



浙江工业大学

本科毕业设计论文

# 附件

题目：大规模网络路由的传输线路选择算法研究

作者姓名 李绍晓

指导教师 王辛刚 副教授

专业班级 电信 1302

学 院 信息工程学院

提交日期 2017 年 6 月 11 日

# 目 录

一、文献综述

二、外文翻译

三、开题报告

四、指导教师评语

五、论文评阅人评语

六、答辩记录

七、毕业设计成果演示记录表

八、教师指导记录表

九、毕业设计进程考核表

十、毕业论文的“知网”检测结果

## 一、文献综述

# 大规模网络路由的传输线路选择算法研究综述

姓名 李绍晓 专业班级 电信 1302

**摘要：**大规模网络路由在传输时，需要确定一个最优的传输路径，来达到更好的传输效率。而该问题可引伸为单源最短路径问题。单源最短路径问题一直是计算机科学、运筹学、地理信息科学等学科中的一个经典问题，国内外大量专家学者对此问题进行了深入研究。随着信息时代和智能时代的来临，该问题正处于不断探索和发展的阶段<sup>[1]</sup>。经典的图论与不断发展完善的计算机数据结构，以及算法的有效性，使得新的算法层出不穷。本文简单介绍了大规模网络路由的传输线路选择问题的产生背景，同时将该问题引伸为最短路径问题，阐述该问题的定义、历史发展、研究现状。

**关键词：**最短路径问题、图论、计算机科学、网络路由传输、路由选择

## 1 网络路由的传输问题

### 1.1 问题产生背景

在当今社会，网络无处不在，网络信息通过路由之间的不断转发，传达到使用者处。转发的过程在IP层进行，选择转发到哪个路由通过路由选择表来确定，而路由选择表在确定最佳路径的过程中被初始化和维护。常见的路由选择协议有RIP、OSPF以及IGRP等<sup>[1]</sup>。

而随着网络通信和云计算的不断发展，企业对于网络路由的选路策略愈发重视。当确定起始点和目标点之后，需要通过上述介绍的路由转发过程到达目标点。当传输网络中的路由数达到几万个时，为了减小用户时延，对选路策略的速度要求会非常高。同时因为各种条件的限制，需要在算法中考虑各种特殊情况。故国内外公司投入了很大的精力，来研发更合适更高效的网络转发算法。

## 1.2 相关问题定义

而该问题通过计算机图论的建模后，可引伸为单源最短路径问题。路由代表图中的顶点，路由的“下一个路由”理解为顶点的边，起点代表起始路由，终点代表终止路由。

该问题需要建模一个图  $G = (V, E, W)$ ，其中  $V$  为节点集合， $E$  为边集合， $W$  为边的权重集合。 $V$  中节点数量为  $n$ ， $E$  中边数为  $m$ 。给出起点  $s$  和终点  $t$ ，通过具体算法，在图  $G$  中求出从  $s$  点到  $t$  点的最短路径及最短路径长度，故该问题被称为最短路径问题（SSSPP, Single Source Shortest Path Problem）。

该问题是计算机科学理论中的一个重要问题，也是数学领域图论中的一个分支。

## 2 相关问题的国内外研究现状

### 2.1 国外研究历史和现状

在 1959 年，Dijkstra E.W.A 提出了著名的 Dijkstra 算法<sup>[2]</sup>，这是最经典的单源点最短路径算法，用于计算一个节点到其他所有节点的最短路径。主要思想是利用思想，通过起始点逐步层层拓展，通过维持“当前最短”状态，来获取起始点到各点的最短距离。该算法作为介绍图结构的经典算法，被广泛用于各计算机类教材中。

然而通过观察 Dijkstra 算法的搜索过程，学者发现搜索过程过慢，尤其是在大规模顶点的图中去搜寻单源最短路径时，需访问大量无关顶点，耗费不必要的时间和空间。故在 1967 年，Doran 提出了 A\*搜索。该算法起初是用来加速游戏地图的搜索，也被称作启发式搜索<sup>[4]</sup>。其主要思想是通过广度搜索算法搜索到达终止点的路径，但是在搜索中加入估值函数和优先队列，使得搜索具有一定的方向感，使得寻路不断向终点进行收敛。

以上两个算法都基于贪心的思想，即算法的运行过程可能会陷入局部最优解中。故在 80 年代初，模拟退火算法被提出<sup>[5]</sup>。其主要思想为算法能够以一定概率接受较差的解并继续向下搜寻，随着解的持续变差，概率将逐步降低。通过这个思想，有助于搜寻最短路径时，跳出局部最优，以求搜寻到全局最优解。

该算法类似于金属冶炼的退火过程，即接受较差结果的概率公式近似于热力学公式中关于温度跳变的公式。

同时，遗传算法也是起因于自然界某些规律，按照自然法则计算的两大分支，有国外学者将模拟退火和遗传算法结合起来，提高效率。于是在 80 年代末产生了并行再生模拟退火算法<sup>[6]</sup>，解决了退火算法的求解局限问题，又处理了传统遗传算法的收敛过早问题。

另外，受自然界真实蚁群集体行为的启发，意大利学者 Dorigo 于 1991 年首次系统地提出了一种基于蚂蚁种群的新型优化算法——蚁群算法（ACO）<sup>[7]</sup>。蚂蚁在搜寻最短路径的过程中，会通过同伴留下的信息激素来选择方向。若某个位置的信息素较多，说明那条路是被其他蚂蚁所选择的路，故引发了一种正反馈现象。该算法通过设定好相应的参数值，能够较好地解决旅行商等问题。故自 1996 年之后的 5 年时间里，蚁群算法逐渐引起了世界各国学者的关注，在应用领域得到了迅速发展。

现在国外的研究热点有以下四点<sup>[9]</sup>，一是依据实际网络的特征，进行特定的处理和优化。在时间复杂度没有太大变动的情况下，尽可能提高细节处的运行效率；二是在限制了网络的某些条件后，采用一些特殊的数据结构和相应算法来处理该问题。三是在拓扑层采用编码路径的视图形式，使用编码存储和运算来处理最短路问题。四是采用并行算法，并应用到并行计算服务中。

## 2.2 国内研究现状和历史

国内在计算机科学领域的起步较迟，故对于该算法，更多是进行算法的完善和改进。

武汉大学的张锦明博士对 Dijkstra 算法进行了分析<sup>[18]</sup>，找出影响该算法效率的关键步骤，并加入各种优化策略：相关邻接结点优化策略、结点分类优化策略、权值排序优化策略、相关边优化策略、结点排序优化策略和单链结点剔除优化策略。通过以上策略的结合，可使 Dijkstra 在解决最短路径问题时得到时间效率上的提高。

东北大学张纪会博士与徐心和教授最先在国内引进并研究蚁群算法<sup>[19]</sup>。他们分析了蚁群算法的优势和劣势，并成功地利用该算法解决了 TSP 旅行商问题，为后人提供了研究的参考价值。

上文介绍的模拟退火和遗传算法都是基于自然界的现象而提出的。由于二者各有缺点，为了提高效率。西安电子科技大学的王雪梅、王义和教授将两个算法结合起来，提出了并行再生模拟退火算法<sup>[6]</sup>，解决了传统遗传算法过早收敛问题，同时又解决了退火算法的局限性问题。

当前的最短路径算法研究正趋于向着各类加速技术、加速方法间的交叉结合的方向发展<sup>[17]</sup>。很多方法单独应用时才能使各自的优势充分体现。总体上该研究仍处于发展中阶段，还有很多工作需要进一步探索和认识。

而国内学者也注意到了使用串行计算机计算最短路径的局限性。现有成熟算法和开源库使得最短路径算法已几乎达到理论上的极限时间复杂度<sup>[9]</sup>。故有人开始研究并行计算，来解决最短路径问题。陕西师范大学计算机学院的卢照博士研究如何设计或实现某个或某类问题的并行求解。此类并行算法的设计往往还伴有在实际系统上的计算实例和性能分析。由于对处理器的数目进行了合理的限制，并行计算系统在实践中更有价值，是最短路径问题算法并行化研究的趋势所在。

### 3 相关问题的其他应用场景

对于最短路径问题，除了网络路由的传输外，还有其他经典应用场景，这些场景意味着该算法的应用广泛，研究的商业和学术价值极大。

#### 3.1 邮递员问题

邮递员从邮局出发，送完信件后回家（或回邮局），需要为他安排行驶路径使得总的行驶距离最短<sup>[11]</sup>。利用 1.2 中的描述，可理解为求 s 点到 e 点，接着再从 e 点到 s 点的最短来回路径。

#### 3.2 公交车线路设计问题

在交通运输中，某汽车队从甲地到乙地，假设选择路线的指标有两个：其一是距离短；其二是沿途中尽可能少的经过一些城市点（这是从时间上考虑到由于城市交通拥挤，经过一个城市需较多的时间）。即：汽车队往往要求所选择的行车路线在沿途经过城市的个数不超过 k 个的限制条件下，总路程最短。例

如，2008 年北京奥运会期间所使用的 34 条经过北京市主要奥运场馆的奥运公交专线的设计<sup>[12]</sup>。这些奥运公交专线在设计上具有一个共同的特点，就是从公交调度起点站出发后途径一些要求经过的奥运场馆站后（这里忽略公交车经过奥运场馆站点的顺序）到达公交终点调度站。

### 3.3 旅行家问题（TSP, Travel-ing Salesman Problem）

给旅行家设计一条旅行线路，使得他从某地出发，游玩一些事先计划的旅游景点后，到达另一目的地的总行驶距离最短<sup>[13]</sup>。此问题求解复杂，在特大规模的图中，是一个 NP 完全问题。（Nondeterministic Polynomial Complete Problem）

### 3.4 网路路由备选路问题

在企业搭建的大规模通讯路由网络中，会规定特定的必经路由集，需要有一个高效优良的寻路算法，来找到起始路由到终点路由的可行通路。当通路上的某段线缆出现问题，且不明确是哪段出问题时，需要有一条备选通路，要求备选通路与前者的重边尽可能地少，以避免故障线路。通过改进最短路算法<sup>[14]</sup>，可解决路由问题。

### 3.5 神经网络问题

为实现当代海量数据及知识的高速处理，一般的多机系统已不能满足要求。因而，出现了某些有别于一般多机系统的特殊结构，最具有代表性的是神经网络结构<sup>[15]</sup>。从便于对数据及知识作分布式并行处理考虑，最可取的方案是对数以千计的处理机作互相交连。但互连处理机之间的连接通道显然不是唯一的路径，因而就存在着如何寻找最短路径问题。如果处理机间连接的路径不是最短路径，则表明并未真正发挥系统的优势。故需要一个优良的最短路算法解决神经网络问题。

## 4 发展趋势

近十年来，最短路径算法的研究取得了许多重要的突破。一方面，算法的



查询时间得到了迅速的提升，另一方面硬件设施的升级使得问题计算能力不断加强，但由于业务规模不断扩大，挑战依然存在。

例如随着云计算的发展，企业对于网络路由的选路策略愈发严格。当面对几万个网络顶点的问题时，对算法的要求非常高。国内外公司和研究所都投入了很大的精力，来研发更合适更高效的网络转发算法。

另外，智能交通、智慧城市等概念的提出，使得“公交车线路问题”受到重视。目前收集道路网数据的技术日趋成熟，公开的道路网数据规模也随之增大，已知的最大道路网——美国道路网<sup>[16]</sup>，涵盖了 2300 多万个节点与 5800 多万条边，庞大的数据量，使得查询速度和查询结果还需要不断地进行优化。

本课题打算就寻找网络路由传输最短路径的两种特殊情况做一些深入研究。该两种情况为经过特定路由集以及求首选和备选路径。拟采用 Dijkstra 算法和 DFS 算法外加一系列的优化，解决该问题。

## 参考文献

- [1] 陆锋. 最短路径算法:分类体系与研究进展[J]. 测绘学报, 2001, 30(3):269-275.
- [2] Dijkstra E W. A note on two problems in connexion with graphs[J]. Numerische Mathematik, 1959, 1:269-271.
- [3] 张波良, 张瑞昌, 关佶红. 道路网上最短路径算法综述[J]. 计算机应用与软件, 2014(10):1-9.
- [4] An approach to automatic problem-solving [J]. Machine Intel-ligence, 1967, 1:105-127.
- [5] C.H.Papadimitrou, K.Steiglitz, Combinatorial optimization:Algorithms and complexity, Prentice-Hall, 1982.
- [6] 王雪梅, 王义和. 模拟退火算法与遗传算法的结合[J]. 计算机学报, 1997, 5(4):381-384.
- [7] Dorigo M, Maniezzo V, Clolorni A. The ant system:Optimization by a colony of cooperationg agents[J]. IEEE Trans on Systems,Man and Cybernetics,Part B, 1996, 26(1):29-41.
- [8] 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):69-71.
- [9] DEON, PANGCY. ShortestPathAlgorithms:TaxonomyandAnnotation[J]. Networks, 1984, 14:275-323.
- [10]Fu L, Sun D, Rilett L R. Heuristic shortest path algorithms for transportation applications: state of the art[J]. Computers & Operations Research, 2006, 33(11):3324-3343.
- [11]黄书力, 胡大裘, 蒋玉明. 经过指定的中间节点集的最短路径算法[J]. 计算机工程与应用 2015, 51(11).
- [12]姚广铮, 孙壮志, 孙福亮, 等. 北京奥运公交专线规划及评价方法[J]. 城市交通, 2008, 6(3):29-34.
- [13]高尚, 韩斌, 吴小俊, 等. 求解旅行商问题的混合粒子群优化算法[J]. 控制与决策, 2004, 19(11):1286-1289.
- [14]张毅, 张猛, 梁艳春. 改进的最短路径算法在多点路由上的应用[J]. 计算机科学, 2009, 36(8):205-207.

- [15]李望超. 神经网络中的最短路径问题[J]. 电子与信息学报, 1996, 2(S1):147-150.
- [16]9th DIMACS Implementation Challenge. Shortest Paths [OL]. 2006.
- [17]宋青, 汪小帆. 最短路径算法加速技术研究综述[J]. 电子科技大学学报, 2012, 41(2):176-184.
- [18]张锦明, 洪刚, 文锐, 等. Dijkstra 最短路径算法优化策略[J]. 测绘科学, 2009, 34(5):105-106.
- [19]张纪会, 徐心和, 等. 一种新的进化算法--蚁群算法[J]. 系统工程理论与实践, 1999, 19(3):84-87.

