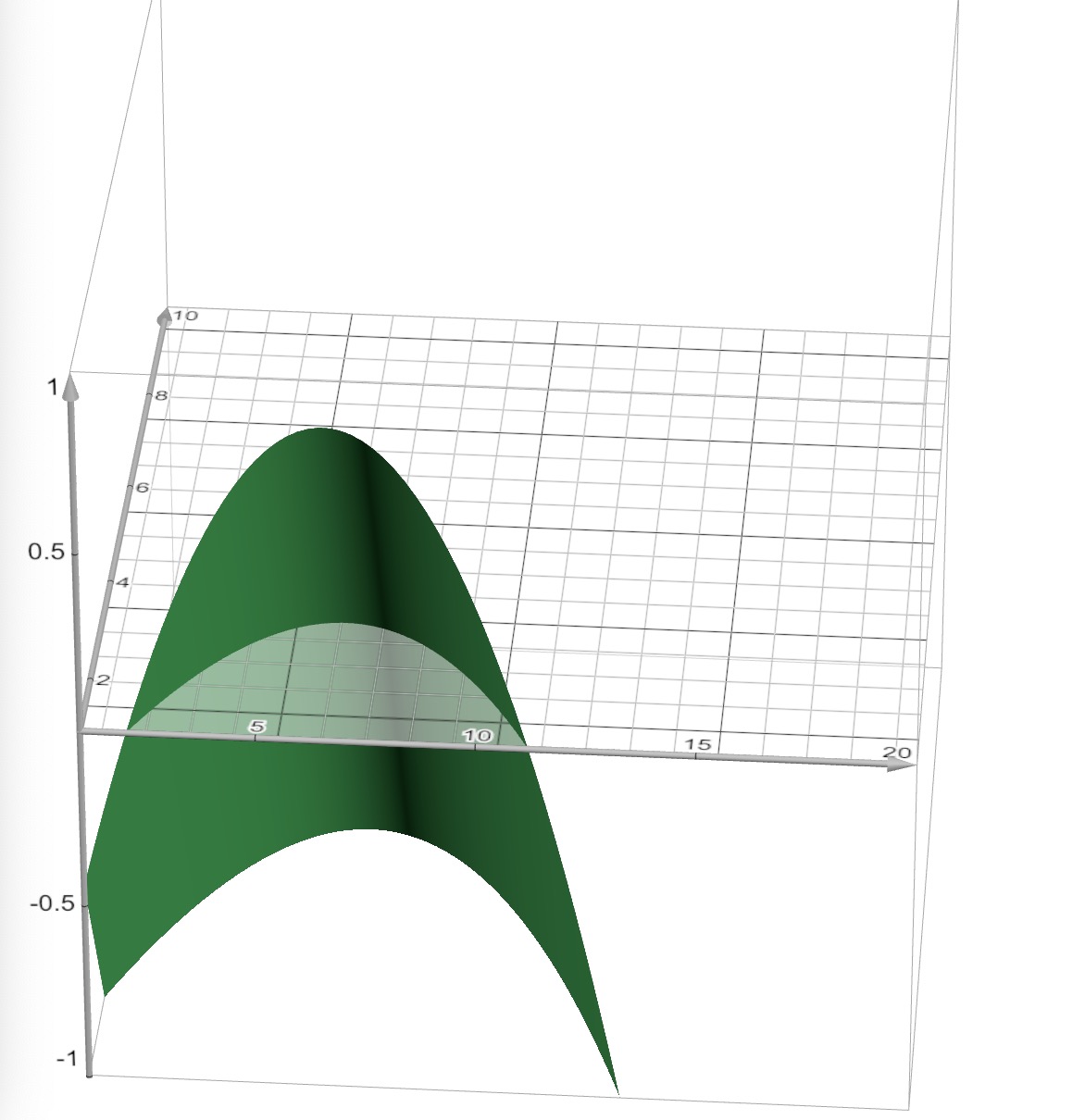
根据。当增加h时，至少要增加，才有。越大，增加h时，会增加的量越小。

从上述分析可知，增加h时，对的变化是敏感的，对的变化是不敏感甚至是负的。Dijkstra算法的性能对的变化是不稳定的，增加时，其时间复杂度会较快地上升。小步算法的性能对的变化是稳定的，的增加时其时间复杂度变化较慢。

