无向实权图上单源最短路径的随机算法

段然* [1]，毛佳怡† [1]，舒鑫凯‡ [2]，尹龙辉§ [1]

[1]清华大学跨学科信息科学研究所[2]香港大学

2023年7月11日

摘要

在具有真实非负权的无向图中，我们给出了一种新的随机√算法，用于比较加法模型中运行时间为O(m log n·log log n)的单源最短路径(SSSP)问题。这是第一个利用斐波那奇堆的Dijkstra算法打破实加权稀疏图的O(m + n log n)时间界限的算法。以往的无向非负SSSP算法在比较加法模型中给出的时间界为O(mα(m, n)+min{n log n, n log log r})，其中α为逆ackermann函数，r为最大与最小边权之比[Pettie & Ramachandran 2005]， RAM模型中整数边权的线性时间[Thorup 1999]。请注意，对于无向实数加权SSSP [Pettie & Ramachandran 2005]的基于层次的算法，有一个建议的复杂度下界Ω(m + min{n log n, n log log r})，但我们的算法不符合该下界所需的性质。作为一种非基于层次的方法，我们的算法以更简单的结构显示出了巨大的优势，并且更容易实现。

*duanran@mail.tsinghua.edu.cn †mjy22@mails.tsinghua.edu.cn ‡xkshu@cs.hku.hk

§ylh21@mails.tsinghua.edu.cn

1 介绍

最短路径是图论中最基本的问题之一，其算法位于图算法研究的核心。在一个图G = (V, E, w)， m = |E|， n = |V |，非负边权w: E→$R_{\geqslant 0}$中，单源最短路径(SSSP)问题要求给定源s∈V到所有其他顶点的距离。Dijkstra算法[8]通过动态规划计算距离dist(s, u)。对于每个顶点u，它维护一个时间距离d(u)，它代表从s到u的最短路径，只经过当前s中的顶点，其中s是算法过程中访问过的顶点集合。在每一轮迭代中，它从未访问的节点中选择d(u)最小的顶点u。最后，当S = V, d(u) = dist(S, u)对于所有顶点u.高级数据结构称为斐波那奇堆[14]和松弛堆[10]，它们的插入和减键时间平摊为O(1)，提取最小值时间平摊为O(log n)，使得Dijkstra算法的时间界限为O(m + n log n).这个时间界限属于比较-加法模型，只允许对边权进行比较和加法操作，并将其视为单位时间操作。这是实数输入中最常见的模型。对于无向图，Pettie和Ramachandran[25]在比较加法模型中提出了运行时间为O(mα(m, n) + min{n log n, n log log r})的SSSP算法，其中α为逆ackermann函数，r为任意两个边权的比值。然而，对于无向图和有向图，没有发现比0 (m + n log n)更快的SSSP算法。

Dijkstra算法的一个副产品是根据到s的距离对所有顶点进行排序，但是Ω(n log n)的下界是基于比较的排序算法。研究人员曾经认为，很多图问题都存在这种排序瓶颈，打破这个瓶颈是一个重要而有趣的方向。Yao[7]给出了一个最小spanning √ tree(MST)算法，其运行时间为O(m log log n)，引用了Tarjan未发表的结果O(m log n)。目前MST的最佳结果是随机线性时间算法[22]，确定性O(mα(m, n))时间算法[4]，以及已证明具有最优(但未知)复杂度的确定性算法[24]。在瓶颈路径问题中，我们希望找到最大化其上两个顶点之间的最小最大边权重的路径。Gabow和Tarjan[18]给出了一个O(m log*n)时间算法来解决有向图中s-t瓶颈路径问题，~~后来~~改进为随机化O(mβ(m, n))时间算法[5]。对于有向图中的单源√全目标瓶颈路径问题，段然等人最近有了O(m log n)时间随机化算法的结果[11]。对于单源非递减路径问题，Virginia V.Williams[34]提出了一个时间约束为O(m log log n)的算法。虽然以上结果都是基于比较的，但这些工作中的技术，如局部构造或分治方法，很难适用于最短路径问题。因此，如何打破SSSP的排序瓶颈是一个问题。

1.1 我们的结果

在本文中，我们提出了第一个针对无向实加权图的SSSP算法，打破了排序瓶颈。

定理1。对于边权为w: E→$R_{\geqslant 0}$的无向图G = (V, E, w)，存在一种comparison-addition √ basedLas-Vegas随机化算法，该算法在O(m log n·log log n)时间内解决单源最短路径问题，其结果总是正确的，并且可以高概率地达到这个时间限制。当m = ω(n)、m = O(n log n)时，时间复杂度可以提高到~~O(mr~~ log n ~~+ n log n log l~~og n)。

注意，在[25]for中存在一个(最坏情况下的)下界Ω(m + min{n log n, n log log r})

“基于层次”的无向实加权SSSP算法，但我们的算法是随机的，而不是基于层次的。讨论见备注2。

技术概述。基于dijkstra算法的瓶颈是优先级队列。为此，我们只在优先队列中添加了一小部分顶点。就像许多关于距离预言器或扳手的工作一样，我们对顶点R的子集进行采样，堆只针对R中的顶点，然后我们将其他每个顶点v“打包”到它在R中最近的顶点，称为b(v)。然后将Ball(v)定义为比b(v)更接近v的顶点的集合。由于该算法不知道最短路径上大多数顶点的正确顺序，因此像Dijkstra算法那样只放松邻居是行不通的。因此，当从堆中弹出一个顶点u∈R时，我们也处理捆绑到u的顶点v。在无向图中，这也意味着|dist(s, u)−dist(s, v)|并不大。这里我们从球(v)中的顶点和它们的邻居放松v，然后从v中我们放松v和它们球中的顶点的邻居。(为了便于描述，我们首先将图更改为具有O(m)个顶点的常数度图。)算法的细节将在第3节讨论，以及正确性和运行时间的分析。为了使算法能够达到时间限制w.h.p, bundle的详细构造将在第4节中介绍。通过放松常数度约束来提高时间复杂度的问题将在第5节讨论。

1.2 其他相关工作

对于实权SSSP算法，是否存在优于O(m + n log n)的算法早已被公开。Pettie和Ramachandran的算法[25]在最大和最小边权之比不是很大的情况下优于O(n log n)。对于整数加权的情况，通常采用随机存取机(random access machine, RAM)模型，允许对边权进行乘法、移位和布尔运算。在整数权值RAM模型上改进堆和SSSP算法的工作有很多[16,17,19,26,27,29,30]。最后Thorup给出了无向图[28]的线性时间算法和有向图[31]的O(m + n log log min{n, C})，其中C是最大边权重。最近，对于负权重的SSSP，也发现了几乎线性时间为O(m1+ O(1) log C)的算法[2,6]。