## 二、外文翻译



# A faster algorithm for the single source shortest path problem with few distinct positive lengths

James B. Orlin<sup>a</sup>, Kamesh Madduri<sup>b,1</sup>, K. Subramani<sup>c,\*</sup>, M. Williamson<sup>c</sup>

<sup>a</sup> Sloan School of Management, MIT, Cambridge, MA, USA

<sup>b</sup> Computational Research Division, Lawrence Berkeley Laboratory, Berkeley, CA, USA

<sup>c</sup> LDCSEE, West Virginia University, Morgantown, WV, USA

## ARTICLE INFO

### Article history:

Received 2 November 2008

Accepted 3 March 2009

Available online 20 March 2009

### Keywords:

Shortest path problem

Dijkstra's algorithm

Linear time

Red-blue graphs

## ABSTRACT

In this paper, we propose an efficient method for implementing Dijkstra's algorithm for the Single Source Shortest Path Problem (SSSPP) in a graph whose edges have positive length, and where there are few distinct edge lengths. The SSSPP is one of the most widely studied problems in theoretical computer science and operations research. On a graph with  $n$  vertices,  $m$  edges and  $K$  distinct edge lengths, our algorithm runs in  $O(m)$  time if  $nK \leq 2m$ , and  $O(m \log \frac{nK}{m})$  time, otherwise. We tested our algorithm against some of the fastest algorithms for SSSPP on graphs with arbitrary but positive lengths. Our experiments on graphs with few edge lengths confirmed our theoretical results, as the proposed algorithm consistently dominated the other SSSPP algorithms, which did not exploit the special structure of having few distinct edge lengths.

© 2009 Published by Elsevier B.V.

## 1. Introduction

In this paper, we provide an algorithm for solving the Single Source Shortest Path Problem (SSSPP) on a graph whose edges have positive length. The SSSPP is an extremely well-studied problem in both the operations research and the theoretical computer science communities because of its applicability in a wide range of domains. Ahuja et al. [2] describe a number of applications of the SSSPP, as well as efficient algorithms for the same. This paper provides an efficient algorithm for the SSSPP in the case where the number of distinct edge lengths is small. Our motivation for focusing on problems with few distinct edge lengths comes from a problem that arises in social networks (see Section 3).

We consider a graph with  $n$  vertices,  $m$  edges, and  $K$  distinct edge lengths. We provide two algorithms: The first algorithm is a simple implementation of Dijkstra's algorithm that runs in time  $O(m + nK)$ . The second algorithm modifies the first algorithm by using binary heaps to speed up the `FINDMIN()` operation. Assuming that  $nK > 2m$ , its running time is  $O(m \log \frac{nK}{m})$ .

For various ranges of the parameters  $n$ ,  $m$ , and  $K$ , the running time of our algorithm is less than the running time of Fredman and Tarjan's Fibonacci Heap implementation [8], which runs in  $O(m + n \log n)$  time. In fact, it improves upon the Atomic Heap implementation of Fredman and Willard [9], which runs in  $O(m + \frac{n \log n}{\log \log n})$  time. (This latter paper relies on a slightly different model of computation than is normally assumed in papers on algorithms.) In particular, our algorithm

\* Corresponding author.

E-mail addresses: jorlin@mit.edu (J.B. Orlin), kmadduri@lbl.gov (K. Madduri), ksmmani@csee.wvu.edu (K. Subramani), mwilli65@mix.wvu.edu (M. Williamson).

<sup>1</sup> This work was supported by the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

<sup>2</sup> This research has been supported in part by the Air Force Office of Scientific Research under grant FA9550-06-1-0050 and in part by the National Science Foundation through Award CCF-0827397.

runs in  $O(m)$  time whenever  $nK = O(m)$ . We also note that even if all edge lengths are distinct, the running time of our algorithm is  $O(m \log m)$ , which is the same time as the binary heap implementation of Dijkstra's algorithm.

The main contributions of this paper are as follows.

- (i) A new algorithm for the SSSPP problem that is parameterized by the number of distinct edge lengths.
- (ii) An empirical analysis of our algorithm that demonstrates the superiority of our approach when the number of distinct edge lengths is small.

The rest of this paper is organized as follows. Section 2 formally specifies the problem under consideration. Section 3 describes the motivation for our work. Related work in the literature is discussed in Section 4. Section 5 describes and analyzes the  $O(m + nK)$  implementation of Dijkstra's algorithm. Section 6 describes the  $O(m \log \frac{nK}{m})$  implementation and also provides a proof of its running time. In Section 7, we provide an empirical analysis, confirming the improved performance of our algorithm on graphs with few distinct edge lengths. We offer brief conclusions in Section 8.

## 2. Statement of problem

We consider a directed graph  $G = (V, E)$ , with a vertex set  $V$  with  $n$  vertices, and an edge set  $E$  with  $m$  edges. For each vertex  $v \in V$ , we let  $E(v)$  denote the set of edges directed out of  $v$ . Let  $L = \{l_1, \dots, l_K\}$  be the set of distinct nonnegative edge lengths given in increasing order. We assume that  $L$  is provided as part of the input and stored as an array. Each edge  $(i, j) \in E$  has an edge length  $c_{ij} \in L$ .

Rather than store the edge length  $c_{ij}$  explicitly, we assume that associated with each edge  $(i, j)$  is an index  $t_{ij}$  such that  $c_{ij} = l_{t_{ij}}$ . We note that in practice, one can determine  $L$  and all of the indices in  $O(m + K \log K)$  expected time; we first use perfect hashing to identify the  $K$  distinct edge lengths, and then sort the edge lengths.

There is a special vertex  $s \in V$  called the *source*. We let  $\delta(v)$  denote the length of the shortest path in  $G$  from vertex  $s$  to vertex  $v$ . If there is no path from  $s$  to  $v$ , then  $\delta(v) = \infty$ . The goal of the SSSPP is to identify a shortest path from vertex  $s$  to each other vertex that is reachable from vertex  $s$ .

## 3. Motivation

Our work was motivated by the “gossip” problem for social networks. Consider a social network, which is composed of clusters of participants. We model the intra-cluster distance by the value 1 and the inter-cluster distances by a real number  $l$ , where  $l > 1$ . The goal is then to determine the fastest manner in which gossip originating in a cluster can reach all the participants in the social network. This is a special case of SSSPP in which  $K = 2$ .

Although the motivating example has  $K = 2$ , we have extended our results to values of  $K$  that can grow with the input size.

Given that the SSSPP problem arises in so many domains, researchers have called for a “toolbox” [5,10] of different implementations that are efficient for different types of input. Possibly, the algorithm presented here would be appropriate for such a toolbox.

## 4. Related work

The literature on the SSSPP problem is large; interested readers are referred to Ahuja et al. [1]. In what follows, we briefly outline various paradigms in algorithmic advancement and provide a context for our work.

The first polynomial time algorithm for the SSSPP problem was devised by Dijkstra [7], the running time of the algorithm depends upon the data structure used to implement the priority queue, which is part of the algorithm. Since then, advances in algorithmic techniques have been along the following fronts:

- (i) New design paradigms – Thorup provided a linear-time algorithm for the SSSPP when the graph edges are undirected [14]. He exploited the connection between the Minimum Spanning Tree of an undirected graph and the Single Source Shortest paths tree.
- (ii) Data Structure improvements – Algorithms based on Dijkstra's approach perform a series of EXTRACT-MIN() and DECREASE-KEY() operations. Inasmuch as each vertex is extracted only once and each edge is processed at most once, the running time of any such algorithm can be represented as:  $T(n, m) = n * \text{EXTRACT-MIN}() + m * \text{DECREASE-KEY}()$ . An optimized priority queue design balances the costs between the two operations; in the comparison based-model, the most efficient priority queue for this pair of operations is the Fibonacci Heap. Dijkstra's algorithm with Fibonacci Heaps has a running time of  $O(m + n \log n)$ . Other heaps proposed for this problem include the  $d$ -heap [3] and the  $R$ -heap [6].
- (iii) Parameterization – In this approach, the design focuses on a certain parameter (or parameters) that may be small in magnitude for an interesting subset of problems. One such parameter is the largest edge length (say  $C$ ); Ref. [2] describes how Dijkstra's approach can be made to run in  $O(m + n\sqrt{\log C})$  time.

- (iv) Input restriction – Special-purpose algorithms have been designed for the case in which the edge lengths are drawn from certain distributions. The analysis of such an algorithm exploits this distribution and provides an estimate of the expected running time [12].

We believe that this is the first shortest path paper to express the running time in terms of the number of distinct edge lengths. However, this parameter or a closely related parameter has been part of the analysis of algorithms for other problems.

## 5. An $O(m + nK)$ implementation of Dijkstra's algorithm

In this section, we provide an  $O(m + nK)$  implementation of Dijkstra's algorithm for solving the SSSPP. While running Dijkstra's algorithm, we maintain the following structures:

- (i) the set  $S$ , which denotes the set of permanently labeled vertices, and
- (ii) the set  $T = V - S$ , which denotes the set of temporarily labeled vertices.

The value  $d(j)$  is the distance label of vertex  $j$ . If  $j \in S$ , then  $d(j) = \delta(j)$  is the length of the shortest path from vertex  $s$  to vertex  $j$  in  $G$ . Finally, we let  $d^* = \max\{d(j) : j \in S\}$  be the distance label of the vertex most recently added to  $S$ .

When implemented naively, the bottleneck operation in Dijkstra's algorithm is the  $\text{FINDMIN}()$  operation, which identifies the minimum distance label of a vertex in  $T$ . Each  $\text{FINDMIN}()$  operation takes  $O(|T|)$  steps and thus the  $\text{FINDMIN}()$  operations take  $O(n^2)$  steps in all. All other updates take  $O(m)$  steps in total. In order to reduce the running time for the  $\text{FINDMIN}()$  step, Dijkstra's algorithm relies on one of many different implementations of a priority queue. Of these, the implementations with the best asymptotic bounds are the Fibonacci Heap implementation [8] and the Atomic Heap implementation [9]. Here we can dramatically speed up the  $\text{FINDMIN}()$  operations in the case that the number of distinct edge lengths is small.

Let  $L = \{l_1, \dots, l_K\}$  be the set of distinct edge lengths. For each  $t = 1$  to  $K$ , the algorithm will maintain a list  $E_t(S) = \{(i, j) \in E : i \in S, c_{ij} = l_t\}$ . These edges are sorted in the order that the tail of the edge is added to  $S$ . That is, if edge  $(i, j)$  occurs prior to edge  $(i', j')$  on  $E_t(S)$ , then  $d(i) \leq d(i')$ . The algorithm also maintains  $\text{CurrentEdge}(t)$ , which is the first edge  $(i, j)$  of  $E_t(S)$  such that  $j \in T$ . If no such edge exists in  $E_t(S)$ , then  $\text{CurrentEdge}(t) = \emptyset$ . If  $(i, j) = \text{CurrentEdge}(t)$ , then we let  $f(t) = d(i) + l_t$ , which is the length of the shortest path from vertex  $s$  to vertex  $i$  followed by edge  $(i, j)$ . It is not necessarily the case that  $f(t) = d(j)$  because there may be edges of other lengths directed into vertex  $j$ .

These additional data structures makes it possible to determine the vertex in  $T$  with minimum distance label by determining  $\argmin\{f(t) : 1 \leq t \leq K\}$ . The time for this  $\text{FINDMIN}()$  operation is  $O(K)$  if implemented directly (and naively) without any priority queue data structure. This leads to an improvement in the overall running time for Dijkstra's algorithm when  $K$  is small.

The subroutine  $\text{UPDATE}(t)$  moves the pointer  $\text{CurrentEdge}(t)$  so that it points to the first edge whose endpoint is in  $T$  (or sets  $\text{CurrentEdge}(t)$  to  $\emptyset$ ). If  $\text{CurrentEdge}(t) = (i, j)$ , then  $\text{UPDATE}(t)$  also sets  $f(t) = d(i) + c_{ij}$ . If  $\text{CurrentEdge}(t) = \emptyset$ , then  $\text{UPDATE}(t)$  sets  $f(t) = \infty$ . The operator  $\text{CurrentEdge}(t).\text{next}$  moves the  $\text{CurrentEdge}$  pointer by one step in the linked list representing  $E_t(S)$  (Algorithm 5.1).

**Theorem 5.1.** *Algorithm 5.2 determines the shortest path from vertex  $s$  to all other vertices in  $O(m + nK)$  time.*

---

```

Function INITIALIZE()
1:  $S := \{s\}$ ;  $T := V - \{s\}$ .
2:  $d(s) := 0$ ;  $\text{pred}(s) := \emptyset$ .
3: for (each vertex  $v \in T$ ) do
4:    $d(v) = \infty$ ;  $\text{pred}(v) = \emptyset$ .
5: end for
6: for ( $t = 1$  to  $K$ ) do
7:    $E_t(S) := \emptyset$ .
8:    $\text{CurrentEdge}(t) := \text{NIL}$ .
9: end for
10: for each edge  $(s, j)$  do
11:   Add  $(s, j)$  to the end of the list  $E_t(S)$ , where  $l_t = c_{sj}$ .
12:   if ( $\text{CurrentEdge}(t) = \text{NIL}$ ) then
13:      $\text{CurrentEdge}(t) := (s, j)$ 
14:   end if
15: end for
16: for ( $t = 1$  to  $K$ ) do
17:    $\text{UPDATE}(t)$ 
18: end for

```

---

**Algorithm 5.1.** The initialization procedure.

---

```

Function NEW-DIJKSTRA()
1: INITIALIZE()
2: while ( $T \neq \emptyset$ ) do
3:   let  $r = \operatorname{argmin}\{f(t): 1 \leq t \leq K\}$ .
4:   let  $(i, j) = \operatorname{CurrentEdge}(r)$ .
5:    $d(j) := d(i) + l_r$ ;  $\operatorname{pred}(j) := i$ .
6:    $S = S \cup \{j\}$ ;  $T := T - \{j\}$ .
7:   for (each edge  $(j, k) \in E(j)$ ) do
8:     Add the edge  $(j, k)$  to the end of the list  $E_t(S)$ , where  $l_t = c_{jk}$ .
9:     if ( $\operatorname{CurrentEdge}(t) = \text{NIL}$ ) then
10:        $\operatorname{CurrentEdge}(t) := (j, k)$ 
11:     end if
12:   end for
13:   for ( $t = 1$  to  $K$ ) do
14:      $\operatorname{UPDATE}(t)$ .
15:   end for
16: end while

```

---

**Algorithm 5.2.** Dijkstra's algorithm with few distinct edge lengths.

---

```

Function UPDATE( $t$ )
1: Let  $(i, j) = \operatorname{CurrentEdge}(t)$ .
2: if ( $j \in T$ ) then
3:    $f(t) = d(i) + c_{ij}$ .
4:   return
5: end if
6: while ( $(j \notin T)$  and  $(\operatorname{CurrentEdge}(t).\text{next} \neq \text{NIL})$ ) do
7:   Let  $(i, j) = \operatorname{CurrentEdge}(t).\text{next}$ .
8:    $\operatorname{CurrentEdge}(t) = (i, j)$ .
9: end while
10: if ( $j \in T$ ) then
11:    $f(t) = d(i) + c_{ij}$ .
12: else
13:   Set  $\operatorname{CurrentEdge}(t)$  to  $\emptyset$ .
14:    $f(t) = \infty$ .
15: end if

```

---

**Algorithm 5.3.** The update procedure.

**Proof.** The algorithm is identical to Dijkstra's algorithm except that it maintains some additional data structures to carry out the  $\text{FINDMIN}()$  operation. Therefore, Algorithm 5.2 computes the shortest paths from vertex  $s$  correctly.