#### 6.4.2 差值分析

（3）

**图四：为横轴，为斜轴，为纵轴 ­**



**原点(1,1,0) 原点(0,0,0)**

使用 Desmos 3D[18] 绘图计算器，绘制函数的三维图像，如图四所示。从图像中可以清楚地观察到，该三维曲面也是是一个马鞍型曲面。

，只在很小的区域 内大于 ，即小步算法为每个节点加入堆中的平均数据项数量，只在区域内大于Dijkstra算法为每个节点加入堆中的平均数据项数量，。当并且时，越大越小于，Dijkstra算法的稳定性不如小步算法。

根据

又根据（6）可知，当时，

（2）

公式（2）和（6）可知，，的，小步算法具有更好的稳定性，愈大则小步算法性能会越好于Dijkstra算法。

### 6.5因变量线性回归模型

是和的因变量，构建[15]回归模型（4）以分析、、之间的关系。

(4)

**表五：参数表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Predictor** | **Coefficient** | **Estimate** | **Standard Error** | 𝑡**-statistic** | 𝑝**-value** |
| Constant | 𝛽0 | 16.2389 | 0.7307 | 22.2246 | 0 |
|  | 𝛽1 | 0.7867 | 0.1618 | 4.8617 | 0 |
|  | 𝛽2 | -17.6992 | 0.8951 | -19.7727 | 0 |
|  | 𝛽1,1 | -0.3459 | 0.0232 | -14.9144 | 0 |
|  | 𝛽1,2 | 2.4953 | 0.124 | 20.1314 | 0 |

**表六：总体适合度**

|  |  |
| --- | --- |
| R-Squared: | =0.8962 |
|  |  |
| Adjusted R-Squared: | =0.8934 |
|  |  |
| Residual Standard Error: | 0.4822 on 145 degrees of freedom. |
|  |  |
| Overall 𝐹-statistic: | 313.103 on 4 and 145 degrees of freedom. |
|  |  |
| Overall 𝑝-value: | 0 |

**表七：方差分析表**

|  |
| --- |
|  |
| Source | df | SS | MS | 𝐹-statistic | 𝑝-value |
| Regression | 4 | 291.1715 | 72.7929 | 313.103 | 0 |
| Residual Error | 145 | 33.7108 | 0.2325 |  |  |
| Total | 149 | 324.8823 | 2.1804 |  |  |

**图六：残差直方**

根据模型（4）、表五、表六、表七和残差直方图六可知，在95%置信水平下，常系数、自变量和 及其、交互项对因变量都有显著影响，该回归模型的整体显著性非常好,模型中包含的所有自变量及其交互项能够很好地解释和预测因变量y的变化。