全对最短路径(all -pair shortest path, APSP)问题要求图g中每对顶点u, v之间的最短路径。我们可以对所有顶点运行Dijkstra算法[8]，运行时间为O(mn + n^2log n)，也可以使用Floyd-Warshall算法[12,32]，运行时间为O(n^3)。从那时起，研究人员进行了许多改进[3,9,15,20,35]，但对于实加权图，甚至[0,n]中的graphs $_{\sqrt{}}$ with整数权值，对于某些常数ω > 0) APSP算法仍然没有真正的亚立方时间(O($n^{3-\boxtimes}$)。Williams[33]给出了一种实数加权图的APSP算法，运行时间为$n^3/2^{\Theta(\log n)}$。对于无向实权图，Pettie和Ramachan-dran的APSP算法[25]的运行时间为O(mn log α(m, n))，对于有向实权图，Pettie[23]给出了O(mn + n^2log log n)时间的APSP算法。

2 预赛

本文研究了一个无向图G = (V, E, w)，其顶点集V，边集E≠v2，非负权函数w: E→$R_{\geqslant 0}$，记为w_{uv}。在一个无向图中w_{uv}=w_{vu}对所有边(u, v)∈E成立。我们用n = |V |， m = |E|表示图中顶点和边的数量，n (u) = {V: (u, V)∈E}表示u的邻居，对于两个顶点u, V∈V，$dist_G$(u，V)是连接u和V的最短路径的长度，即图G中u和V的距离，背景清楚时可以省略下标G。设s为源点。

我们算法的目标是为每个v∈v找到dist(s, v)。在不失一般性的前提下，我们假设G是连通的，因此m⩾n−1。
Constant-Degree图。在整篇论文中，我们需要一个常数度的图。为了实现这一点，给定一个图G，我们通过一个经典变换构造G '(见[13]):

. Substitute每个具有|N(v)循环的顶点v |顶点x_{vw}(w∈N(v))与零权边连接，即对于v的每个邻居w，在这个循环上都有一个顶点x_{vw}。

· 对于G中的每条边(u, v)，在对应的顶点x_{uv}和x_{vu}之间添加一条权值为w_{uv}的无向边。

我们可以看到任意u '∈N(u)和v '∈N(v)的距离$dist_{G'}(x_{uu'}, x_{vv'})=dist_G(u, v)$。G’中的每个顶点的度数最多为3，而G’是一个顶点为O(m)、边为O(m)的图。

Comparison-Addition模型。在本文中，我们的算法在比较-加法模型下工作，在该模型中，实数只受比较和加法操作的影响。在这个模型中，每次加法和比较都需要单位时间，并且不允许对边的权重进行其他计算。

斐波那契堆。在这样的模型下，可以构造一个斐波那契堆H，其初始化、插入、减键操作的平摊时间为O(1)，每个最小提取操作的平摊时间为O(log |H|)[14]。当我们从堆中提取最小元素时，我们也称该元素为从堆中“弹出”。

3 主要算法

在接下来的几节中，我们假设G是连通的，G中的每个顶点的度都不超过3。(G中有O(m)个顶点和O(m)条边，但我们仍然使用O(log n)，它等价于O(log m)，其中n是原始图中没有度约束的顶点数量。)

我们的算法是基于原始的Dijkstra算法[8]。由于Dijk-stra算法的主要瓶颈是从斐波那契堆中提取每个顶点的时间为O(log n)[14]，因此我们只将R个顶点的子集插入到堆中。每个剩余的顶点v∈v \ R被捆绑到它在R中最近的顶点上。在整个算法中，只有当某个顶点u∈R从堆中弹出时，顶点才会被更新。我们的算法包括两个阶段:bundle构造和bundle Dijkstra，其细节将分别在3.1节和3.2节中介绍。
give To √ 演示我们算法的主要思想，在本节中我们首先给出一个算法，其运行时间预期为O(m log n·log log n)，但不是“具有高概率”。在第4节中，我们对bundle构建阶段进行了改进，从而得到了一个~~以高概率运行~~时间为O(m log n·log log n)的算法。这两种算法总是给出正确的答案。

3.1 包施工

bundle construction的简单版本如下1 (k是后面要确定的参数):

[1]One may notice that sampled set, closest sampled vertex and balls are common techniques in papers on shortest path algorithms, distance oracles and spanners, and there are deterministic construction algorithms for such “domi-nating set” (e. g. [1]), but the extra O(log n) factor for deterministic approaches introduced on the size of dominating set or construction time is not affordable here.

- 以概率$\frac{1}{k}$独立采样每个顶点$v \in V \setminus \{s\}$形成集合R，然后加上s 在R。
- For每个顶点$v \notin R$，运行从v开始的Dijkstra算法，直到从堆中提取R的第一个顶点，记为b(v)。因此b(v)是R中离v最近的顶点之一，即$b(v) \in \arg\min_{u \in R} dist(u, v)$，我们称v与b(v)捆绑在一起。
- 对于每个$u \in R$，设b(u) = u, Bundle(u) = {v: u = b(v)}是捆绑到u的顶点集合。根据定义，$\{Bundle(u)\}^{u \in R}$形成了顶点集合v的一个划分。
- 对于每个$\notin R$的顶点v，定义Ball(v) = {$w \in V$: dist(v, w) < dist(v, b(v))}，即比其捆绑的顶点b(v)更接近v的顶点集合。在之前的Dijkstra算法中，我们可以得到Ball(v)以及对于所有$w \in Ball(v) \cup \{b(v)\}$的dist(v, w)的值。

Bundle构建的时间分析。对于每个顶点$v \notin R$，在不丧失一般性的前提下，我们假设它的Dijkstra算法以确定性的方式打破束缚。因此，从堆中提取的顶点顺序是固定的。

我们可以看到E[|R|] = O(m/k)。对于每个顶点$v \notin R$，设S_v为Dijkstra算法停止前提取的顶点集，则Ball(v)(S_v。根据R的定义，$|S_v|$遵循几何分布，成功概率$\frac{1}{k}$，因此$E[|S_v|] = k$, $E[|Ball(v)|] \leqslant k$。根据常数度性质，曾经添加到堆中的顶点数也是$O(|S_v|)$，因此，的总时间

备注1。我们可能会注意到，x log x是一个凸函数，因此E[|Sv| log |Sv|] = O(k log k)并不成立。我们在这里给出一个简单的证明:(通过几何分布$E[|S_v|^2] = 2k^2 - k$)

束结构的期望值为$O(\sum_{v \in V \setminus R} E[|S_v| \log |S_v|]) = O(mk \log k)$。

$$
\begin{aligned}
\mathbb{E}[|S_v| \log |S_v|] &= \sum_{n=1}^{\infty} \frac{1}{k}\left(1-\frac{1}{k}\right)^{n-1} \cdot n \log n \\
&\le \sum_{n \le k^2} \frac{1}{k}\left(1-\frac{1}{k}\right)^{n-1} \cdot n \log n + \sum_{n > k^2} \frac{1}{k}\left(1-\frac{1}{k}\right)^{n-1} \cdot n^2 \\
&\le 2 \log k \sum_{n \le k^2} \frac{1}{k}\left(1-\frac{1}{k}\right)^{n-1} \cdot n + \sum_{n=1}^{\infty} \frac{1}{k}\left(1-\frac{1}{k}\right)^{k^2+(n-1)} (n+k^2)^2 \\
&\le 2 \log k \cdot \mathbb{E}[|S_v|] + \left(1-\frac{1}{k}\right)^{k^2} \cdot \mathbb{E}[(|S_v| + k^2)^2] \\
&\le 2k \log k + e^{-k} \cdot O(k^4) = O(k \log k).
\end{aligned}
$$