The initialization takes  $O(n)$  time. The potential bottleneck operations are the determining of  $r = \operatorname{argmin}\{f(t): 1 \leq t \leq K\}$  and the time to perform  $\text{UPDATE}(t)$  over all iterations. All other steps have running times dominated by one of these two steps. We first note that the time to compute (see Algorithm 5.3)  $r = \operatorname{argmin}\{f(t): 1 \leq t \leq K\}$  is  $O(K)$  per iteration of the **while** loop and  $O(nK)$  over all iterations.

We next consider the time needed to perform  $\text{UPDATE}(t)$  over all iterations. The procedure  $\text{UPDATE}(t)$  is called  $O(nK)$  times, and its total running time is  $O(m + nK)$ . To see this, first note that the running time as restricted to iterations in which  $\operatorname{CurrentEdge}(t)$  is not changed is  $O(nK)$ . We now consider those iterations at which  $\operatorname{CurrentEdge}(t)$  is changed. Suppose  $(i, j) = \operatorname{CurrentEdge}(t)$  at the beginning of an iteration, and suppose that  $i \in S$  and  $j \in S$ . Because the edges in  $E_t(S)$  are scanned sequentially, the edge  $(i, j)$  is never scanned again after updating  $\operatorname{CurrentEdge}(t)$ . So, the running time over all iterations in which  $\operatorname{CurrentEdge}(t)$  is changed is  $O(m)$ . We conclude that the total running time of Algorithm 5.2 is  $O(m + nK)$ .  $\square$

The original motivation for this paper was the case that  $K = 2$ , in which case the algorithm is particularly efficient. In the next section, we show how to speed up the algorithm in the case that  $K$  is permitted to grow with the problem size.

## 6. A faster algorithm if $K$ is permitted to grow with problem size

We now revise the algorithm to improve its running time in the case that  $K$  is not a constant. We let  $q = \frac{nK}{m}$ . We assume that  $q \geq 2$ ; if not, Algorithm 5.2 runs in linear time. To simplify the exposition, we will assume that  $q$  is an integer divisor of  $K$ , and we let  $h = \frac{K}{q}$ . Since we are focused on asymptotic analysis, we can make this assumption without loss of generality.

We will show that Algorithm 5.2 can be refined to run in time  $O(m \log q)$ . In order to achieve this running time, we need to speed up the bottlenecks in Algorithm 5.2 that depend on  $K$ . We need to compute  $r$  more efficiently, and we need to call  $\text{UPDATE}(t)$  less frequently. We first address speeding up the computation of  $r$ .

In order to speed up the determination of  $r$ , we will store the values  $f()$  in a collection of  $h$  different binary heaps. The first binary heap stores the values  $f(j)$  for  $j = 1$  to  $q$ , the second binary heap stores the values  $f(j)$  for  $j = q + 1$  to  $2q$ ,



and so on. We denote the heaps as  $H_1, H_2, \dots, H_h$ . Finding the element with minimum value in the binary heap  $H_i$  takes  $O(1)$  steps. The time to insert an element into  $H_i$  or delete an element from  $H_i$  takes  $O(\log q)$  steps. (For more details on binary heaps see [3].)

We find the minimum  $f()$  value by first finding the element with minimum value in each of the  $h$  heaps and then choosing the best of these  $h$  elements. Finding the minimum key in a heap takes  $O(1)$  steps; so the time for this implementation of the `FINDMIN()` operations is  $O(h)$  per iteration of the while loop, and  $O(hn) = O(m)$  overall. After finding the minimum element in the  $h$  different heaps, we delete the element from its heap. This takes  $O(n \log q)$  steps over all iterations.

We now address the time spent in `UPDATE()`. In order to minimize the time required by `UPDATE()`, we first relax the requirement on `CurrentEdge`. If `CurrentEdge(t)` has both of its endpoints in  $S$ , we say that `CurrentEdge(t)` is *invalid*. We permit `CurrentEdge(t)` to be invalid at some intermediate stages of the algorithm. We then modify the `FINDMIN()` again. If the minimum element in heap  $H_i$  is  $f(t)$  for some  $i$ , and if `CurrentEdge(t)` is invalid, the algorithm then performs `UPDATE()`, followed by finding the new minimum element in  $H_i$ . It iterates in this manner until the minimum element corresponds to a valid edge. In this way, whenever the algorithm calls `UPDATE()`, it leads to a modification of `CurrentEdge()`. Moreover, whenever the algorithm selects the minimum element among the  $q$  different heaps, the minimum element in each of the heaps corresponds to a valid edge. Since every modification of `CurrentEdge()` leads to a change in one of the values in a heap, and since there are at most  $m$  modifications of `CurrentEdge()`, the total running time for `UPDATE()` over all iterations is  $O(m \log q)$ .

We summarize the previous discussion with [Theorem 6.1](#).

**Theorem 6.1.** *The binary heap implementation of Dijkstra's algorithm with  $O(\frac{K}{q})$  binary heaps of size  $O(q)$  with  $q = \frac{nK}{m}$  determines the shortest path from vertex  $s$  to all other vertices in  $O(m \log q)$  time.*

## 7. Empirical results

### 7.1. Experimental setup

We evaluate performance of our algorithms on several graph families. Some of the generators and graph instances are part of the 9th DIMACS Shortest Path Implementation Challenge benchmark package [5]:

- *Random graphs:* We generate graphs according to the Erdos–Renyi random graph model, and ensure that the graph is connected. The generator may produce parallel edges as well as self-loops. The ratio of the number of edges to the number of vertices can be varied, and we experiment with both dense and sparse graphs. Random graphs have a low diameter and a Gaussian degree distribution.
- *Mesh graphs:* This synthetic generator produces two-dimensional meshes with grid dimensions  $x$  and  $y$ . We generate *Long* ( $x = \frac{n}{16}$ ,  $y = 16$ ) and *Square* grids ( $x = y = \sqrt{n}$ ) in this study. The diameter of these graphs is significantly higher than random graphs and all the vertices have a constant degree.
- *Small-world graphs:* We use the R-MAT graph model for real-world networks [4] to generate graphs with small-world characteristics. These graphs have a low diameter and an unbalanced degree distribution.

The edge weights are chosen from a fixed set of distinct random integers, as our new algorithm is designed for networks with small  $K$ .

We compare the execution time of our algorithm with the reference SSSPP solver used in the 9th DIMACS Shortest Paths Challenge (an efficient implementation of Goldberg's algorithm [11,13], which has expected-case linear running time, and highly optimized for integer edge weights) and the baseline Breadth-First Search (BFS) on every graph family. The BFS running time is a natural lower bound for SSSPP implementations. It is also reasonable to directly compare the execution times of DIMACS reference solver code and our implementation: both use a similar adjacency array representation for the graph, are written in C/C++, and compiled and run in identical experimental settings. Note that our implementation can process graphs with real as well as integer weights, but is only efficient for networks with a few distinct edges. We use only integer weights in this study for comparison with the DIMACS solver.

Our test platform for performance results is a 2.8 GHz 32-bit Intel Xeon machine with 4 GB memory, 512 KB cache and running RedHat Enterprise Linux 4 (linux kernel 2.6.9). We compare the sequential performance of our implementation with the DIMACS reference solver [5]. Both the codes are compiled with the Intel C compiler (icc) version 9.0, and the optimization flag `-O3`. We report the average execution time of five independent runs for each experiment.

### 7.2. Results and analysis

We conduct an extensive study to empirically evaluate the performance dependence on the graph topology, the problem size, the value of  $K$ , and the edge weight distribution. We report the execution time of Breadth-First Search on all the graph instances we studied in [Tables 1 and 2](#). The figures plot the execution times of the shortest path implementations normalized to the BFS time. Thus a ratio of 1.0 is the best we can achieve, and smaller values are desirable.

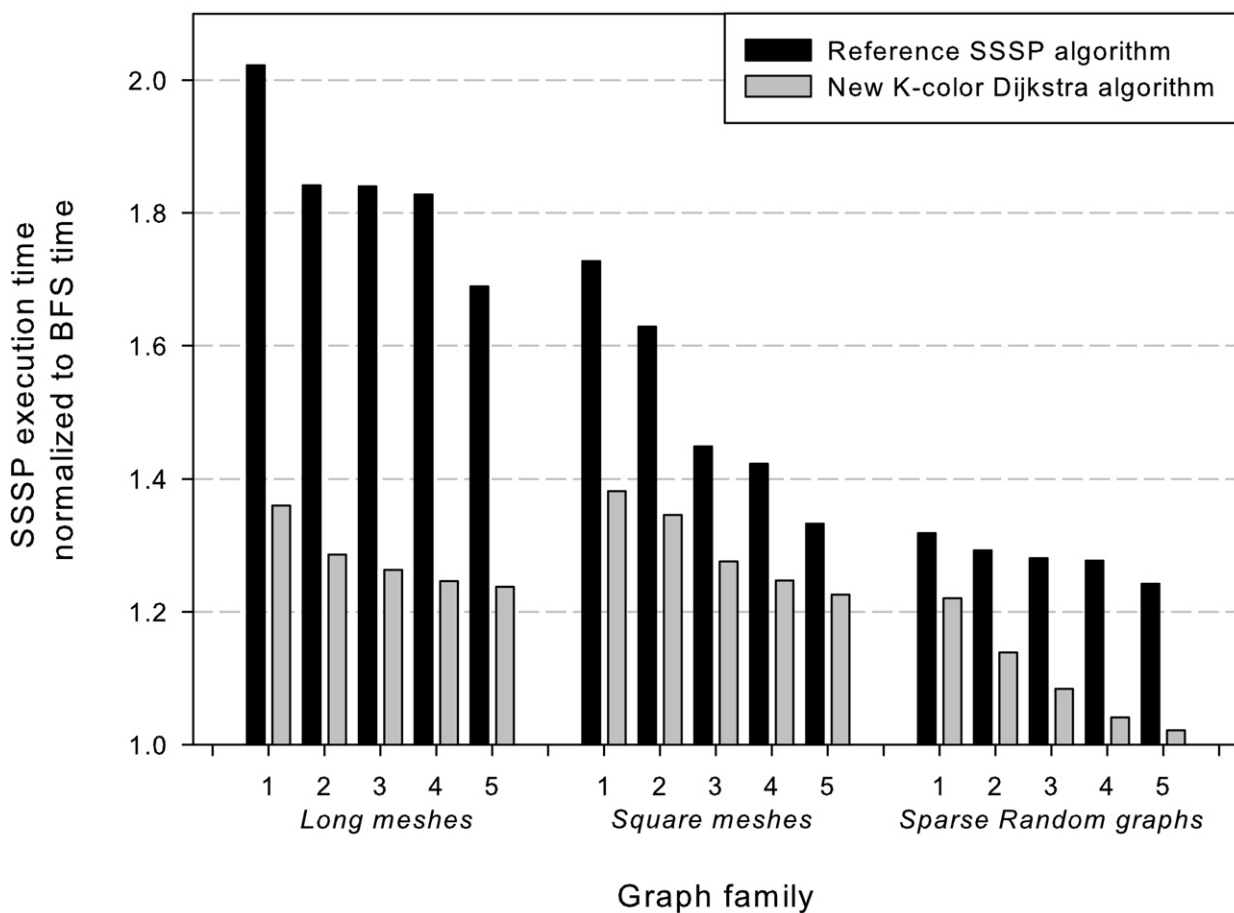
**Table 1**

Breadth-First Search execution time (in milliseconds) for various graph families on the test sequential platform.

Problem instance	Graph size	BFS time (milliseconds)		
		Long mesh	Square mesh	Random
1	100K vertices, 400K edges	50	55	95
2	500K vertices, 2M edges	290	350	540
3	1M vertices, 4M edges	660	870	1180
4	5M vertices, 20M edges	4160	6400	8390
5	10M vertices, 40M edges	8590	13 500	17 980

**Table 2**Breadth-First Search execution time (in milliseconds) for various graph families (the value of  $K$  is varied in experiments) on the test sequential platform.

Problem instance	BFS time (milliseconds)
1 Sparse random, 2M vertices, 8M edges, $C = 10000$	6430
2 Dense random, 100K vertices, 100M edges, $C = 100$	150
3 Long mesh, 2M vertices, 8M edges, $C = 100$	3260
4 Square mesh, 2M vertices, 8M edges, $C = 100$	4900
5 Small-world graph, 2M vertices, 8M edges, $C = 10000$	5440

**Fig. 1.** Performance of our shortest implementation and the reference solver for three graph families, as the problem size is varied. Graph 1 corresponds to the smallest network in our study, and 5 is the largest.

We first study the dependence of execution time on the problem size. We vary the size up to two orders of magnitude for three different graph families and compute the average SSSP execution time for the reference code and our implementation. Fig. 1 plots these normalized values for the different graph families. The problem sizes are listed in Table 1. The value of  $K$  is set to 2 in this case, and the ratio of the largest to the smallest edge weight in the graph (denoted by  $C$ ) is set to 100. Also,

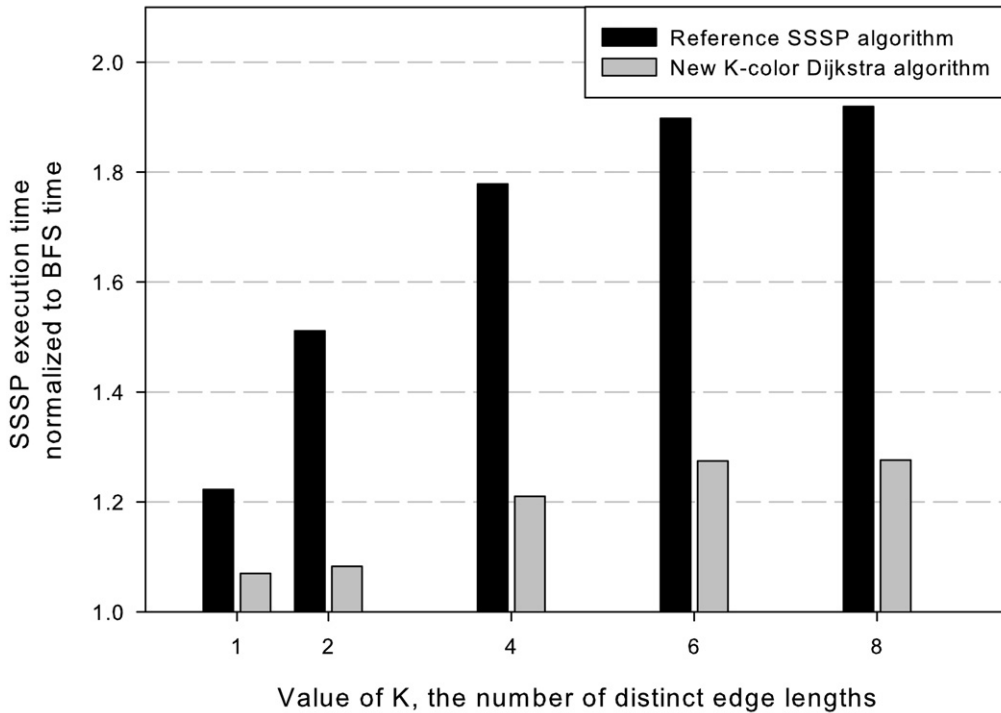


Fig. 2. Normalized SSSPP performance for a sparse random graph (4 million vertices, 16 million edges) as the value of  $K$  is varied.