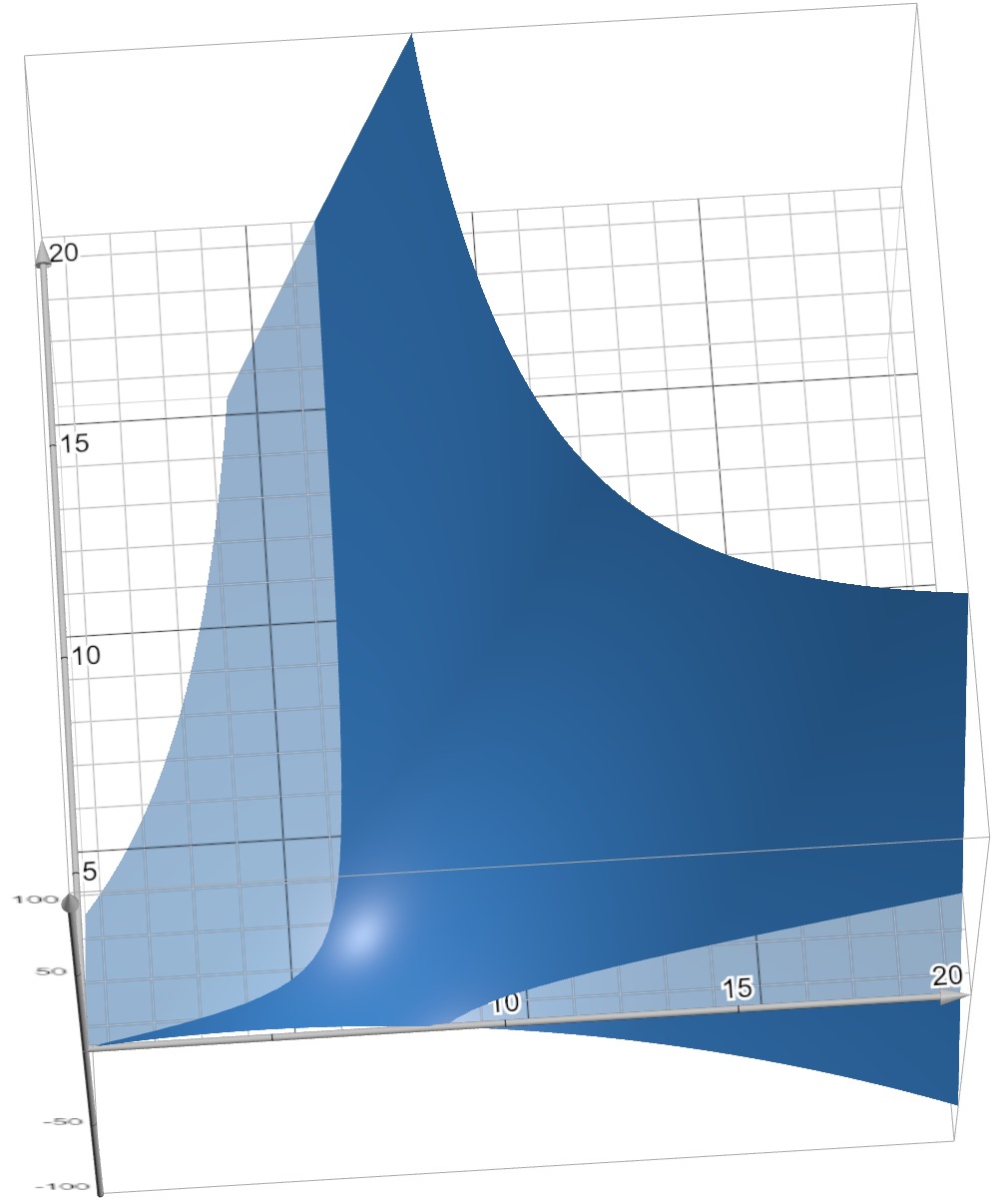
（5）

当时；或当时；或当时；或当时；或当时；或当时；或当时，随着的增加，y值迅速增加，有：

（6）

和的线性关系表明小步算法是非常稳定的。最大性能优势愈大，需要的最小值愈大。时，其值愈大，小步算法的性能愈好于Dijkstra算法。

**图七：为横轴，为斜轴，y为纵轴，原点(1,1,0)**



使用 Desmos 3D[18] 绘图计算器，绘制函数的三维图像，如图五所示。设，的区域。在大量的抽样测试中，的样本比例均不大于0.3%，且多数集中在靠近轴的的狭长区域内。根据公式（6）可知，当y为一特定值时，当比较小时，靠近轴的区域内有少量的的样本分布。当比较大时，样本会分布在与轴之间的区域内。

## 7总结

松弛算法需要在内存中保留节点的路径值，通过更新路径值来计算节点的最短路径。松弛操作对计算机系统的功耗、高速缓存访问、内存占用和分布式系统的通信都造成了严峻的挑战。

本文提出小步算法完全克服了松弛操作存在的本质问题。该方法只需使用一个比特来标记节点是否确定最短路径，可完全避免松弛操作引入的中间结果，一次性的确定节点的最短路径。用一个相对很小的连续内存空间，便可存储所有节点的1比特标记，可节省大量的内存空间。

算法高频访问的存储空间的复杂度，决定计算机缓存系统的效率，对算法性能有重大影响。小步算法高频访问的数据的空间复杂度是，远小于Dijkstra算法高频访问的数据的空间复杂度。本文基于已有最短路径算法提出了一种节点聚类算法，以高效的存储和访问与节点关联的标记等数据，提高最短路径算访问数据的效率。

根据理论推演和实验数据可知，是很小的，复点是节点的比较普遍特性，会迅速降低小步算法和Dijkstra算法的时间复杂度。当邻接边数量增加时，Dijkstra算法对应的会缓慢地增加，小步算法对应的则增加的更为缓慢。在邻接边数量小的图中，后者均值大体上大于前者，随着邻接边数量的增加，前者均值大体上将大于后者。