3.2 Bundle Dijkstra

给定集合R和bundle的划分，主要算法工作如下，伪代码在算法1中给出:

最初我们为所有其他顶点v设置d(s) = 0和d(v) = +∞，并将R的所有顶点插入斐波那契堆[14]中。每当我们从堆中弹出一个顶点$u \in R$时，我们通过以下步骤更新距离。(这里用值D来放松顶点v意味着我们用min{D (v)， D}来更新D (v)。)

1. 对于捆绑到u的每个顶点v，我们需要找到dist(s, v)的确切值，首先用d(u) + dist(u, v)放松v;然后对于每个顶点$y \in Ball(v)$，用d(y) + dist(y, v)松弛v;并且对于每一个

$z_2 \in Ball(v) \cup \{v\}$和$z_1 \in N(z_2)$，对v进行松弛$d(z_1)+w_{z1,z2}+ dist(z_2, v)$，即我们通过其捆绑的顶点u、Ball(v)内的顶点以及与v和Ball(v)相邻的顶点来更新d(v)。

2. 在为每个$x \in Bundle(u)$更新d(x)之后，我们更新顶点$y \in N(x)$和顶点$z_1 \in Ball(y)$。即对所有的$y \in N(x)$，通过$d(x) +w_{x,y}$来松弛y，然后对所有的$z1 \in Ball(y)$，通过d(x) + wx,y + dist(y, z1)来松弛z_1。

3.每当我们更新一个顶点$v \in / R$时，我们也通过d(v) + dist(v, b(v))对其捆绑的顶点b(v)进行松弛。(但稍后我们会看到，这只在步骤2中需要，而不是步骤1，因为在步骤1中v被捆绑到u，但当从堆中弹出u时，dist(s, u)的距离已经找到了。)

从算法中可以很自然地得出以下观察结果。

观察2。$d(v) \geqslant dist(s, v)$对所有$v \in v$始终成立。

Algorithm 1: BundleDijkstra(G, s)

Input : A graph $G = (V, E, w)$ and starting vertex $s \in V$
Output: Distance $d(v)$ from s to v for every vertex $v \in V$

```
Construct Bundles as described in Section 3.1;
Set label d(s) ← 0 and d(v) ← +∞ for all v ∈ V \ {s};
Initialize Fibonacci heap H with all vertices of R and key d(·);
while H is not empty do
    u ← H.ExtractMin();
    foreach v ∈ Bundle(u) do                                   // Step 1
        Relax(v, d(u) + dist(u, v));
        foreach y ∈ Ball(v) do
            Relax(v, d(y) + dist(y, v));
        foreach z2 ∈ Ball(v) ∪ {v} do
            foreach z1 ∈ N(z2) do
                Relax(v, d(z1) + w_{z1,z2} + dist(z2, v));
    foreach x ∈ Bundle(u) do                                   // Step 2
        foreach y ∈ N(x) do
            Relax(y, d(x) + w_{x,y});
            foreach z1 ∈ Ball(y) do
                Relax(z1, d(x) + w_{x,y} + dist(y, z1));
function Relax(v, D):
    if D < d(v) then
        d(v) ← D;
        if v ∈ H then
            H.DecreaseKey(v, D)
        else if v ∉ R then
            Relax(b(v), d(v) + dist(v, b(v)))                  // Step 3
```

束Dijkstra的时间分析。对于Bundle Dijkstra阶段，只有R中的顶点被插入到堆中，因此提取最小操作总共只需要O(|R| log n)时间。由于V \ R中的每个顶点在步骤1和步骤2中分别只以V和x的形式出现一次，并且根据常数度性质，每个顶点在步骤2中以顶点y∈N(x)的形式出现常数次，因此步骤1中每个V的顶点z_1、z_2的数量为O(|球(V)|)，步骤2中每个y的顶点z_1的数量为O(|球(y)|)。另外需要注意的是，由于b(v)∈R，第3步中的递归调用of$_p$Relax只能递归一次。因此Step$_p$1、2、3的总时间为O($_{v\in v\setminus R}$ |Ball(v)|)。因此，束Dijkstra阶段的期望时间为E[O(|R|·log n + |Ball(v)|)] = O($_{m\ v\in v\setminus R\ k}$ log n + mk)。

现在，我们可以看到，两个qstage的期望总时间为O($_{mk\log n + mk}$ log k)，如果我们选择k = log log n，则期望总时间最小化为O(m $\sqrt{\log n \cdot \log\log n}$)。接下来，我们来解释一下我们关于正确性证明的主要思想。3.3节给出了形式化证明。

主要的想法。以下命题在算法中成立。(这里u的迭代指的是pop u∈R时执行的迭代;发现了一个真实距离dist(s, v)意味着d(v) = dist(s, v)已经成立。)

命题3。当从堆中弹出u∈R时，其距离dist(s, u)已经被找到。

命题4。在u的迭代步骤1之后，找到了所有v∈Bundle(u)的dist(s, v)。

下面的引理包含了算法的主要思想。

引理5。对于任意顶点u∈R，任意从s到u的路径P，如果P经过顶点y, dist(s, b(y))最多为P的长度。

证明。dist(s, y)最多为P从s到y的子路径长度，根据b(y)的定义，dist(y, b(y))最多为P从y到u的子路径长度，将两条子路径串联在一起，dist(s, b(y))≤dist(s, y) + dist(y, b(y))最多为P的长度。

□

引理5表明，对于任意顶点u∈R，从s到u的最短路径只包含顶点y且dist(s, b(y))≤dist(s, u)。这就是为什么R的顶点按dist(s，·)递增的顺序出现的直观原因。然而，从s到某个顶点v∈Bundle(u)的最短路径可能会经过某个dist(s, b(y))≥dist(s, u)的顶点y，即b(y)仍然没有从堆中弹出。但令人惊讶的是，根据引理6我们甚至可以在b(y)迭代之前处理这种情况。

引理6。对于一个顶点v∈/ R，如果s到v的最短路径短于dist(s, b(v)) + dist(b(v)，v)，并且它经过一个顶点y(非v)使得dist(s, b(y))≥dist(s, b(v))，那么在y到v的最短路径上有两个相邻的顶点z_1，z_2使得z_1∈Ball(y)∪{y}和z^2∈$^{\text{Ball}(v)}$∪{v}。

证明。我们有dist(y, v) = dist(s, v)−dist(s, y)和dist(s, v) < dist(s, b(v)) + dist(b(v)，v)。通过三角不等式，dist(s, y)≥dist(s, b(y))−dist(y, b(y))，以及通过dist(s, b(y))≥dist(s, b(v))，

$$\mathsf{dist}(y,v) < \mathsf{dist}(\mathsf{b}(v),v) + \mathsf{dist}(y,\mathsf{b}(y)) + \mathsf{dist}(s,\mathsf{b}(v)) - \mathsf{dist}(s,\mathsf{b}(y)) \le \mathsf{dist}(\mathsf{b}(v),v) + \mathsf{dist}(y,\mathsf{b}(y))$$