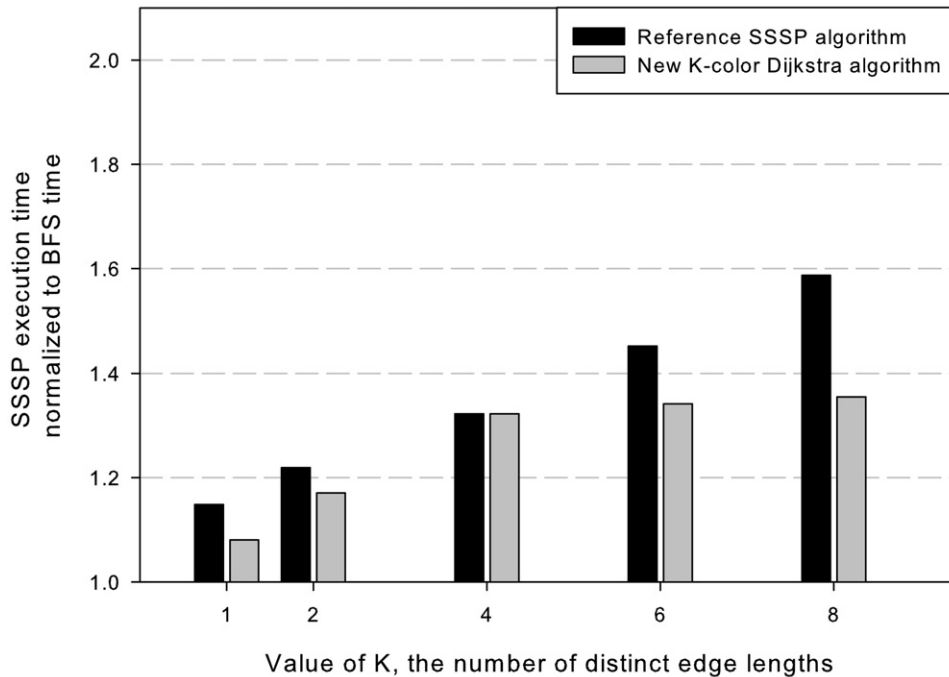


Fig. 3. Normalized SSSPP performance for a small-world graph (4 million vertices, 16 million edges) as the value of  $K$  is varied.

note that the ratio  $m/n$  is 4 in all the cases. In Fig. 1, we observe that our new implementation outperforms the reference solver for all graph families. Furthermore, the performance ratio is less than 2 in most cases, which is quite significant. On closer inspection, we observe that the performance improvements for long and square mesh graphs are comparatively higher than the random graphs. We attribute this to the fact that we do not maintain a priority queue data structure. Thus we avoid the priority queue overhead involved in evaluating long paths in mesh networks, such as frequent updates to the

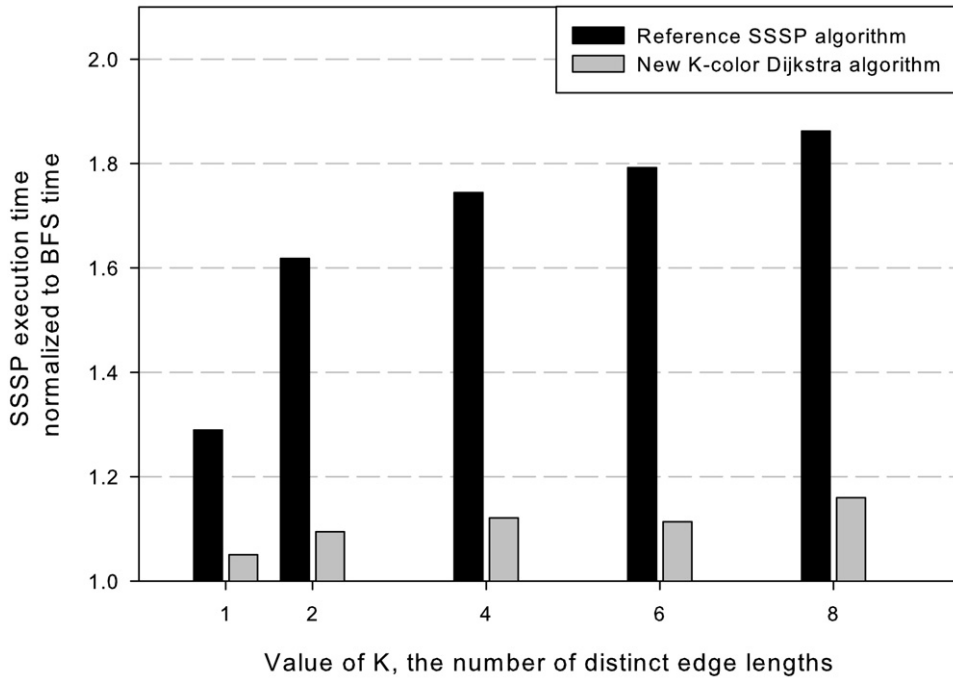


Fig. 4. Normalized SSSPP performance for a long mesh (4 million vertices, 16 million edges) as the value of  $K$  is varied.

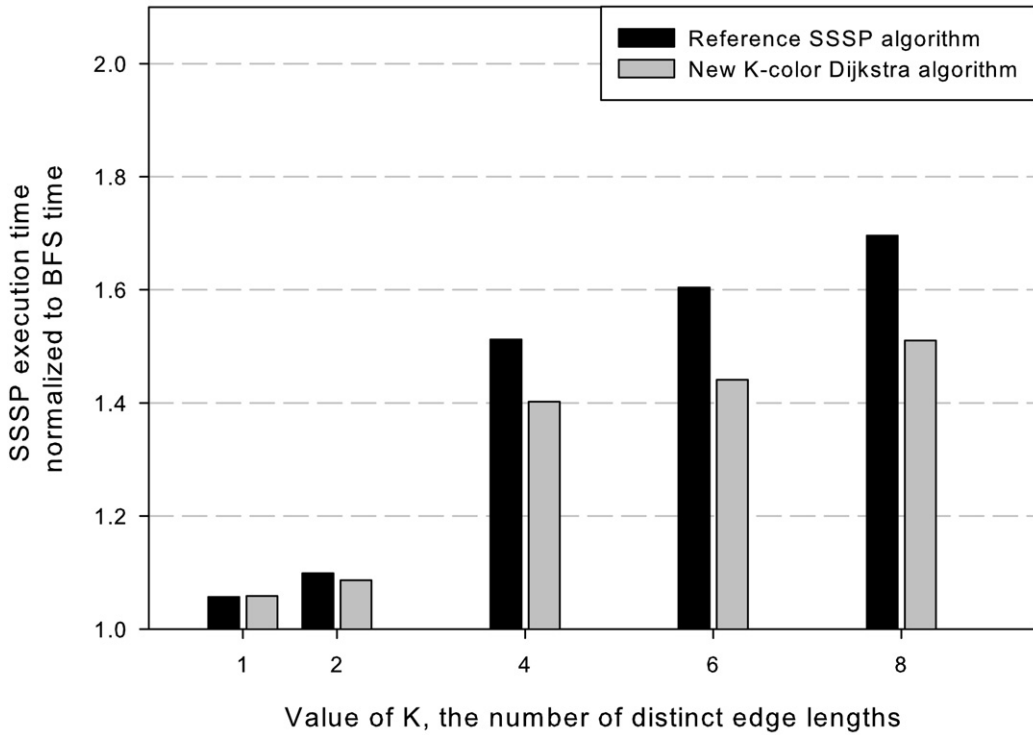


Fig. 5. Normalized SSSPP performance for a square mesh (4 million vertices, 16 million edges) as the value of  $K$  is varied.

distance values. We also observe that the performance is better for larger graph instances compared to smaller ones. Also, the performance ratios for sparse random networks (1.02–1.22) are very impressive for the problem instances we studied.

We next study the performance of the algorithm on each graph family as the value of  $K$ , the number of distinct edge weights is varied. We vary the value of  $K$  from 1 to 8 in each case, and plot the execution time. Fig. 2 plots the normalized

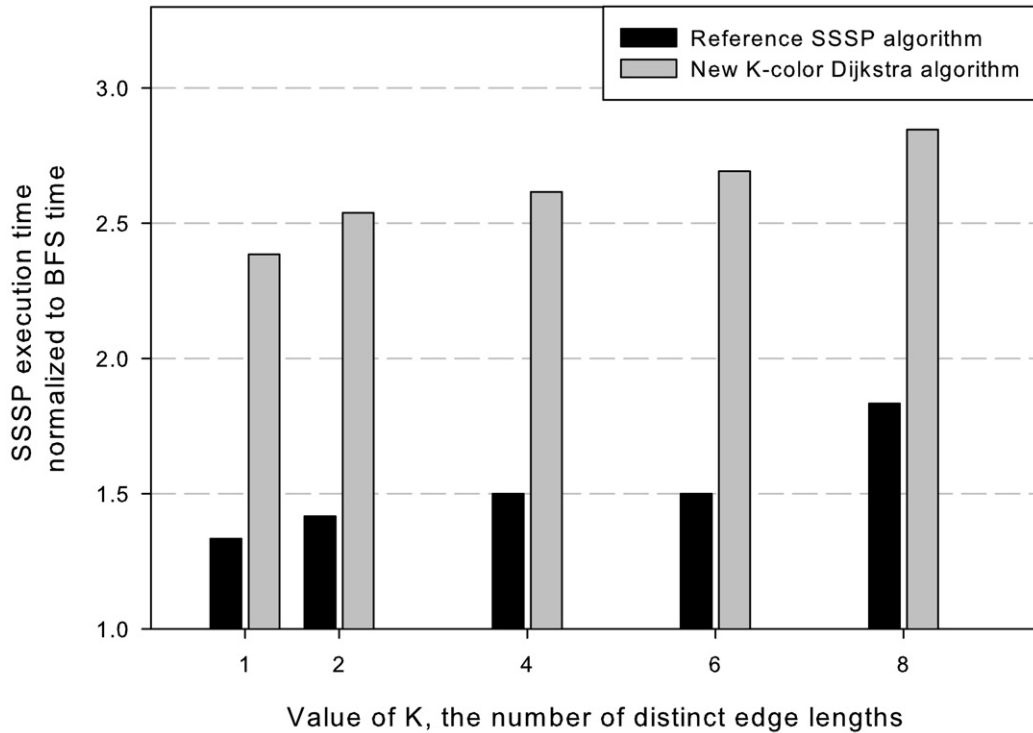


Fig. 6. Normalized SSSPP performance for a dense random graph (100K vertices, 10 million edges) as the value of  $K$  is varied.

execution time for a sparse random graph of 4 million vertices and 16 million edges. We observe that our implementation is significantly faster than the reference solver (up to 70% faster for  $K = 8$ ), and also scales slower than the reference solver with increase in  $K$ . The SSSP running time for random graphs is comparatively lower than the execution time for other graph families.

Fig. 3 plots the normalized execution time for synthetic small-world graphs. These are low diameter graphs similar to sparse random networks, but also have topological properties that are observed in large-scale social and biological networks. In this case, we observe that the reference solver and our algorithm perform very similarly.

For long meshes (Fig. 4), we observe significant performance gains over the reference solver. Also, the execution time of our algorithm scales much slower than the reference solver as  $K$  is increased. For square meshes (Fig. 5) also, we find that our implementation dominates for all values of  $K$ .

Fig. 6 plots the normalized execution time for a dense random graph of 100K vertices and 100 million edges as the value of  $K$  is varied. We observe that the reference solver is faster than our algorithm in this case, but the speedup falls as the value of  $K$  increases. We attribute this to the fast execution time of BFS (as this is a comparatively small graph instance). The overhead in executing our algorithm is high in this case, and it does not perform as well as the reference solver for a dense graph of this size.

The execution times of BFS for all these graph instances are listed in Table 2. The ratio of the highest to the least edge weight in the graph ( $C$ ) is also listed, as the worst case running time of the reference solver depends on this value. The performance of our algorithm, however, is independent of  $C$ .

## 8. Conclusions

The SSSPP is a fundamental problem within computer science and operations research communities. In this paper, we studied the SSSPP when the number  $K$  of distinct edge lengths is small. Our algorithm runs in linear time when the number of distinct edge lengths is smaller than the density of the graph.

## Acknowledgements

The third author was introduced to the gossip problem in social networks, also known as the Red–Blue Shortest Paths Problem, at the Sandia National Laboratories by Bruce Hendrickson.

## References

- [1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, *Network Flows: Theory, Algorithms and Applications*, Prentice-Hall, 1993.
- [2] R.K. Ahuja, K. Mehlhorn, J.B. Orlin, R.E. Tarjan, Faster algorithms for the shortest path problem, *J. ACM* 37 (2) (April 1990) 213–223.
- [3] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, *Introduction to Algorithms*, MIT Press, 2001.
- [4] D. Chakrabarti, Y. Zhan, C. Faloutsos, R-MAT: A recursive model for graph mining, in: *Proc. 4th SIAM Intl. Conf. on Data Mining*, Florida, USA, April 2004.
- [5] C. Demetrescu, A.V. Goldberg, D. Johnson, 9th DIMACS implementation challenge – Shortest Paths, <http://www.dis.uniroma1.it/challenge9/>, 2005.
- [6] J.R. Driscoll, H.N. Gabow, R. Shrairman, R.E. Tarjan, Relaxed heaps: An alternative to Fibonacci heaps with applications to parallel computation, *Commun. ACM* 31 (11) (1988) 1343–1354.
- [7] E.W. Dijkstra, A note on two problems in connexion with graphs, *Numer. Math.* 1 (1959) 269–271.
- [8] M.L. Fredman, R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, *J. ACM* 34 (3) (1987) 596–615.
- [9] M.L. Fredman, D.E. Willard, Trans-dichotomous algorithms for minimum spanning trees and shortest paths, *J. Comput. Syst. Sci.* 48 (3) (1994) 533–551.
- [10] A.V. Goldberg, Network optimization library, <http://www.avglab.com/andrew/soft.html>.
- [11] A.V. Goldberg, A simple shortest path algorithm with linear average time, in: *ESA, 2001*, pp. 230–241.
- [12] A.V. Goldberg, A simple shortest path algorithm with linear average time, in: *9th Ann. European Symp. on Algorithms (ESA 2001)*, Aachen, Germany, in: *Lecture Notes in Computer Science*, vol. 2161, Springer, 2001, pp. 230–241.
- [13] U. Meyer, Single-source shortest-paths on arbitrary directed graphs in linear average-case time, in: *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-01)*, New York, January 7–9, 2001, ACM Press, 2001, pp. 797–806.
- [14] M. Thorup, Undirected single source shortest path in linear time, in: *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS-97)*, Los Alamitos, October 20–22, 1997, IEEE Computer Society Press, 1997, pp. 12–21.