根据线性回归模型，增加时Dijkstra算法的时间复杂度会较快地上升，小步算法的时间复杂度变化较慢，小步算法比Dijkstra算法具有更好的可扩展性。节点的平均邻接边数量越大，小步算法的性能优势越大。尤其在分布式系统中，小步算法在节省内存空间和计算资源、提高通信效率和降低能源消耗等方面，都具有巨大的价值。

Dijkstra算法与小步算法的时间复杂度公式是一样的。但是小步算法的时间复杂度所对应的主要操作及其高频访问的存储空间的复杂度都显著优于Dijkstra算法。基于分析小步算法和Dijkstra算法的时间复杂度更具有理论和实践的意义。Dijkstra算法中增长率要远大于小步算法，所以理论上，当堆操作成为小步算法的主要操作时，小步算法的性能也会显著优于Dijkstra算法。

小步算法间断性地访问邻接边会降低缓存的命中率，可根据最短路径值设计更加适用于小步算法的缓存管理方法。当|V|非常大时，随着邻接边数量、和单个数据项需要的存储空间的增加，最小堆在运行过程中需要内存空间是不可控的，pop和push操作将会消耗很多计算资源，设计高效的空间复杂度为的堆具有重要意义。

## 8参考文献

[1] E. W. Dijkstra. A note on two problems in connexion with graphs. Numerische Mathematik, pages 269–271, 1959.

[2] R. Bellman. On a routing problem. Quarterly of Applied Mathematics, 1958.

[3] Johnson, Donald B. (1977), "Efficient algorithms for shortest paths in sparse networks", Journal of the ACM, 24 (1): 1–13, doi:10.1145/321992.321993, S2CID 207678246.

[4] Controversial, see Moshe Sniedovich (2006). ["Dijkstra's algorithm revisited: the dynamic programming connexion"](https://www.infona.pl/resource/bwmeta1.element.baztech-article-BAT5-0013-0005/tab/summary). Control and Cybernetics. 35: 599–620. and [below part](https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#Dynamic_programming_perspective).

[5] A. V. Goldberg and C. Harrelson. Computing the Shortest Path: A ∗ Search Meets Graph Theory. In Proc. 16th ACM-SIAM Symposium on Discrete Algorithms, pages 156–165, 2005.

[6] D. Schultes. Fast and Exact Shortest Path Queries Using Highway Hierarchies. Master’s thesis, Department of Computer Science, Universitt des Saarlandes, Germany, 2005

[7] Amgad Madkour, Walid G. Aref, Faizan Ur Rehman, Mohamed Abdur Rahman, Saleh Basalamah. A Survey of Shortest-Path Algorithms. arXiv preprint arXiv:1705.02044, 2017.

[8] Idri A, Oukarfi M, Boulmakoul A, Zeitouni K, Masri A. A new time-dependent shortest path algorithm for multimodal transportation network. Procedia Computer Science. 2017; 109:692–697. [https://doi.org/10. 1016/j.procs.2017.05.379](https://doi.org/10.%201016/j.procs.2017.05.379)

[9] W. Feijen and G. Schäfer, "Dijkstra's algorithm with predictions to solve the single-source many-targets shortest-path problem," arXiv preprint arXiv:2112.11927, 2021.

[10] [Mehlhorn, Kurt](https://en.wikipedia.org/wiki/Kurt_Mehlhorn); [Sanders, Peter](https://en.wikipedia.org/wiki/Peter_Sanders_(computer_scientist)) (2008). ["Chapter 10. Shortest Paths"](http://people.mpi-inf.mpg.de/~mehlhorn/ftp/Toolbox/ShortestPaths.pdf) (PDF). Algorithms and Data Structures: The Basic Toolbox. Springer. [doi](https://en.wikipedia.org/wiki/Doi_(identifier)):[10.1007/978-3-540-77978-0](https://doi.org/10.1007%2F978-3-540-77978-0). [ISBN](https://en.wikipedia.org/wiki/ISBN_(identifier)) [978-3-540-77977-3](https://en.wikipedia.org/wiki/Special:BookSources/978-3-540-77977-3).

[11] U. N. Raghavan, R. Albert, and S. Kumara, “Near linear time algorithm to detect community structures in large-scale networks,” Phys. Rev. E, vol. 76, no. 3, 2007, Art. no. 036106.