设z_1是y到v满足dist(y，z_1)< dist(y, b(y))的最短路径上的最后一个顶点，那么z_1∈Ball(y)。那么z_2将是z_1之后的下一个顶点，所以dist(y，z_2)≥dist(y, b(y))，并且dist(z_2，v) < dist(v, b(v))，所以z_2∈Ball(v)。(如果dist(y, b(y)) = 0则z_1= y，如果dist(y, v) < dist(y, b(y))则z2 = v且z1为v之前的顶点)

□

然后我们可以看到命题3和命题4在整个算法中迭代地成立:(形式证明将在3.3节给出)

· 当我们从堆中弹出源s时，d(s) = 0，所有v∈Bundle(s)的距离dist(s, v)在Bundle构建步骤中可以找到，并且可以在步骤1中放入d(v)。
· 假设命题4对第一个弹出的i个顶点成立，那么所有捆绑到弹出顶点的顶点的真实距离都被找到了。通过第2步和第3步，我们可以看到对于所有未弹出的u∈R，如果s到u的最短路径不经过堆中捆绑到其他未弹出顶点的顶点，则可以找到距离dist(s, u)。如果下一个弹出的顶点u’不满足这个条件，则设y是s到u’的最短路径上的第一个顶点，该最短路径被捆绑到u’以外的未弹出的顶点b(y)，因此可以找到dist(s, b(y))。根据引理5,dist(s, b(y))≤dist(s, u ')，所以如果d(u ') > dist(s, u ') b(y)将是下一个弹出的顶点。因此，下一个弹出的顶点u '的dist(s, u ')在它被弹出之前就被找到了。

. If一个未弹出的顶点u '∈R在弹出顶点u的迭代中被更新，到u '的新路径必须经过Bundle(u)中的一个顶点。根据引理5,d(u ')不能更新为小于dist(s, u)，所以未弹出的顶点必须比任何弹出的顶点具有更长或相等的距离。

· 因此，当弹出一个顶点u∈R时，它的距离dist(s, u)已经被发现。对于所有顶点v∈Bundle(u)，如果dist(s, v)不是由d(u) + dist(u, v)直接得到的，即dist(s, v) < dist(s, u) + dist(u, v)，则设x为s到v最短路径上的最后一个顶点，使得b(x)在u之前出现，y为x之后的下一个顶点。我们可以看到dist(s, b(y))≥dist(s, u)，因此根据引理6，我们得到z_1和z_2。那么从命题4可以在b(x)迭代的第1步中找到dist(s, x)，那么在该迭代的第2步中可以找到dist(s，z_1))。在u的这次迭代中，dist(s, v)可以在步骤1中设置为dist(s，z_1)+$w_{z1,z2}$+dist(z_2，v)，那么命题4在这次迭代后仍然成立。

3.3 正确性证明

我们基于算法1的伪代码给出了一个形式化证明。定义u_i∈R为算法1中while-loop第i次迭代提取的顶点。我们下面的关键引理显示了算法的主要属性，因此无论R如何选择，束Dijkstra都是正确的。

引理7。对于束Dijkstra(算法1)中任意i≥1，以下属性成立:

1. 当u_i从堆中提取时，d(u_i)= dist(s，u_i)保持。
2. 在while循环的第i次迭代后，对于所有u∈R\ {uj}j≤i, d(u)≥d(ui)。

3.while循环第i次迭代步骤1后，对于所有v∈Bundle(ui)， d(v) = dist(s, v)。

证明。我们将在i上用归纳法证明这个引理。

7.这个引理对于i = 1成立，因为d(s) = 0, d(v) = dist(s, v)对于所有v∈Bundle(s)在Line之后

假设这个引理对每一个i≤t−1都成立，考虑i = t的情况。

1. 考虑一条从s到u_t的最短路径P。设x是P上的最后一个顶点，使得x∈Bundle(u_j)对于某些j < t, y是x之后的下一个顶点，因此对于某些u∈R\{u_ℓ}$_{\ell<t}$，y∈Bundle(u)。根据归纳假设d(x) = dist(s, x)在第j次迭代的步骤1后的性质3。之后，算法更新d(y)，并在第15行进一步更新d(u)，因为y∈N(x)。

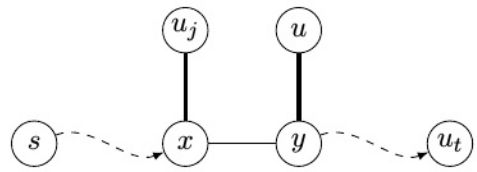

图1:从s到u_t的最短路径示意图。

因此在(t−1)-th迭代后d(y) = dist(s, x) + dist(x, y) = dist(s, y)和d(u)≤dist(s, y) + dist(y, u)。进一步:

$$\begin{aligned} d(u) &\le \mathsf{dist}(s,y) + \mathsf{dist}(y,u) \\ &\le \mathsf{dist}(s,y) + \mathsf{dist}(y,u_t) && (\mathsf{b}(y)=u) \\ &= \mathsf{dist}(s,u_t) && (y \text{ on shortest path}) \\ &\le d(u_t). && (\text{Observation } 2) \end{aligned}$$

另一方面，算法立即从斐波那契堆H中提取出

(t−1)次迭代，因此$d(u_t)$≤d(u)。所以上面所有的不等式都应该是方程，因此

$d(u_t)$= dist(s，u_t)。

2. 当执行第t次迭代的第5行时，d(u)≥$d(u_t)$对每个u∈R\$\{u_j\}_{j\leqslant t}$自H以来保持

是一个斐波那契堆。假设d(u) <$d(u_t)$对于某些u∈R\$\{u_j\}_{j\leqslant t}$，在第t次迭代后。的

d(u)的进一步更新必须从d(x)开始，针对某些x∈Bundle(u_t)。对于最后这样的更新，

在这条从s到x再到u的路径上应用引理5，我们有:

$$\begin{aligned} d(u) &\ge \mathsf{dist}(s,u_t) && (\text{Lemma } 5) \\ &= d(u_t), && (\text{Property } 1) \end{aligned}$$

导致矛盾。

3.我们想要证明d(v) = dist(s, v)对所有v都成立∈Bundle(u_t)。假设在步骤1之后存在一个顶点v∈Bundle(u_t)，使得d(v) > dist(s, v)。设x是P上的最后一个顶点，对于某个j < t, x∈Bundle(u_j)，y是P上x之后的下一个顶点，因此对于某个u∈R, y∈Bundle(u) \$\{u_\ell\}_{\ell<t}$。根据归纳假设的性质3，在第j次迭代的步骤1之后，d(x) = dist(s, x)。与上文相同，我们可以证明，在t-th迭代之前，d(y) = dist(s, x) + dist(x, y) = dist(s, y)且d(u)≤dist(s, y) + dist(y, u)(其中u = b(y))。我们有:

$$\begin{aligned} \mathsf{dist}(s,y) &\ge d(u) - \mathsf{dist}(y,u). \\ \mathsf{dist}(s,v) &< d(v) && (\text{Assumption}) \\ &\le d(u_t) + \mathsf{dist}(u_t,v). && (d(v) \text{ updated in Line } 7) \end{aligned}$$