# 较少正边长数量的单源最短路径快速算法

James B. Orlin a, Kamesh Madduri b, 1, K. Subramanic, 2, M. Williamson c

翻译人 李绍晓 专业班级 电信 1302

**摘要：**在本文中，我们提出了一种实现 Dijkstra 算法的有效算法，该算法用于边长为正值的单源最短路径问题。单源最短路径是计算机科学理论和实验中被研究最广泛的问题。在 $n$ 个顶点、 $m$ 条边和 $K$ 种不同边长的图中，若 $nk \geq 2m$ ，则该算法的时间复杂度为 $O(m)$ ，否则为 $O(m \log \frac{nk}{m})$ 。我们在任意正长度的图上测试了我们的算法，并且和其他最快单源最短路径算法进行了比对。我们的实验证实了该算法在边长数量少的图中，能够优于其他算法，没有利用一些特殊的边长结构。

## 1. 引言

在本文中，我们提供一个在边长为正数的图中解决单源最短路径问题（以下简称 SSSPP）的算法。在工程实践和计算机科学理论领域，SSSPP 都是一个非常值得研究的问题，因为它的适用领域非常广。Ahuja et al<sup>[2]</sup>描述了 SSSPP 被应用的数量，以及相同场景的高效算法。本文针对具有少数不同边长的 SSSPP 提供了一个高效算法。我们对该问题的研究动机来自于社交网络中出现的问题。

我们所考虑的图具有 $n$ 个顶点， $m$ 条边以及 $K$ 种权值。我们提供了两种算法：第一种算法是一种时间复杂度为 $O(m + nk)$ 的 Dijkstra 算法简单实现。第二种算法是在第一种算法的基础上进行修改，使用二进制堆加快操作。假设 $nk \geq 2m$ ，其运行时间为 $O\left(\log \frac{nk}{m}\right)$ 。

对于不同范围的参数 $n$ ， $m$ 和 $K$ ，我们算法的运行时间小于 Fredman 和 Tarjan's 的 Fibonacci Heap 实现。它们的时间复杂度为 $O(m + n \log n)$ 。实际上，他改善了 Fredman 的原子堆实现，它的时间复杂度为 $O\left(m + \frac{n \log n}{\log \log n}\right)$ 。（后面

的文章依赖于一种不同的计算模型用来假设文中的算法)。当 $nk = O(m)$ 时, 我们算法的时间复杂度为 $O(m)$ 。我们还注意到, 即使所有的边长都是不同的, 我们算法的时间复杂度依然为 $O(m \log \frac{nK}{m})$ 。这与 Dijkstra 算法的二进制堆实现相同。

本文的主要贡献如下:

- (i) 提供一种针对具有不同边长数量的 SSSPP 新算法。
- (ii) 提供一种实证分析, 证明该算法在少量不同边长数量时的优势。

本文的其余部分组织如下。第 2 节正式规定了所考虑的问题。第 3 节描述了我们工作的动机。文献中的相关工作在第 4 节中讨论。第 5 节描述和分析 Dijkstra 算法的 $O(m + nK)$ 实现。第 6 节描述了执行时间复杂度为 $O(m \log \frac{nK}{m})$ , 并提供其证明。在第 7 节, 我们提供一个实证分析, 证明该算法在具有少量不同边长的图上具有一定优势。我们在第 8 节中提供简要的结论。

## 2. 问题描述

我们考虑有向图 $G = (V, E)$ , 顶点集 $V$ 含有 $n$ 个顶点, 边集 $E$ 含有 $m$ 条边。对于每个 $v \in V$ ,  $E(v)$ 表示从该点引出的边集。 $L = \{l_1, l_2, \dots, l_K\}$ 是不同的非负边长集合, 以递增的顺序给出。我们假设 $L$ 是输入的一部分, 并存储为数组。每条边 $(i, j) \in E$ , 其边长 $c_{ij} \in L$ 。并不是直接存储 $c_{ij}$ , 而是每条边 $(i, j)$ 对应一个索引 $t_{ij}$ , 通过该索引获得 $c_{ij} = l_{t_{ij}}$ 。在实践中我们注意到, 一旦确定了 $L$ , 我们就可以确定 $O(m + K \log K)$ 中所有的指数。我们首先使用哈希表去识别这 $K$ 种长度的边长, 并进行排序。有一个特殊的顶点称为 $s$ , 称为源点。我们让 $\delta(v)$ 表示在 $G$ 图中从顶点 $s$ 到顶点 $v$ 的最短路径长度。若不存在该路径, 则令 $\delta(v) = \infty$ 。SSSPP 的目标就是求得从 $s$ 到 $v$ 的最短路径, 且 $s$ 可到达其他每个顶点。

## 3. 研究动机

我们的研究动机来自于社交网络的“八卦消息”问题。考虑一个由参与者



集群组成的社交网络。我们将群集距离的最小单位定义为 1，群集的距离定义为一个整数 $l$ ，其中 $l > 1$ 。其目标是确定在社交网络中，源自某集群的八卦消息传播到所有参与者最快方式。这是 SSSPP 的一种特殊情况，其中 $K = 2$ 。

## 4. 相关工作

关于 SSSPP 问题的文献很多；感兴趣的读者可参考 Ahuja et al<sup>[1]</sup>。在下面，我们简要概述了算法进步中的各种范式，并为我们的工作提供了一个上下文。

第一种解决 SSSPP 问题的算法由 Dijkstra<sup>[7]</sup>设计，算法的时间复杂度取决于实现优先队列的数据结构，这是算法中的一部分。从那以后，改进的算法一直沿用下面几个方面：

(i) 新的设计方式——为 SSSPP 的无向图情况提供线性的时间算法。他利用无向图的最小生成树和单源最短路径树来进行连接。

(ii) 数据结构的改进——基于 Dijkstra 的算法执行一系列 *Extract - Min()* 和 *Decrease - key()* 操作。由于每个顶点只能使用一次，故该算法的运行时间为  $T(n, m) = n * \text{Extract} - \text{Min}() + m * \text{Decrease} - \text{Key}()$ 。优化的优先队列在基于比较的模型中设计平衡了两个操作之间的成本。

对该操作最有效的优先级队列是 Fibonacci 堆。Dijkstra 的算法与 Fibonacci 堆的运行时间为  $O(m + n \log n)$ 。为此问题提出的其他堆包括 d-heap<sup>[3]</sup>和 R-heap<sup>[6]</sup>。

(iii) 参数——在这种方法中，设计将侧重于某个很小的参数（或多个参数）。对于一个有趣的问题子集，该参数可能是边集的最大边长。参考文献[2]描述了如何使 Dijkstra 方法在  $O(m + n\sqrt{\log C})$  的时间内运行。

(iv) 输入限制——为绘制边长的情况设计了专用算法。这种算法的分析利用了这种分布并提供了估计，预期运行时间。

我们认为，这是第一个以不同边长种类，来区分运行时间的最短路论文。然而，这个参数，或是其他密切相关的参数，已经是其他算法问题分析的一部分。

## 5. Dijkstra 算法的 $O(m + nK)$ 实现

在本节中，我们提供了一个用于解决 SSSPP 的 Dijkstra 算法的 $O(m + nK)$ 实现。运行 Dijkstra 的算法时，我们保持以下结构：

- (i) 集合 $S$ 。指的是已被标记的顶点集合。
- (ii) 集合 $T = V - S$ 。指的是未被标记的顶点集合。

值 $d(j)$ 是顶点 $j$ 的距离标签。如果 $j \in S$ ，则 $d(j) = \delta(j)$ 是从顶点  $s$  到顶点  $j$  的最短路径长度。最后， $d^* = \max \{d(j) : j \in S\}$ 指最近被添加到 $S$ 集合的顶点距离标签。

当初步执行时，Dijkstra 算法中的瓶颈操作是 $FindMin()$ 操作，它标识在  $T$  中的顶点的最小距离标签。每个 $FindMin()$ 操作需要执行 $O(|T|)$ 次，因此 $FindMin()$ 操作总共执行 $O(n^2)$ 次。所有其他更新总共需要执行 $O(m)$ 次。为了减少 $FindMin()$ 的运行时间，Dijkstra 算法依赖于许多不同优先级队列中的一个。其中，具有最好的渐近边界是 Fibonacci Heap 实现<sup>[8]</sup>和 Atomic Heap 实现<sup>[9]</sup>。这里我们可以显着加快 $FindMin()$ 操作，在不同边长的数量很小的情况下。

我们令不同边长集合 $L = \{l_1, l_2, \dots, l_K\}$ 。对于每个 $t = 1$  to  $K$ ，算法将保持列表 $E_t(S) = \{(i, j) \in E; i \in S, c_{ij} = l_t\}$ 。这些边根据进入 $S$ 的顺序进行排序。若 $edge(i, j)$ 要覆盖已经在 $E_t(S)$ 中的 $edge(i', j')$ ，且 $d(i) \leq d(i')$ 。这个算法会维持一个 $CurrentEdge(t)$ 数组，指的是第一个 $E_t(S)$ 中的 $(i, j)$ 边。如果 $E_t(S)$ 中暂时不存在边，则 $CurrentEdge(t) = \emptyset$ 。如果 $(i, j) = CurrentEdge(t)$ ，那么我们让 $f(t) = d(i) + l_t$ 。这是顶点 $s$ 到顶点 $i$ 的最短路径长度加上 $edge(i, j)$ 。但这并不代表 $f(t) = d(j)$ ，因为可能有其他更短的路径指向 $j$ 。

这些附加数据结构使得可以通过确定 $K$ 来确定具有最小距离标签的  $T$  中的顶点 $\operatorname{argmin} \{f(t) : 1 \leq t \leq K\}$ 。如果没有任何优先级队列数据结构直接获取， $FindMin()$ 操作的时间是 $O(K)$ 。当 $K$ 很小时，能够让 Dijkstra 算法的总体运行时间获得改进。

子程序 $UPDATE(t)$ 移动指针 $CurrentEdge(t)$ ，使其指向结束点在 $T$ 内的第一条边。(或者设置 $CurrentEdge(t)$ 为空集)。如果 $CurrentEdge(t) = (i, j)$ ，

那么  $UPDATE(t)$  会令  $f(t) = d(i) + c_{ij}$ 。如果  $CurrentEdge(t) = \emptyset$ ，将会令  $f(t) = \infty$ 。操作符  $CurrentEdge(t).next$  将  $CurrentEdge$  指针在链表中移动一步来表示  $Et(S)$ 。(算法 5.1)

**定理 5.1** 算法 5.2 在  $O(m + nK)$  时间内确定从顶点  $s$  到所有其他顶点的最短路径。

---

**Function** *Initialize()*

```

1:  $S := \{s\}; T := V - \{s\}$ .
2:  $d(s) := 0; pred(s) := \emptyset$ .
3: for (each vertex  $v \in T$ ) do
4:  $d(v) = \infty; pred(v) = \emptyset$ .
5: end for
6: for ( $t = 1$  to  $K$ ) do
7:  $E_t(S) := \emptyset$ .
8:  $CurrentEdge(t) := NIL$ .
9: end for
10: for each edge  $(s, j)$  do
11: Add  $(s, j)$  to the end of the list  $E_t(S)$ , where  $l_t = c_{sj}$ .
12: if ( $CurrentEdge(t) = NIL$ ) then
13:  $CurrentEdge(t) := (s, j)$ 
14: end if
15: end for
16: for ( $t = 1$  to  $K$ ) do
17:  $Update(t)$ 
18: end for

```

---

算法 5.1 初始化过程

---

**Function** *New – Dijkstra()*

```

1: Initialize()
2: while ( $T \neq \emptyset$ ) do
3: let  $r = \operatorname{argmin}\{f(t) : 1 \leq t \leq K\}$ .

```

```

4: let ( $i, j$ ) = CurrentEdge( $r$ ).
5:  $d(j) := d(i) + l_r$ ;  $pred(j) := i$ .
6:  $S = S \cup \{j\}$ ;  $T := T - \{j\}$ .
7: for (each edge ( $j, k$ )  $\in E(j)$ ) do
8:   Add the end of the list  $E_t(S)$ , where  $l_t = c_{jk}$ .
9:   if (CurrentEdge( $t$ ) = NIL) then
10:    CurrentEdge( $t$ ) := ( $j, k$ )
11:   end if
12: end for
13: for ( $t = 1$  to  $K$ ) do
14:   Update( $t$ ).
15: end for
16: end while

```

---

算法 5.2 利用不同种类边长的 Dijkstra 算法

---

**Function** *Update*( $t$ )

```

1: Let ( $i, j$ ) = CurrentEdge( $t$ ).
2: if ( $j \in T$ ) then
3:    $f(t) = d(i) + c_{ij}$ 
4:   return
5: end if
6: while (( $j \notin T$ ) and (CurrentEdge( $t$ ).next = NIL)) do
7:   Let ( $i, j$ ) = CurrentEdge( $t$ ).next.
8:   CurrentEdge( $t$ ) = ( $i, j$ ).
9: end while
10: if ( $j \in T$ ) then
11:    $f(t) = d(i) + c_{ij}$ 
12: else
13:   Set CurrentEdge( $t$ ) to  $\emptyset$ .

```

14:  $f(t) = \infty$ .