[12] J. Leskovec and R. Sosič, "SNAP: A General-Purpose Network Analysis and Graph-Mining Library," ACM Transactions on Intelligent Systems and Technology (TIST), vol. 8, no. 1, p. 1, 2016.

[13] R.A. Rossi, N.K. Ahmed, The network data repository with interactive graph analytics and visualization, 2015, http://networkrepository.com .

[14] Alexandru Iosup et al. “LDBC Graphalytics: A Benchmark for Large-Scale Graph Analysis on Parallel and Distributed Platforms”. In: PVLDB 9.13 (2016), pp. 1–12.

[15] Stats.Blue. Free, Easy-to-Use, Online Statistical Software. <https://stats.blue/>. Accessed 2024-08-08.

[16] <https://zyppy.com/seo/google-index-size/>

[17] <https://www.growth-memo.com/p/googles-index-is-smaller-than-we-think-and-might-not-grow-at-all>

[18] Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). Network Flows: Theory, Algorithms, and Applications. Prentice Hall.

[18] Desmos | 3D Graphing Calculator, <https://www.desmos.com/3d>. Accessed 2024-08-08.

[19] [Fredman, Michael Lawrence](https://en.wikipedia.org/wiki/Michael_Fredman); [Tarjan, Robert E.](https://en.wikipedia.org/wiki/Robert_Tarjan" \o "Robert Tarjan) (July 1987). ["Fibonacci heaps and their uses in improved network optimization algorithms"](http://bioinfo.ict.ac.cn/~dbu/AlgorithmCourses/Lectures/Fibonacci-Heap-Tarjan.pdf) (PDF). [Journal of the Association for Computing Machinery](https://en.wikipedia.org/wiki/Journal_of_the_Association_for_Computing_Machinery). 34 (3): 596–615. [CiteSeerX](https://en.wikipedia.org/wiki/CiteSeerX_(identifier)" \o "CiteSeerX (identifier)) [10.1.1.309.8927](https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.309.8927).

[20] [Stasko, John T.](https://en.wikipedia.org/wiki/John_Stasko" \o "John Stasko)*;*[Vitter, Jeffrey S.](https://en.wikipedia.org/wiki/Jeffrey_Vitter)*(1987),*["Pairing heaps: experiments and analysis"](http://akira.ruc.dk/~keld/teaching/algoritmedesign_f08/Artikler/02/Stasko87.pdf)(PDF)*, Communications of the ACM, 30 (3): 234–249,*[CiteSeerX](https://en.wikipedia.org/wiki/CiteSeerX_(identifier)" \o "CiteSeerX (identifier))[10.1.1.106.2988](https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.106.2988)*,*

[21] [Fredman, Michael L.](https://en.wikipedia.org/wiki/Michael_Fredman" \o "Michael Fredman)*(1999).*["On the efficiency of pairing heaps and related data structures"](https://web.archive.org/web/20110721051135/http:/wwwens.uqac.ca/azinflou/Fichiers840/EfficiencyPairingHeap.pdf)(PDF)*. Journal of the ACM. 46 (4): 473–501.*[doi](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[10.1145/320211.320214](https://doi.org/10.1145%2F320211.320214)*.*[S2CID](https://en.wikipedia.org/wiki/S2CID_(identifier))[16115266](https://api.semanticscholar.org/CorpusID:16115266)*. Archived from*[the original](http://wwwens.uqac.ca/azinflou/Fichiers840/EfficiencyPairingHeap.pdf)(PDF)*on 2011-07-21*. Retrieved 2011-05-03*.*

[22] Pettie, Seth (2005). [Towards a Final Analysis of Pairing Heaps](http://web.eecs.umich.edu/~pettie/papers/focs05.pdf) (PDF). FOCS '05 Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science. pp. 174–183. [CiteSeerX](https://en.wikipedia.org/wiki/CiteSeerX_(identifier)) [10.1.1.549.471](https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.549.471).

[23] M. L. Fredman, R. Sedgewick, D. D. Sleator, and R. E. Tarjan. The pairing heap: a new form of self-adjusting heap. Algorithmica, 1(1):111-129,1986.

[24] 王举范. 一种基于知识图谱最短路径的信息获取方法及装置[发明专利]. ZL 202211058393.2, 2022年08月31日, 2024-02-20.

[25] 王举范. 一种基于最短路径的数据读写优化方法及装置[发明专利]. 2024108672621, 2024 年 06 月 28 日.