另一方面，d(u)≥$d(u_t)$经过性质2的第t次迭代后。由于$d(u_t)$不会改变(性质1和观察2)，而且d(u)只能在第t次迭代时减少，所以d(u)≥$d(u_t)$在整个第t次迭代中都成立。因此:

$$\mathsf{dist}(y,v) = \mathsf{dist}(s,v) - \mathsf{dist}(s,y) < \mathsf{dist}(u_t,v) + \mathsf{dist}(y,u), \tag{1}$$

而方程成立，因为y位于s到v的最短路径上。

因此有两种可能的情况:

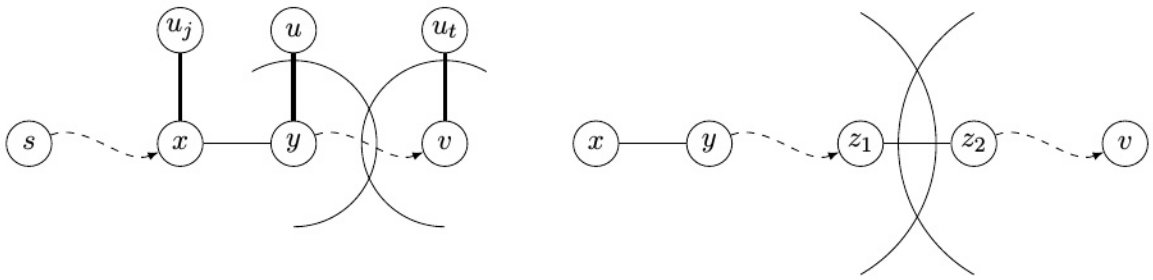

图2:左:如果d(v) > dist(s, v)从s到v的最短路径图解，右:如果dist(y, v)≥dist(ut, v)仔细观察路径x到v。

- dist(y, v) < dist(ut, v)。
  在这种情况下，y∈Ball(v)，因此我们可以在第9行将d(v)更新为dist(s, y) + dist(y, v)，与d(v) > dist(s, v)相反。
- dist(y，v)≥dist(u_t，v)。
  首先，由不等式(1)，dist(y, u) > dist(y, v)−dist(u_t，v)≥0。设z_1为路径P上dist(y，z_1)< dist(y, u)，则有z_1∈Ball(y)。设z_2是路径上的下一个顶点，则dist(y，z_2)≥dist(y, u)，因此dist(z_2, v) = dist(y, v)−dist(y，z_2)≤dist(y, v)−dist(y, u) <dist(u_t，v)，即z_2∈Ball(v)。(如果z_2不存在，则z_1= v)根据归纳假设的性质3,d(x)在第j次迭代的步骤1之后= dist(s, x)，因此d(z_1)在第j次迭代的第17行更新为dist(s，z_1))。因此，d(v)在第t次迭代的第12行更新为dist(s, v)(因为j < t)，与假设相矛盾。

因此，在第t次迭代的步骤1之后，对于所有v∈Bundle(ut)， d(v) = dist(s, v)。

□

备注2。Pettie和Ramachandran[25]证明，在比较-加法模型中，无向图上任何基于层次结构的SSSP算法至少需要Ω(m+min{n log log r, n log n})时间，其中r为边权重的最大值与最小值之比。当r呈指数级增大时，这个界限就变成了Ω(m+n log n)。在这里，基于层次结构的算法被定义为生成一个满足层次性质的排列π_s:dist(s, v)≥dist(s, u) + sep(u, v)⇒π_s(u)<π_s(v)，其中sep(u, v)是u和v之间的MST路径上最长的边。排列π_s，尽管没有被定义为，通常对于[25]中讨论的算法来说，是算法访问顶点的顺序。然而，这是一个最坏情况下的下界，我们的算法是随机的。还订单,我们的算法访问顶点不遵循等级属性:想到两个顶点x和y都是通过边连接到u (x, u), (y, u)这两个重量1,x, y都是捆绑你。有可能9月(x, y) = 1和dist(年代,y) = dist(年代,x) + 2但我们不限制订单我们访问x和y,它是可能的我们之前访问y x。这就解释了为什么我们的算法可以打破这个Ω(m + n o (log n))下界[25]。

4 改进的束结构

在本节中，我们提出了一种改进的bundle构造，其运行时间为O(m √log n·log log n)，~~且具有高概率。~~在3.3节中，我们证明了Bundle Dijkstra的正确性不依赖于R的选择，只要b(·)，Ball(·)和Bundle(·)被正确计算

对于r，构建bundle的运行时间为O(Dijkstra是O(Pv∈V \R |Ball(V)| + |R| log n)， Pv∈V \R |Sv | log |Sv)， Bundle

自然，$|S_v|$是每个顶点v∈v服从几何分布的随机变量，并且它们不是独立的，因为一个顶点x∈v可能出现在几个集合中。但是，对于一个子集W⊆V，如果任何一个顶点在$\{S_v\}_{v\in W}$中最多出现一次，则对应的随机变量$\{|S_v|\}_{v\in W}$是独立的。由引理9可知，如果每个随机变量都依赖于少数其他变量，那么它们的和就会以指数小概率偏离期望。因此，我们手动包含所有那些$|S_v|\geqslant k \log k$到R的顶点。这样，对于V \ R中的每个顶点，它的随机变量只依赖于有限数量的其他顶点，我们可以高概率地约束它们的和。

下面我们将介绍如何生成R并计算$\{b(v)\}_{v\in V\setminus R}$，以及$\{Ball(v)\}_{v\in V\setminus R}$， $\{Bundle(u)\}u\in R$和$dist(v,_q u)$对于u∈Ball(v)∪{b(v)}。伪代码在算法2中给出。我们还是像第3节那样设置参数$k = {}^{\log n} \log \log n$。

I改进了束结构。

· 以概率$_{1\ k}$对每个顶点v∈v \ {s}进行采样，形成集合R1，并将s添加到R1中;

. For每个v∈v \ R_1，从v开始运行Dijkstra算法，直到我们在R_1;中提取了一个顶点或已经弹出了k log k个顶点。

· 在前一种情况下，将提取的顶点按它们出现的顺序表示为列表Vextract(v)。注意$V_{extract}{}^{(v)}$is类似于第3节的S_v。

在后一种情况下，在R_2;中添加v

. SetR = R_1∪R_2，对于v∈v \ R，设$V_{extract}{}^{(v)}$that中位于R中的第一个顶点为b(v);

. With上面的结果，计算u∈R的Bundle(u)， v∈v \ R的Ball(v)，并记录u∈Ball(v)∪{b(v)}的dist(v, u)。这一步的时间是线性的。

pthis束构造的正确性遵循Dijkstra算法[8]。我们只需要分析|R|， $_{v\in v\setminus R}$ |Ball(v)|及其运行时间。这种改进的束结构的性能在引理8pbelow中得到表征。根据引理8，束的构造需要O(mk log k)时间，束Dijkstra需要O($_{v\in v\setminus R}$ |Ball(v)| + |R| log n) = O(mk + m log n/k)，概率为$1\sqrt{-e-\Omega(n^{1-o(1)})}$。因此，我们的算法的总运行时间为O(mk log k + m log n/k) = O(m log n·log log n) w.h.p。引理8的证明是基于引理9。

引理8。通过运行算法2，概率为$1-e-\Omega(n^{1-o(1)})$，得到以下属性:

(a) P|R| = O(mk)。(b) $_{v\in v\setminus R}$ |球(v)| = O(mk)。(c)算法2的运行时间为O (mk log k)。

证明。首先，V \ {s}的每个顶点以概率$_{k1}$独立地插入到R_1中，因此通过切尔诺夫界，概率为1−O(e−m/k) = $1-e-\Omega(n^{1-O(1)})$， |R1| = Θ(m/k)，同时m ':= |V \ R1| = Θ(m)。

对于每个顶点v∈v \ R_1，定义X_v= I[v∈R_2]]和Y_v=$V_{extract}{}^{(v)}$，则X_v为伯努利随机变量，Xv∈[0,1]概率为1,E[Xv] =(1−k1)k $_{\log k}$ = Θ(k1)。Yv是a

Algorithm 2: BUNDLECONSTRUCTION(G, s, k)

Input : A graph $G = (V, E, w)$, the source vertex $s \in V$ and parameter k
Output: All R, $\mathsf{b}(\cdot)$, $\mathsf{Ball}(\cdot)$, $\mathsf{Bundle}(\cdot)$ as described above
Initialize $R_1 \leftarrow \{s\}$, and insert each $v \in V \backslash \{s\}$ to R_1 independently with probability $\frac{1}{k}$;
Initialize $R_2 \leftarrow \emptyset$;
foreach $v \in V \backslash R_1$ **do** `// Truncated Dijkstra`
  Initialize Fibonacci heap $H^{(v)}$ with vertex v and its key $d^{(v)}(v) \leftarrow 0$;
  Initialize an ordered list $V^{(v)}_{extract} \leftarrow ()$;
  while $H^{(v)}$ *is not empty* **do**
    $u \leftarrow H.\text{EXTRACTMIN}()$;
    $V^{(v)}_{extract}.\text{APPEND}(u)$;
    if $u \in R_1$ **then** quit the **while-loop**;
    else if $\left|V^{(v)}_{extract}\right| > k \log k$ **then** set $R_2 \leftarrow R_2 \cup \{v\}$ and quit the **while-loop**;
    else
      foreach $x \in N(u)$ **do**
        if $x \notin H^{(v)}$ *and* $x \notin V^{(v)}_{extract}$ **then** $H^{(v)}.\text{INSERT}(x, d^{(v)}(u) + w_{ux})$;
        else if $d^{(v)}(x) > d^{(v)}(u) + w_{ux}$ **then** $H.\text{DECREASEKEY}(x, d^{(v)}(u) + w_{ux})$;
$R \leftarrow R_1 \cup R_2$;
foreach $v \in V \setminus R$ **do**
  $\mathsf{b}(v) \leftarrow$ the first vertex in the ordered list $V^{(v)}_{extract}$ that lies in R;
Compute $\{\mathsf{Bundle}(u)\}_{u \in R}$, $\{\mathsf{Ball}(v)\}_{v \in V \backslash R}$ and record $\mathsf{dist}(v, u)$ for $u \in \mathsf{Ball}(v) \cup \{\mathsf{b}(v)\}$;

几何随机变量除了其值在k log k处截断，$Y_v \in [0, k \log k]$概率为1且$E[Y_v] = k(1-(1-{}_{k^1})k \log k) = \Theta(k)$。

对于每个顶点$v \in v \setminus R_1$，表示$V_{full}^{(v)}$as在Dijkstra算法中提取的前k log k个顶点，如果它没有截断。它们是由G的结构决定的，因此在Vfull中不存在随机-(v)性。Xv和Yv的值取决于Vfull (v)中的顶点是否插入到R1中。因此，如果Vfull(v1)、Vfull(v2)、⋯、Vfull(vj)是不相交的，则Xv1、Xv2、⋯、Xvj是独立的，同样，Y_{v1}、Y_{v2}、⋯、Y_{vj}也是独立的。

对于每个顶点$w \in V_{full}^{(v)}$，因为w是由v在Dijkstra算法的k log k步内找到的，所以从v到w的路径必须不超过k log k条边，所以根据常次性质，最多有$3 \cdot (1 + 2 + \cdots + 2k \log k - 1) \leqslant 3 \cdot 2k \log k$个不同的u，使得$w \in V_{full(u)}$。因此，对于每一个v，最多有$3k \log k \cdot 2k \log k = O(no(1))$个不同的$u \in v \setminus R1$使得Vfull (v)∩v full (u)6=∅。

为了应用引理9，对于每个$v \in R_1$，也定义X_v，Y_v和$V_{full}^{(v)}$in，就像在算法2的循环中执行v一样。

现在，我们对$\{X_v\}_{v \in V}$和$\{Y_v\}_{v \in V}$应用引理9。对于$\{X_v\}_{v \in V}$，S是V，$|V| = m$，$\mu = \Theta({}_k^1)$，$b = 1$, $T = O(no(1))$，$\{W_v\}_{v \in V}$是$\{V_{full}\}_{v \in V}$，我们可以验证$8Tb\,\mu^{-1} = O(no(1))$ (V)

且$8b3T/\mu^3 = O(no(1))$，因此至少有$1 - e^{-O(m/n^{o(1)})}$的概率，$\sum_{v \in S} X_v = O(m/k)$，而对于$\{Y_v\}_{v \in V}$，$\{V_{full}\}_{v \in V}$，同样，我们以$1 - e^{-\Omega(m/n)}$的概率推断，$\sum_{v \in S} Y_v = O(m/k)$。而对于$\{Y_v\}_{v \in V}$，S为V，$\mu = \Theta(k)$，$b = k \log k$, $T = O(n^{o(1)})$，$\{W_v\}_{v \in V}$也为(V)

PΘ(可)。$_{v\in V\setminus R1}$因此，Yv = Θwe (mk)。得出概率为1−e−Ω($n_{1-o(1)}$)， Pv∈V $_{\setminus R1}$ Xv = Θ(m/k)

然后，我们证明了这个引理的三个结论。

对于(a)，by$_P$定义|R|=$_m$|R1|+ |R2|，因此由并界，以概率1−e−Ω($n_{1-o(1)}$)， |R| = |R1| + $_{v\in V\setminus R1}$ Xv = $o_{(k)}$。

对于(b)，由定义|球(v)|≤Y_v，所以 1−e−Ω($n_{1-o(1)}$)，Pv∈V $_{\setminus R}$ |Ball(V)| = O(mk)。 Pv∈V $_{\setminus R}$ |Ball(V)|≤Pv∈V $_{\setminus R1}$ Yv。因此，有概率

对于(c)，我们计算截断Dijkstra算法在所有迭代中的总时间。对于每个顶点v∈v \ R_1，根据常数度性质，插入操作的次数是O(Y_v)，因此H $_{(v)}$ =O(|Y_v|)= O(k log k)，因此，每次ExctractMin操作需要时间O(log(Y_v))= O(log k)，其他操作需要常数时间。因此截断的Dijkstra算法vtakes$_P$time O(Yv log k).因此，在概率为1−e−Ω($n_{1-O(1)}$)的情况下，算法2的总时间为O($_{v\in V\setminus R1}$ Yv log k) = O(mk log k)。 □