15: **end if**

---

### 算法 5.3 更新过程

**证明。**该算法与 Dijkstra 算法相同，除了它维护一些附加的数据结构来携带输出  $FindMin()$  操作。因此，算法 5.2 正确地计算从顶点的最短路径。初始化需要  $O(n)$  时间。潜在的瓶颈操作是确定  $r = \operatorname{argmin}\{f(t): 1 \leq t \leq K\}$  和在所有迭代上执行  $Update(t)$  的时间。所有其他步骤的运行时间由这两个之一决定时间复杂度。我们首先注意到计算的时间（见算法 5.3）每次循环计算  $r = \operatorname{argmin}\{f(t): 1 \leq t \leq K\}$  的时间为  $O(K)$ ，而总循环所耗的时间为  $O(nK)$ 。

我们接下来考虑在所有迭代上执行  $Update(t)$  所需的时间。过程  $Update(t)$  需要  $O(nK)$  次，其总运行时间为  $O(m + nK)$ 。要看到这一点，首先要注意运行时间限制为迭代过程。其中  $CurrentEdge(t)$  不改变  $O(nK)$ 。我们现在考虑改变  $CurrentEdge(t)$  中的那些迭代。假设  $(i, j) = CurrentEdge(t)$ ，并假设  $i \in S$  和  $j \in S$ 。因为边  $E_t(S)$  被顺序地扫描，在更新  $CurrentEdge(t)$  之后，边缘  $(i, j)$  从未被再次扫描。所以， $CurrentEdge(t)$  的所有迭代运行时间为  $O(m)$ 。我们得出结论，算法 5.2 的总运行时间是  $O(m + nK)$ 。本文的原始动机是  $K = 2$  的情况，在这种情况下算法特别有效。在下一节中，我们将展示如何在  $K$  随问题大小增长的情况下加快算法。

## 6. 一个更快的算法，如果 $K$ 允许随问题大小增长

我们现在修改算法以提高其运行时间，在  $K$  不是常数的情况下。我们让  $q = \frac{nK}{m}$ 。我们假设  $q \geq 2$ ；如果不是，算法 5.2 在线性时间运行。为了简化说明，我们假设  $q$  是一个整数  $K$  的约数，我们让  $h = \frac{K}{q}$ 。由于我们专注于渐近分析，我们可以做出这个假设而没有损失概论。

我们可令算法 5.2 被细化以在时间  $O(m \log q)$  中运行。为了达到这个运行时间，我们需要加速依赖于  $K$  的算法 5.2 中的瓶颈。我们需要更高效地计算  $r$ ，我们需要以较少频率调用  $Update(t)$ 。我们首先解决加速  $r$  的计算。

为了加快 $r$ 的确定，我们将值 $f()$ 存储在 $h$ 个不同二进制堆的集合中。第一二进制堆存储用于 $j = 1$ 至 $q$ 的值 $f(j)$ ，第 2 个二进制堆存储用于 $j = q + 1$ 至 $2q$ 的值 $f(j)$ 等。我们将堆表示为 $H_1, H_2, \dots, H_h$ 。在二进制堆  $H_i$  中找到具有最小值的元素 $O(1)$ 步骤。将元素插入  $H_i$  或从  $H_i$  中删除元素需要 $O(\log q)$ 个步长。

我们首先通过在每个 $h$ 堆找到具有最小值的元素，然后找到最小  $f()$ 值来选择这些 $h$ 元素中最好的。在堆中找到最小元素需要 $O(1)$ 步；所以这个实现的时间的 $FindMin()$ 操作是 while 循环的每次迭代的 $O(h)$ ，以及 $O(hn) = O(m)$ 。发现后最小元素在  $h$  不同的堆中，我们从其堆中删除元素。这需要 $O(n \log q)$ 个步骤迭代。

我们现在解决在 $Update()$ 中花费的时间。为了最小化 $Update()$ 所需的时间，我们首先放松对 $CurrentEdge(t)$ 的要求。如果 $CurrentEdge(t)$ 的两个端点都在  $S$  中，我们说 $CurrentEdge(t)$ 无效。我们允许 $CurrentEdge(t)$ 在算法的某些中间阶段无效。然后我们再次修改 $FindMin()$ 。如果堆  $H_i$  中的最小元素对于一些  $i$  是 $f(t)$ ，则算法执行 $Update()$ 。然后在 $H_i$ 中找到新的最小元素。它以这种方式迭代，直到最小元素对应到有效边缘。这样，每当算法调用 $Update()$ 时，都会导致对 $CurrentEdge()$ 的修改。此外，每当算法选择 $q$ 个不同堆中的最小元素时，每个最小元素堆对应于有效边缘。由于对 $CurrentEdge()$ 的每次修改都会导致 $a$ 中某个值的更改堆，并且由于对  $CurrentEdge()$  有至多  $m$  个修改，因此  $Update()$  在所有迭代中的总运行时间是 $O(m \log q)$ 。

我们总结前面的讨论与定理 6.1。

**定理 6.1** 二进制堆实现 Dijkstra 的算法，需要 $O(Kq)$ 个大小为 $O(q)$ 的二进制堆，其中 $q = \frac{nK}{m}$ 。确定在 $O(m \log q)$ 时间中从顶点 $s$ 到所有其他顶点的最短路径。

## 7. 实证结果

### 7.1. 实验配置

我们在几个系列的图中评估算法性能。一些生成器和图实例是部分 DIMACS 最短路径实现图<sup>[5]</sup>。

随机图：我们根据鄂尔多斯-仁的随机图模型生成图形，并确保图形是连接的。发生器可以产生平行边缘以及自循环。边的数量与可以改变顶点的数量，并且我们对密集和稀疏图形进行实验。随机图有一个低直径和高斯分布。

网格图：这种合成发电机产生具有网格维度 $x$ 和 $y$ 的二维网格。我们生成直接( $x = \frac{n}{16}$ ,  $y = 16$ )和方形网格( $x = y = \sqrt{n}$ )。这些图的直径比随机图和所有顶点更具有恒定度。

小世界图：我们使用 R-MAT 图模型为现实世界网络<sup>[4]</sup>生成小世界图特性。这些图具有低直径和不平衡度分布。

边权重从一组固定的不同随机整数中选择，因为我们的新算法是为网络设计的小的参数 $K$ 。

我们将我们的算法的执行时间与第 9 个 DIMACS 最短中使用的参考 SSSPP 求解器进行比较路径挑战（Goldberg 算法的高效实现<sup>[11, 13]</sup>，它具有预期的线性运行时间，并针对整数边缘权重高度优化）和每个图族上的基线宽度优先搜索（BFS）。BFS 运行时间是 SSSPP 实现的自然下界。直接执行 times 的 DIMACS 参考解算器代码，都使用类似的邻接数组表示图形，用 C/C++ 编写，并在相同的实验设置下编译和运行。注意我们的实现可以具有实数和整数权重的过程图，但仅对具有几个明显边缘的网络有效。我们用在本文中只有整数权重用于与 DIMACS 求解器比较。我们的性能测试平台是一个 2.8 GHz 32 位 Intel Xeon 机器，具有 4 GB 内存，512 KB 缓存并运行 RedHat Enterprise Linux 4（linux 内核 2.6.9）。我们比较我们实现的顺序性能与 DIMACS 参考解算器<sup>[5]</sup>。这两个代码都是使用 Intel C 编译器（icc）9.0 版本编译的优化标志-O3。我们报告每个实验的五次独立运行的平均执行时间。

## 7.2 结果和分析

我们进行了广泛的研究，经验性地评估对图形拓扑结构的性能依赖性，问题尺寸， $K$  的值和边缘权重分布。在表 1 和 2 所研究的实例图中，我们报告了所有的宽度优先搜索的执行时间。这些图绘制了最短路径实现的执行时间归一化到 BFS 时间。因此，1.0 的比率是我们可以实现的最好的，并且较小的值是期望的。

表 1 宽度优先搜索测试顺序平台上各种图系列的执行时间（以毫秒为单位）。

Problem instance		BFS time (milliseconds)		
ID	Graph size	Long mesh	Square mesh	Random
1	100K vertices, 400K edges	50	55	95
2	500K vertices, 2M edges	290	350	540
3	1M vertices, 4M edges	660	870	1180
4	5M vertices, 20M edges	4160	6400	8390
5	10M vertices, 40M edges	8590	13 500	17 980

表 2 宽度优先搜索各种图形的搜索执行时间（以毫秒为单位， $K$  在实验中变化）

Problem instance		BFS time (milliseconds)
1	Sparse random, 2M vertices, 8M edges, $C = 10\,000$	6430
2	Dense random, 100K vertices, 100M edges, $C = 100$	150
3	Long mesh, 2M vertices, 8M edges, $C = 100$	3260
4	Square mesh, 2M vertices, 8M edges, $C = 100$	4900
5	Small-world graph, 2M vertices, 8M edges, $C = 10\,000$	5440



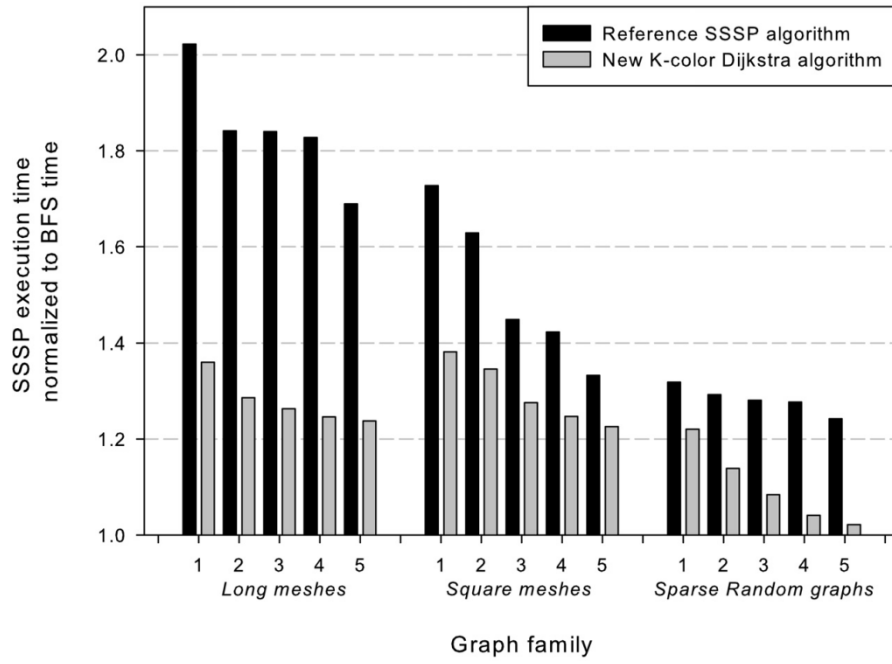


图 1 对于三个图族，展现我们的最短结果和参考求解器的性能，因为问题的大小是变化的。图族 1 对应我们研究中最小的网络，图族 5 是最大的。

我们首先研究问题规模对执行时间的影响。我们改变三个不同的图族的数量级大小，并计算参考代码和我们的实现的平均 SSSP 执行时间。图 1 绘制了不同图谱族的这些归一化值。问题大小列在表 1 中。 $K$  的值在这种情况下设置为 2，并且图中的最大边缘权重与最小边缘权重的比率（由  $C$  表示）被设置为 100。

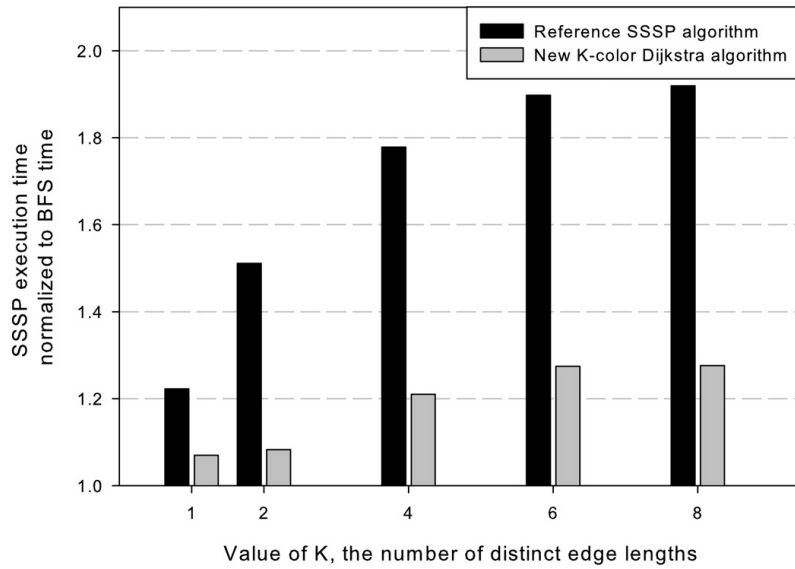


图 2 随  $K$  变化的稀疏随机图（4 百万个顶点，1600 万个边）SSSP 性能。

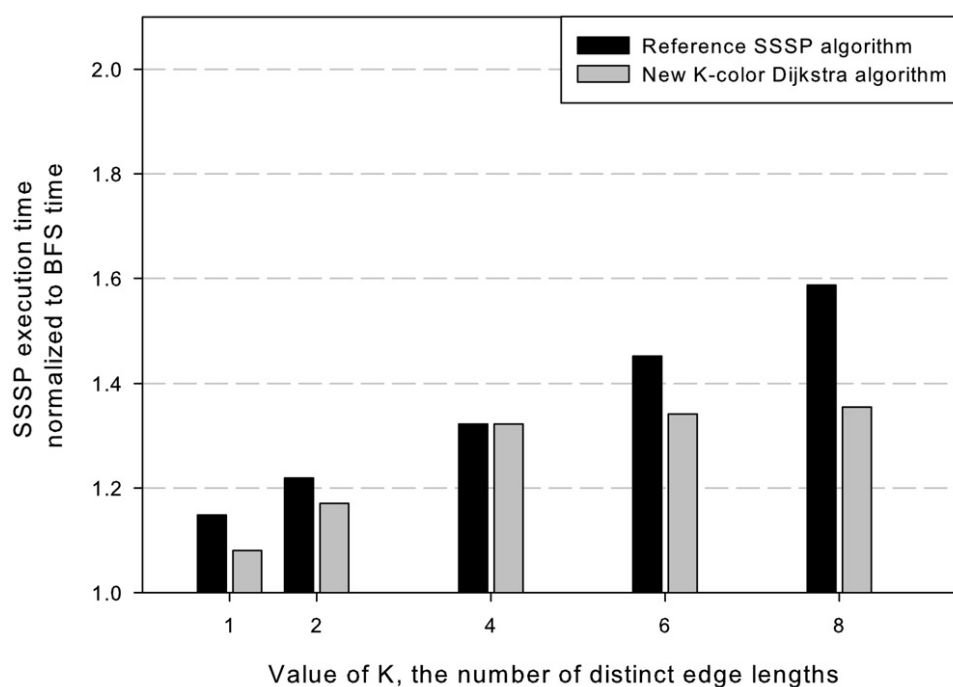


图 3 随 K 变化的小图（4 百万个顶点，1600 万个边）SSSP 性能。

注意在所有情况下比值  $m/n$  为 4:1，我们观察到我们的新算法胜过参考求解器。此外，在大多数情况下，性能比小于 2，这是相当显著的。仔细观察，我们观察到长方形网格图的性能改进是相对地高于随机图。我们认为这是因为我们不维护优先级队列数据结构。从而我们避免了在网状网络中评估长路径时涉及的优先级队列开销，例如频繁更新。