引理9。(类似的参数如[21])一组随机变量{Z_v}$_{v\in S}$满足假设为每个v∈年代,E[Z_v]=μZ_v∈(0,b)的概率1,并且每个Z_v对应于一组固定的确定性W_v,如果W_{v1},W_{v2},⋯,W_{vj}分离,然后$Z_{v1}Z_{v2}$,⋯,Z_{vj}独立,W_v相交最多TW_u不同。

$_{\mu 3\,|S|P}$则在概率至少为1−8Tbμ$_{-1}$·e−8b3T的情况下，成立$_{v\in S}$ Zv = Θ(|S|μ)。

证明。我们尝试将{Z_v}$_{v\in S}$划分为几个子集{Z_t}，使得每个Z_t中的所有Z_v’s都是独立的，这样我们就可以应用Hoeffding不等式，或者Z_t的大小很小，这样我们就可以用上界b来约束它们，最后用联合界来组合所有的东西。还请注意，我们不需要实际计算{Z_t}，因为它们只是为了这个数学证明而引入的。固定参数p=$_{\frac{1}{4}|S|}$4Tbμ。由于每个W_v与最多T个不同的其他W_u相交，每当{Z_v}$_{v\in S}$中至少有p(T + 1)个元素时，我们可以从W_v不相交的元素中选择p个不同的Z_v，使它们独立:任意选择一个Z_v，丢弃那些Z_u如果W_u∩W_v6=∅;由于最多有T个这样的Z_u不同于Z_v，所以每次我们最多舍弃T + 1个元素。所以从p(T + 1)个元素中我们可以选出p个。我们让它们形成Z_t，并将它们从{Zv}v∈S中移除。重复此过程，我们将得到{Zv}v∈S的分区{Z1, Z2，⋯，Zq, Zq+1}，使得:|Zt| = p，且所有Zv∈Zt对于1≤t≤q是独立的;|Zq+1|≤p(T +1)≤2pT = $_{2b\mu 1}$ ||。

根据定义μ≤b，所以|Z^{q+1}|≤$_{\frac{1}{2}}$|S|。

然后根据Hoeffding不等式，对于每一个1≤t≤q,

$$\Pr\left[\left|\sum_{v\in\mathcal{Z}_t} Z_v - |\mathcal{Z}_t|\,\mu\right| > \frac{1}{2}\,|\mathcal{Z}_t|\,\mu\right] \le 2e^{-\frac{2(\frac{1}{2}|\mathcal{Z}_t|\mu)^2}{|\mathcal{Z}_t|b^2}} = 2e^{-\frac{\mu^2 p}{2b^2}}.$$

且0≤$_{\frac{1}{2}\mu}$ By union bound，概率至少为1−2qe−2b2,

$$\sum_{v\in S} Z_v \ge \frac{1}{2}\sum_{t=1}^{q}|\mathcal{Z}_t|\,\mu = \frac{1}{2}(|S| - |\mathcal{Z}_{q+1}|)\mu \ge \frac{1}{2}\left(|S| - \frac{1}{2}\,|S|\right)\mu = \frac{1}{4}\,|S|\,\mu,$$

与此同时

$$\sum_{v\in S} Z_v \le \frac{3}{2}\sum_{t=1}^{q}|\mathcal{Z}_t|\,\mu + |\mathcal{Z}_{q+1}|\,b \le \frac{3}{2}\,|S|\,\mu + \frac{\mu}{2b}\,|S|\cdot b = 2\,|S|\,\mu.$$

由|S|≥Pqt=1 3|Zt|≥qp得出q≤|S| /p = 4Tb/μ。因此，概率至少为1−8Tbμ−1 e−μ3|S| 8b T，认为Pv∈S Zv = Θ(|S|μ)。□

5 讨论

我们感谢一位匿名审稿人提出常数度不是这个算法的必要条件，因此我们可以在m = ω(n)和m = o(n log n)的情况下得到时间复杂度的改进，我们没有制作3次图，而是用类似的方法将>次m/n的顶点分割为≤m/n的顶点，这样顶点的数量仍然是o(n)。然后，在每一步中:

. Inbundle构造时，对每个顶点v进行Dijkstra搜索的时间将变成O(|S_v|·m_n+|Sv| log(|Sv|·mn))，因为堆的大小最多为|S |·m v n，所以总共O(mk+nk log(mk/n))。

. TheBundle Dijkstra的时间将变成O(nk log n + mk)，因为步骤1中每个v的顶点数z_2是O(mn | Ball(v)|)，步骤2中每个y的顶点数z_1是O(|Ball(y)|)，但每个顶点在步骤2中出现的次数是O(m/n)次。

· 当m/n = o(log n)时，可以检查第4节中的独立性分析

(v) (u)

可以，因为不同u∈V \ R_1每个V有V_{full}∩V_{full}6=∅的个数仍然是O(no(1))。

因此，qthis算法的时间复杂度为O(nk log n + mk + nk log(mk/n))。当m < nlog log n时，k仍然等于log n，时间界限为O(n√ log nlog log n)，当p nlog log n nlog log n≤m < nlog n时，设k = m log n，时间界限将为O(√ mn log n)。

参考文献

[1] D. Aingworth, C. Chekuri, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). In Proceedings of the Seventh Annual ACM-SIAM Sympo-sium on Discrete Algorithms, SODA ' 96, page 547–553, USA, 1996. Society for Industrial and Applied Mathematics. ISBN 0898713668.

[2] Aaron Bernstein, Danupon Nanongkai, and Christian Wulff-Nilsen. Negative-weight single-source shortest paths in near-linear time. In 2022 IEEE 63rd Annual Symposium on Foun-dations of Computer Science (FOCS), pages 600–611, 2022. doi: 10.1109/FOCS54457.2022. 00063.

[3] Timothy M. Chan. All-pairs shortest paths with real weights in O(n^3/ log n) time. Algorithmica, 50(2):236–243, 2008. doi: 10.1007/s00453-007-9062-1. URL https://doi.org/10.1007/s00453-007-9062-1.

[4] Bernard Chazelle. A minimum spanning tree algorithm with inverse-ackermann type complex-ity. J. ACM, 47(6):1028–1047, nov 2000. ISSN 0004-5411. doi: 10.1145/355541.355562. URL https://doi.org/10.1145/355541.355562.

[5] Shiri Chechik, Haim Kaplan, Mikkel Thorup, Or Zamir, and Uri Zwick. Bottle-neck paths and trees and deterministic graphical games. In Nicolas Ollinger and Heribert Vollmer, editors, STACS, volume 47 of LIPIcs, pages 27:1–27:13. Schloss

Dagstuhl - Leibniz-Zentrum f¨ ur Informatik, 2016. ISBN 978-3-95977-001-9. URL http://dblp.uni-trier.de/db/conf/stacs/stacs2016.html#ChechikKTZZ16.

[6] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In 2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS), pages 612–623, 2022. doi: 10.1109/FOCS54457.2022.00064.

[7] Andrew Chi chih Yao. An O(|E| log log |V |) algorithm for finding min-imum spanning trees. Information Processing Letters, 4(1):21–23, 1975. ISSN 0020-0190. doi: https://doi.org/10.1016/0020-0190(75)90056-3. URL https://www. sciencedirect. com/science/article/pii/0020019075900563.