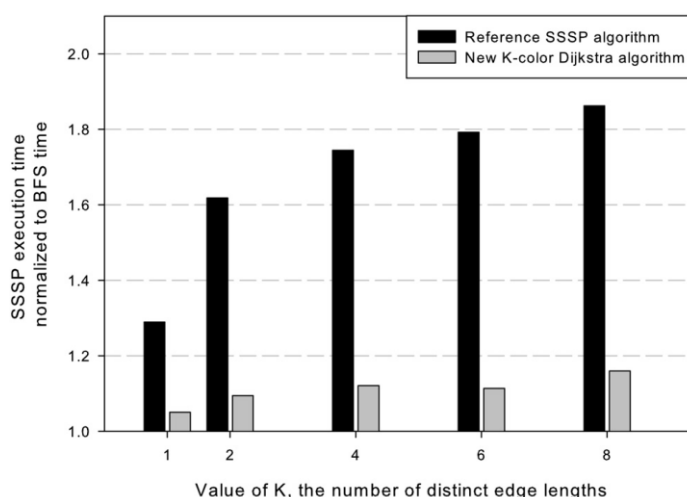


图 4 随着 K 的值的变化的，对应长网格图（400 万个顶点，1600 万个边）SSSP 性能。

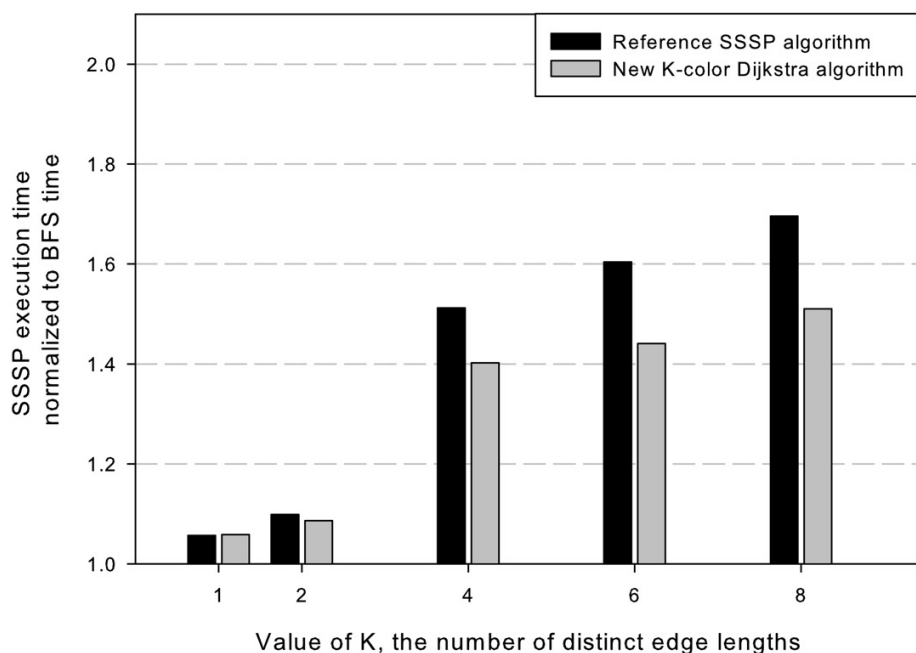


图 5 随着  $K$  值变化，针对正方形网格图（4 百万个顶点，16 百万个边）的 SSSPP 性能。

我们还观察到，与较小的图像相比，较大的图像中算法的性能更好。稀疏随机网络（1.02-1.22）的性能对比在我们研究的例子中效果特别明显。

我们接下来研究算法中  $K$  值对每个图族的性能影响，明显边长的数量权重是变化的。我们在每种情况下将  $K$  的值从 1 改为 8，并绘制执行时间。

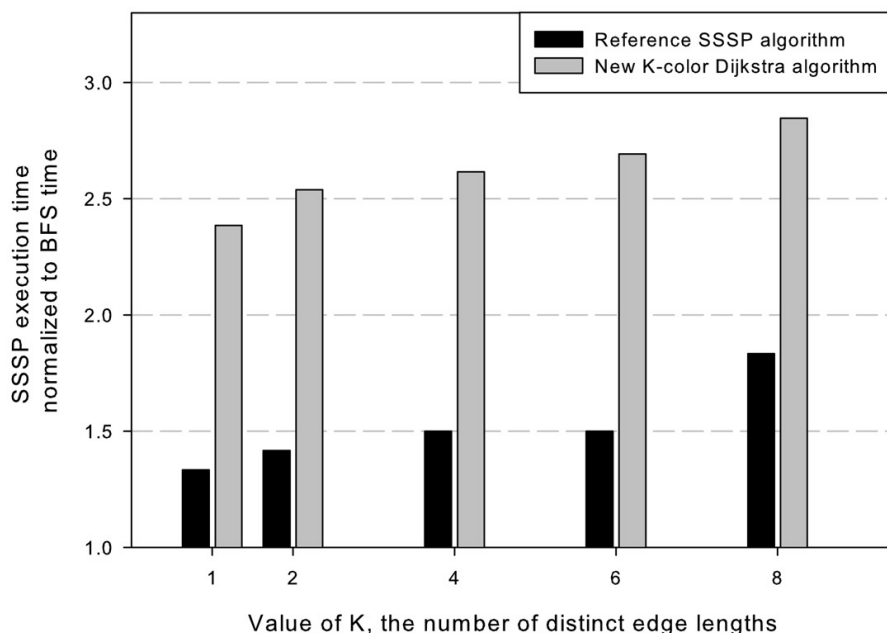


图 6 随着  $K$  值的变化，密集随机图（100K 个顶点，1000 万个边）的 SSSPP 性能。

我们观察到我们的算法明显快于参考求解器（对于  $K = 8$ ，速度高达 70%），并且随着  $K$  的增加，效果愈发明显。随机图的 SSSP 运行时间相对低于其他图的

执行时间。

图 3 绘制了小规模图的标准化执行时间。这些图类似于低直径稀疏随机网络，但也具有在大规模社会和生物网络中观察到的拓扑性质。在这种情况下，我们观察到参考求解器和我们的算法执行非常相似。

对于长网格（图 4），我们观察到参考求解器的显著性能增益。另外，当  $K$  增加时，我们算法的运行时间比参考求解器变慢一些。

图 6 绘制了随着  $K$  值的变化，100K 个顶点和 1 亿个边的密集随机图的标准化执行时间。我们观察到随着  $K$  值的变化，参考求解器比我们的算法变化更快。我们将这归因于 BFS 的快速执行时间（因为这是一个相对较小的图实例）。在这种情况下，执行我们的算法的开销很高，在这个大小的密集图下，它的性能不如参考求解器。

所有这些图形实例的 BFS 的执行时间在表 2 中列出。图 C 中还列出了权重的最高和最低边的比率，因为参考求解器的最坏情况运行时间取决于该值，故我们的算法性能是独立于图 C 的。

## 8. 结论

SSSPP 是计算机科学和运营研究中的一个根本问题。在本文中，我们在边长的数量  $K$  较小的情况下研究 SSSPP。当不同边长数量小于图的密度时，我们的算法运行在线性时间内。

## 参考文献

- [1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows: Theory, Algorithms and Applications, Prentice-Hall, 1993.
- [2] R.K. Ahuja, K. Mehlhorn, J.B. Orlin, R.E. Tarjan, Faster algorithms for the shortest path problem, J. ACM 37 (2) (April 1990) 213–223.
- [3] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, MIT Press, 2001.
- [4] D. Chakrabarti, Y. Zhan, C. Faloutsos, R-MAT: A recursive model for graph mining, in: Proc. 4th SIAM Intl. Conf. on Data Mining, Florida, USA, April, 2004.
- [5] C. Demetrescu, A.V. Goldberg, D. Johnson, 9th DIMACS implementation challenge – Shortest Paths, <http://www.dis.uniroma1.it/challenge9/>, 2005.
- [6] J.R. Driscoll, H.N. Gabow, R. Shrairman, R.E. Tarjan, Relaxed heaps: An alternative to Fibonacci heaps with applications to parallel computation, Commun.ACM 31 (11) (1988) 1343–1354.
- [7] E.W. Dijkstra, A note on two problems in connexion with graphs, Numer. Math. 1 (1959) 269–271.
- [8] M.L. Fredman, R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, J. ACM 34 (3) (1987) 596–615.
- [9] M.L. Fredman, D.E. Willard, Trans-dichotomous algorithms for minimum spanning trees and shortest paths, J. Comput. Syst. Sci. 48 (3) (1994) 533–551.
- [10] A.V. Goldberg, A simple shortest path algorithm with linear average time, in: ESA, 2001, pp. 230–241.
- [11] A.V. Goldberg, A simple shortest path algorithm with linear average time, in: 9th Ann. European Symp. on Algorithms (ESA 2001), Aachen, Germany, in: Lecture Notes in Computer Science, vol. 2161, Springer, 2001, pp. 230–241.
- [12] U. Meyer, Single-source shortest-paths on arbitrary directed graphs in linear average-case time, in: Proceedings of the Twelfth Annual ACM–SIAM Symposium on Discrete Algorithms (SODA-01), New York, January 7–9, 2001, ACM Press, 2001, pp. 797–806.

# On the Shoshan-Zwick Algorithm for the All-Pairs Shortest Path Problem

Pavlos Eirinakis \*

Athens University of Economics and Business,  
Athens, Greece  
{peir@aueb.gr}

Matthew Williamson †

West Virginia University Institute of Technology  
Montgomery, WV, USA  
{matthew.williamson@mail.wvu.edu}

K. Subramani †

West Virginia University,  
Morgantown, WV, USA  
{ksmani@csee.wvu.edu}

## Abstract

The Shoshan-Zwick algorithm solves the all pairs shortest paths problem in undirected graphs with integer edge costs in the range  $\{1, 2, \dots, M\}$ . It runs in  $\tilde{O}(M \cdot n^\omega)$  time, where  $n$  is the number of vertices,  $M$  is the largest integer edge cost, and  $\omega < 2.3727$  is the exponent of matrix multiplication. It is the fastest known algorithm for this problem. This paper points out the erroneous behavior of the Shoshan-Zwick algorithm and revises the algorithm to resolve the issues that cause this behavior. Moreover, it discusses implementation aspects of the Shoshan-Zwick algorithm using currently-existing sub-cubic matrix multiplication algorithms.

## 1 Introduction

The Shoshan-Zwick algorithm [22] solves the all-pairs shortest paths (APSP) problem in undirected graphs, where the edge costs are integers in the range  $\{1, 2, \dots, M\}$ . This is accomplished by computing  $O(\log(M \cdot n))$  distance products of  $n \times n$  matrices with elements in the range  $\{1, 2, \dots, M\}$ . The algorithm runs in  $\tilde{O}(M \cdot n^\omega)$  time, where  $\omega < 2.3727$  is the exponent for the fastest known matrix multiplication algorithm [29].

The APSP problem is a fundamental problem in algorithmic graph theory. Consider a graph with  $n$  nodes and  $m$  edges. For directed or undirected graphs with real edge costs, we can use known methods that run in  $O(m \cdot n + n^2 \cdot \log n)$  time [8, 11, 17] and  $O(n^3)$  time [10]. Sub-cubic APSP algorithms have been obtained that run in  $O(n^3 \cdot ((\log \log n)/\log n)^{1/2})$  time [12],  $O(n^3 \cdot \sqrt{\log \log n / \log n})$  time [24],  $O(n^3/\sqrt{\log n})$  time [9],  $O(n^3 \cdot (\log \log n / \log n)^{5/7})$  time [15],  $O(n^3 \cdot (\log \log n)^2 \log n)$  time [25],  $O(n^3 \cdot \log \log n / \log n)$  time [26],  $O(n^3 \cdot \sqrt{\log \log n} / \log n)$  time [31],  $O(n^3 / \log n)$  time [3], and  $O(n^3 \cdot \log \log n / \log^2 n)$  time [16]. For

\*This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thales. Investing in knowledge society through the European Social Fund. (MIS: 380232)

†This research was supported in part by the National Science Foundation and Air Force of Scientific Research through Awards CNS-0849735, CCF-1305054, and FA9550-12-1-0199.

undirected graphs with integer edge costs, the problem can be solved in  $O(m \cdot n)$  time [27, 28]. Fast matrix multiplication algorithms can also be used for solving the APSP problem in dense graphs with small integer edge costs. [13, 21] provide algorithms that run in  $\tilde{O}(n^\omega)$  time in unweighted, undirected graphs, where  $\omega < 2.3727$  is the exponent for matrix multiplication [29]. [4] improves this result with an algorithm that runs in  $o(m \cdot n)$  time.

In this paper, we revise the Shoshan-Zwick algorithm to resolve some issues that cause the algorithm to behave erroneously. This behavior was identified when, in the process of implementing the algorithm as part of a more elaborate procedure (i.e., identifying negative cost cycles in undirected graphs), we discovered that the algorithm is not producing correct results. Since the Shoshan-Zwick algorithm is incorrect in its current version, any result that uses this algorithm as a subroutine is also incorrect. For instance, the results provided by Alon and Yuster [2], Cygan et. al [7], W. Liu et al [19], and Yuster [30] all depend on the Shoshan-Zwick algorithm. Thus, their results are currently incorrect. By applying our revision to the algorithm, the issues with the above results are resolved. We also discuss issues concerning the implementation of the Shoshan-Zwick algorithm using known sub-cubic matrix multiplication algorithms.

The principal contributions of this paper are as follows:

- (i) A counter-example that shows that the Shoshan-Zwick algorithm is incorrect in its current form.
- (ii) A detailed explanation of where and why the algorithm fails.
- (iii) A modified version of the Shoshan-Zwick algorithm that corrects the problems with the previous version. The corrections do not affect the time complexity of the algorithm.
- (iv) A discussion concerning implementing the matrix multiplication subroutine that is used in the algorithm.

The rest of the paper is organized as follows. We describe the Shoshan-Zwick algorithm in Section 2. Section 3 establishes that the published version of the algorithm is incorrect by providing a simple counter-example. The origins of this erroneous behavior is identified in Section 4. In Section 5, we present a revised version of the algorithm and provide a formal proof of correctness. In Section 6, we discuss the efficacy of the Shoshan-Zwick algorithm. Section 7 concludes the paper by summarizing our contributions and discussing avenues for future research.

## 2 The Shoshan-Zwick Algorithm

In this section, we review the Shoshan-Zwick algorithm for the APSP problem in undirected graphs with integer edge costs. Consider a graph  $G = (V, E)$ , where  $V = \{0, 1, 2, \dots, n\}$  is the set of nodes, and  $E$  is the set of edges. The graph is represented as an  $n \times n$  matrix  $\mathbf{D}$ , where  $d_{ij}$  is the cost of edge  $(i, j)$  if  $(i, j) \in E$ ,  $d_{ii} = 0$  for  $1 \leq i \leq n$ , and  $d_{ij} = +\infty$  otherwise. Note that, without loss of generality, the edge costs are taken from the range  $\{1, 2, \dots, M\}$ , where  $M = 2^m$  for some  $m \geq 1$ .