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

[9] Wlodzimierz Dobosiewicz. A more efficient algorithm for the min-plus multiplication. International Journal of Computer Mathematics, 32(1-2):49–60, 1990. doi: 10.1080/ 00207169008803814. URL https://doi.org/10.1080/00207169008803814.

[10] James R. Driscoll, Harold N. Gabow, Ruth Shrairman, and Robert E. Tarjan. Relaxed heaps: An alternative to fibonacci heaps with applications to parallel computation. Com-mun. ACM, 31(11): 1343–1354, nov 1988. ISSN 0001-0782. doi: 10.1145/50087.50096. URL https://doi.org/10.1145/50087.50096.

[11] Ran Duan, Kaifeng Lyu, Hongxun Wu, and Yuanhang Xie. Single-source bottleneck path algorithm faster than sorting for sparse graphs. CoRR, abs/1808.10658, 2018. URL http://arxiv.org/abs/1808.10658.

[12] Robert W. Floyd. Algorithm 97: Shortest path. Communications of the ACM, 5:345, 1962.

[13] Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees. In Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing, STOC ' 83, page 252–257, New York, NY, USA, 1983. Association for Computing Machinery. ISBN 0897910990. doi: 10.1145/800061.808754. URL https://doi.org/10.1145/800061.808754.

[14] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network opti-mization algorithms. Journal of the ACM, 34(3):596–615, 1987.

[15] Michael L. Fredman. New bounds on the complexity of the shortest path prob-lem. SIAM Journal on Computing, 5(1):83–89, 1976. doi: 10.1137/0205006. URL https://doi.org/10.1137/0205006.

[16] Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. Journal of Computer and System Sciences, 47(3):424–436, 1993. ISSN 0022-0000. doi: https://doi. org/10. 1016/0022-0000(93)90040-4. URL https://www. sciencedirect. com/science/article/pii/0022000093900404.

[17] Michael L. Fredman and Dan E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. Journal of Computer and System Sciences, 48(3):533– 551, 1994. ISSN 0022-0000. doi: https://doi.org/10.1016/S0022-0000(05)80064-9. URL https://www.sciencedirect.com/science/article/pii/S0022000005800649.

[18] Harold N Gabow and Robert E Tarjan. Algorithms for two bottle-neck optimization problems. Journal of Algorithms, 9(3):411−417, 1988. ISSN 0196-6774. doi: https://doi.org/10.1016/0196-6774(88)90031-4. URL https://www.sciencedirect.com/science/article/pii/0196677488900314.

[19] Torben Hagerup. Improved shortest paths on the word ram. In Ugo Montanari, Jos´e D. P. Rolim, and Emo Welzl, editors, Automata, Languages and Programming, pages 61−72, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. ISBN 978-3-540-45022-1.

[20] Yijie Han and Tadao Takaoka. An O(n^3 log log n/ $\log^2$ n) time algorithm for all pairs shortest paths. In Proceedings of the 13th Scandinavian Conference on Algorithm Theory, SWAT' 12, page 131−141, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 9783642311543. doi: 10.1007/ 978-3-642-31155-0 12. URL https://doi.org/10.1007/978-3-642-31155-0_12.

[21] Svante Janson. Large deviations for sums of partly dependent random variables. Random Struct. Algorithms, 24(3):234−248, may 2004. ISSN 1042-9832.

[22] David R. Karger, Philip N. Klein, and Robert E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. J. ACM, 42(2):321−328, mar 1995. ISSN 0004-5411. doi: 10.1145/201019.201022. URL https://doi.org/10.1145/201019.201022.

[23] Seth Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. Theoretical Computer Science, 312(1):47−74, 2004. ISSN 0304-3975. doi: https://doi.org/10.1016/S0304-3975(03)00402-X. URL https://www.sciencedirect.com/science/article/pii/S030439750300402X. Automata, Languages and Programming.

[24] Seth Pettie and Vijaya Ramachandran. An optimal minimum spanning tree algorithm. J. ACM, 49(1):16−34, jan 2002. ISSN 0004-5411. doi: 10.1145/505241.505243. URL https://doi.org/10.1145/505241.505243.

[25] Seth Pettie and Vijaya Ramachandran. A shortest path algorithm for real-weighted undirected graphs. SIAM Journal on Computing, 34(6):1398−1431, 2005. doi: 10.1137/S0097539702419650. URL https://doi.org/10.1137/S0097539702419650.

[26] Rajeev Raman. Priority queues: Small, monotone and trans-dichotomous. In Proceedings of the Fourth Annual European Symposium on Algorithms, ESA' 96, page 121−137, Berlin, Heidelberg, 1996. Springer-Verlag. ISBN 3540616802.

[27] Rajeev Raman. Recent results on the single-source shortest paths problem. SIGACT News, 28(2):81−87, jun 1997. ISSN 0163-5700. doi: 10.1145/261342.261352. URL https://doi.org/10.1145/261342.261352.

[28] Mikkel Thorup. Undirected single-source shortest paths with positive integer weights in linear time. J. ACM, 46(3):362−394, may 1999. ISSN 0004-5411. doi: 10.1145/316542.316548. URL https://doi.org/10.1145/316542.316548.

[29] Mikkel Thorup. On ram priority queues. SIAM Journal on Computing, 30(1):86−109, 2000. doi: 10.1137/S0097539795288246. URL https://doi.org/10.1137/S0097539795288246.

[30] Mikkel Thorup. Floats, integers, and single source shortest paths. J. Algorithms, 35(2):189−201, may 2000. ISSN 0196-6774. doi: 10.1006/jagm.2000.1080. URL https://doi.org/10.1006/jagm.2000.1080.

[31] Mikkel Thorup. Integer priority queues with decrease key in constant time and the sin-gle source shortest paths problem. Journal of Computer and System Sciences, 69(3): 330−353, 2004. ISSN 0022-0000. doi: https://doi.org/10.1016/j.jcss.2004.04.003. URL https://www.sciencedirect.com/science/article/pii/S002200000400042X. Special Is-sue on STOC 2003.

[32] Stephen Warshall. A theorem on boolean matrices. J. ACM, 9(1):11−12, jan 1962. ISSN 0004-5411. doi: 10.1145/321105.321107. URL https://doi.org/10.1145/321105.321107.

[33] Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing, STOC ' 14, page 664−673, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450327107. doi: 10.1145/2591796.2591811. URL https://doi.org/10.1145/2591796.2591811.

[34] Virginia Vassilevska Williams. Nondecreasing paths in a weighted graph or: How to optimally read a train schedule. ACM Trans. Algorithms, 6(4), sep 2010. ISSN 1549-6325. doi: 10.1145/ 1824777.1824790. URL https://doi.org/10.1145/1824777.1824790.

[35] Uri Zwick. A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths. In Algorithms and Computation: 15th International Symposium, ISAAC 2004, Hong Kong, China, December 20-22, 2004. Proceedings, page 921−932, Berlin, Heidel-berg, 2004. Springer-Verlag. ISBN 978-3-540-24131-7. doi: 10.1007/978-3-540-30551-4 78. URL https://doi.org/10.1007/978-3-540-30551-4_78.

-