The algorithm computes a logarithmic number of distance products in order to determine the shortest paths. Let  $\mathbf{A}$  and  $\mathbf{B}$  be two  $n \times n$  matrices. Their distance product  $\mathbf{A} \star \mathbf{B}$  is an  $n \times n$  matrix such that:

$$(\mathbf{A} \star \mathbf{B})_{ij} = \min_{k=1}^n \{a_{ik} + b_{kj}\}, 1 \leq i, j \leq n.$$

The distance product of two  $n \times n$  matrices with elements within the range  $\{1, 2, \dots, M, +\infty\}$  can be computed in  $\tilde{O}(M \cdot n^\omega)$  time [1]. The distance product of two matrices whose finite elements are taken from  $\{1, 2, \dots, M\}$  is a matrix whose finite elements are taken from  $\{1, 2, \dots, 2 \cdot M\}$ . However, the Shoshan-Zwick algorithm is based on not allowing the range of elements in the matrices it uses to increase. Hence, if  $\mathbf{A}$  is an  $n \times n$  matrix, and  $a, b$  are two numbers such that  $a \leq b$ , the algorithm utilizes two operations, namely  $\text{clip}(\mathbf{A}, a, b)$  and  $\text{chop}(\mathbf{A}, a, b)$ , that produce corresponding  $n \times n$  matrices such that

$$\begin{aligned} (\text{clip}(\mathbf{A}, a, b))_{ij} &= \begin{cases} a & \text{if } a_{ij} < a \\ a_{ij} & \text{if } a \leq a_{ij} \leq b \\ +\infty & \text{if } a_{ij} > b \end{cases} \\ (\text{chop}(\mathbf{A}, a, b))_{ij} &= \begin{cases} a_{ij} & \text{if } a \leq a_{ij} \leq b \\ +\infty & \text{otherwise} \end{cases}. \end{aligned}$$

The algorithm also defines  $n \times n$  matrices  $\mathbf{A} \wedge \mathbf{B}$ ,  $\mathbf{A} \bar{\wedge} \mathbf{B}$ , and  $\mathbf{A} \vee \mathbf{B}$  such that

$$\begin{aligned} (\mathbf{A} \wedge \mathbf{B})_{ij} &= \begin{cases} a_{ij} & \text{if } b_{ij} < 0 \\ +\infty & \text{otherwise} \end{cases} \\ (\mathbf{A} \bar{\wedge} \mathbf{B})_{ij} &= \begin{cases} a_{ij} & \text{if } b_{ij} \geq 0 \\ +\infty & \text{otherwise} \end{cases} \\ (\mathbf{A} \vee \mathbf{B})_{ij} &= \begin{cases} a_{ij} & \text{if } a_{ij} \neq +\infty \\ b_{ij} & \text{if } a_{ij} = +\infty, b_{ij} \neq +\infty \\ +\infty & \text{if } a_{ij} = b_{ij} = +\infty \end{cases}. \end{aligned}$$

Finally, if  $\mathbf{C} = (c_{ij})$  and  $\mathbf{P} = (p_{ij})$  are matrices, the algorithm defines the Boolean matrices  $(\mathbf{C} \geq 0)$  and  $(0 \leq \mathbf{P} \leq M)$  such that

$$\begin{aligned} (\mathbf{C} \geq 0)_{ij} &= \begin{cases} 1 & \text{if } c_{ij} \geq 0 \\ 0 & \text{otherwise} \end{cases} \\ (0 \leq \mathbf{P} \leq M)_{ij} &= \begin{cases} 1 & \text{if } 0 \leq p_{ij} \leq M \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

Using the definitions above, the Shoshan-Zwick algorithm computes the shortest paths by finding the  $\lceil \log_2 n \rceil$  most significant bits of each distance and the remainder that each distance leaves modulo  $M$ . This provides enough information to reconstruct the distances. The procedure is shown in Algorithm 2.1, and the proof of correctness is given in [22, Lemma 3.6].

The algorithm computes  $O(\log n + \log M) = O(\log(M \cdot n))$  distance products of matrices, whose finite



elements are in the range  $\{0, 1, 2, \dots, 2 \cdot M\}$  or in the range  $\{-M, \dots, M\}$ . Note that each distance product can be reduced to a constant number of distance products of matrices with elements in the range  $\{1, 2, \dots, M\}$ . All other operations in the algorithm take  $O(n^2 \cdot (\log n + \log M))$  time. Therefore, the total running time of the algorithm is  $\tilde{O}(M \cdot n^\omega)$  time [22, Theorem 3.7].

**Function SHOSHAN-ZWICK-APSP( $\mathbf{D}$ )**

```

1:  $l = \lceil \log_2 n \rceil$ .
2:  $m = \log_2 M$ .
3: for ( $k = 1$  to  $m + 1$ ) do
4:    $\mathbf{D} = \text{clip}(\mathbf{D} \star \mathbf{D}, 0, 2 \cdot M)$ .
5: end for
6:  $\mathbf{A}_0 = \mathbf{D} - M$ .
7: for ( $k = 1$  to  $l$ ) do
8:    $\mathbf{A}_k = \text{clip}(\mathbf{A}_{k-1} \star \mathbf{A}_{k-1}, -M, M)$ .
9: end for
10:  $\mathbf{C}_l = -M$ .
11:  $\mathbf{P}_l = \text{clip}(\mathbf{D}, 0, M)$ .
12:  $\mathbf{Q}_l = +\infty$ .
13: for ( $k = l - 1$  down to  $0$ ) do
14:    $\mathbf{C}_k = [\text{clip}(\mathbf{P}_{k+1} \star \mathbf{A}_k, -M, M) \wedge \mathbf{C}_{k+1}] \vee [\text{clip}(\mathbf{Q}_{k+1} \star \mathbf{A}_k, -M, M) \bar{\wedge} \mathbf{C}_{k+1}]$ .
15:    $\mathbf{P}_k = \mathbf{P}_{k+1} \vee \mathbf{Q}_{k+1}$ .
16:    $\mathbf{Q}_k = \text{chop}(\mathbf{C}_k, 1 - M, M)$ .
17: end for
18: for ( $k = 1$  to  $l$ ) do
19:    $\mathbf{B}_k = (\mathbf{C}_k \geq 0)$ .
20: end for
21:  $\mathbf{B}_0 = (0 \leq \mathbf{P}_0 < M)$ .
22:  $\mathbf{R} = \mathbf{P}_0 \bmod M$ .
23:  $\Delta = M \cdot \sum_{k=0}^l 2^k \cdot \mathbf{B}_k + \mathbf{R}$ .
24: return  $\Delta$ .
```

**Algorithm 2.1:** Shoshan-Zwick APSP Algorithm

### 3 A Counter-Example

In this section, we provide a detailed presentation of the counter-example presented that shows that the Shoshan-Zwick algorithm is incorrect as presented in [22]. Consider the APSP problem with respect to graph  $G'$  presented in Figure 1.

Graph  $G'$  can also be represented as a  $3 \times 3$  matrix:

$$\mathbf{D} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix}.$$

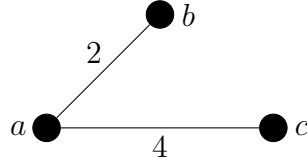


Figure 1: Graph  $G'$  of the counter-example for the Shoshan-Zwick algorithm.

We now walk through each step of the algorithm. First,  $l = \lceil \log_2 3 \rceil = 2$ , and  $m = \log_2 4 = 2$ . This means that in the **for** loop in lines 3 to 5, we set  $\mathbf{D} = \text{clip}(\mathbf{D} \star \mathbf{D}, 0, 8)$  three times. At each such step, we get

$$\begin{aligned}
 \mathbf{D} \star \mathbf{D} &= \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix} \star \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix}, & \mathbf{D} &= \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} \text{ (for } k=1), \\
 \mathbf{D} \star \mathbf{D} &= \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} \star \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix}, & \mathbf{D} &= \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} \text{ (for } k=2), \\
 \mathbf{D} \star \mathbf{D} &= \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} \star \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix}, & \mathbf{D} &= \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} \text{ (for } k=3).
 \end{aligned}$$

The next step is to set  $\mathbf{A}_0 = \mathbf{D} - \mathbf{M}$ , which gives us

$$\mathbf{A}_0 = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} - 4 = \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix}.$$

In the **for** loop in lines 7 to 9, we compute  $\mathbf{A}_k = \text{clip}(\mathbf{A}_{k-1} \star \mathbf{A}_{k-1}, -4, 4)$  for  $k = 1$  and  $k = 2$ . This gives us

$$\begin{aligned}
 \mathbf{A}_0 \star \mathbf{A}_0 &= \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix} \star \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix} = \begin{bmatrix} -8 & -6 & -4 \\ -6 & -8 & -2 \\ -4 & -2 & -8 \end{bmatrix}, \mathbf{A}_1 = \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix}, \\
 \mathbf{A}_1 \star \mathbf{A}_1 &= \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} \star \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} = \begin{bmatrix} -8 & -8 & -8 \\ -8 & -8 & -8 \\ -8 & -8 & -8 \end{bmatrix}, \mathbf{A}_2 = \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -4 \\ -4 & -4 & -4 \end{bmatrix}.
 \end{aligned}$$

In lines 10 to 12, we set  $\mathbf{C}_2 = -4$ ,  $\mathbf{P}_2 = \text{clip}(\mathbf{D}, 0, 4)$ , and  $\mathbf{Q}_2 = +\infty$ . Thus, we have

$$\mathbf{C}_2 = \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -4 \\ -4 & -4 & -4 \end{bmatrix}, \mathbf{P}_2 = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix}, \mathbf{Q}_2 = \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix}.$$

We now compute the **for** loop in lines 13 to 17. This means we run the lines

$$\mathbf{C}_k = [\text{clip}(\mathbf{P}_{k+1} \star \mathbf{A}_k, -4, 4) \bigwedge \mathbf{C}_{k+1}] \bigvee [\text{clip}(\mathbf{Q}_{k+1} \star \mathbf{A}_k, -4, 4) \bar{\bigwedge} \mathbf{C}_{k+1}],$$

$$\mathbf{P}_k = \mathbf{P}_{k+1} \bigvee \mathbf{Q}_{k+1}, \text{ and}$$

$$\mathbf{Q}_k = \text{chop}(\mathbf{C}_k, -3, 4)$$

twice, i.e., for  $k = 1$  and  $k = 0$ . After this loop, we get

$$(\text{for } k = 1) \quad \mathbf{P}_2 \star \mathbf{A}_1 = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix} \star \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} = \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix},$$

$$\text{clip}(\mathbf{P}_2 \star \mathbf{A}_1, -4, 4) = \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix},$$

$$\text{clip}(\mathbf{P}_2 \star \mathbf{A}_1, -4, 4) \bigwedge \mathbf{C}_2 = \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} \bigwedge \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -4 \\ -4 & -4 & -4 \end{bmatrix} = \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix},$$

$$\mathbf{Q}_2 \star \mathbf{A}_1 = \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix} \star \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} = \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix},$$

$$\text{clip}(\mathbf{Q}_2 \star \mathbf{A}_1, -4, 4) = \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix},$$

$$\text{clip}(\mathbf{Q}_2 \star \mathbf{A}_1, -4, 4) \bar{\bigwedge} \mathbf{C}_2 = \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix} \bar{\bigwedge} \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -4 \\ -4 & -4 & -4 \end{bmatrix} = \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix},$$

$$\mathbf{C}_1 = \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} \bigvee \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix} = \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix},$$

$$\mathbf{P}_1 = \mathbf{P}_2 \bigvee \mathbf{Q}_2 = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix} \bigvee \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix},$$

$$\mathbf{Q}_1 = \text{chop}(\mathbf{C}_1, -3, 4) = \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & -2 \\ \infty & -2 & \infty \end{bmatrix},$$

$$(\text{and for } k = 0) \quad \mathbf{P}_1 \star \mathbf{A}_0 = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix} \star \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix} = \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix},$$

$$\text{clip}(\mathbf{P}_1 \star \mathbf{A}_0, -4, 4) = \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix},$$

$$\begin{aligned}
clip(\mathbf{P}_1 \star \mathbf{A}_0, -4, 4) \bar{\wedge} \mathbf{C}_1 &= \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix} \bar{\wedge} \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} = \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix}, \\
\mathbf{Q}_1 \star \mathbf{A}_0 &= \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & -2 \\ \infty & -2 & \infty \end{bmatrix} \star \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix} = \begin{bmatrix} \infty & \infty & \infty \\ -2 & 0 & -6 \\ -4 & -6 & -4 \end{bmatrix}, \\
clip(\mathbf{Q}_1 \star \mathbf{A}_0, -4, 4) &= \begin{bmatrix} \infty & \infty & \infty \\ -2 & 0 & -4 \\ -4 & -4 & -4 \end{bmatrix}, \\
clip(\mathbf{Q}_1 \star \mathbf{A}_0, -4, 4) \bar{\wedge} \mathbf{C}_1 &= \begin{bmatrix} \infty & \infty & \infty \\ -2 & 0 & -4 \\ -4 & -4 & -4 \end{bmatrix} \bar{\wedge} \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} = \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix}, \\
\mathbf{C}_0 &= \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix} \vee \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix} = \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix}, \\
\mathbf{P}_0 = \mathbf{P}_1 \vee \mathbf{Q}_1 &= \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix} \vee \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & -2 \\ \infty & -2 & \infty \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & -2 \\ 4 & -2 & 0 \end{bmatrix}, \\
\mathbf{Q}_0 = chop(\mathbf{C}_0, -3, 4) &= \begin{bmatrix} \infty & -2 & \infty \\ -2 & \infty & 2 \\ \infty & 2 & \infty \end{bmatrix}.
\end{aligned}$$

In the **for** loop in lines 18 to 20, we set  $\mathbf{B}_k = (\mathbf{C}_k \geq 0)$  for  $k = 1$  and  $k = 2$ . This means

$$\begin{aligned}
\mathbf{B}_1 = (\mathbf{C}_1 \geq 0) &= \left( \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} \geq 0 \right) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \\
\mathbf{B}_2 = (\mathbf{C}_2 \geq 0) &= \left( \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -4 \\ -4 & -4 & -4 \end{bmatrix} \geq 0 \right) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.
\end{aligned}$$

We then set  $\mathbf{B}_0 = (0 \leq \mathbf{P}_0 < 4)$ ,  $\mathbf{R} = \mathbf{P}_0 \bmod 4$ , and  $\Delta = 4 \cdot \sum_{k=0}^2 2^k \cdot \mathbf{B}_k + \mathbf{R}$  according to lines 21 to 23:

$$\begin{aligned}
\mathbf{B}_0 = (0 \leq \mathbf{P}_0 < 4) &= \left( 0 \leq \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & -2 \\ 4 & -2 & 0 \end{bmatrix} < 4 \right) = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \\
\mathbf{R} = \mathbf{P}_0 \bmod 4 &= \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & -2 \\ 4 & -2 & 0 \end{bmatrix} \bmod 4 = \begin{bmatrix} 0 & 2 & 0 \\ 2 & 0 & -2 \\ 0 & -2 & 0 \end{bmatrix},
\end{aligned}$$

$$\begin{aligned}
\Delta &= 4 \cdot \sum_{k=0}^2 2^k \cdot \mathbf{B}_k + \mathbf{R} \\
&= 4 \cdot \left( 2^0 \cdot \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + 2^1 \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + 2^2 \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right) + \begin{bmatrix} 0 & 2 & 0 \\ 2 & 0 & -2 \\ 0 & -2 & 0 \end{bmatrix} \\
&= 4 \cdot \left( \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right) + \begin{bmatrix} 0 & 2 & 0 \\ 2 & 0 & -2 \\ 0 & -2 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 4 & 4 & 0 \\ 4 & 4 & 0 \\ 0 & 0 & 4 \end{bmatrix} + \begin{bmatrix} 0 & 2 & 0 \\ 2 & 0 & -2 \\ 0 & -2 & 0 \end{bmatrix} = \begin{bmatrix} 4 & 6 & 0 \\ 6 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix}
\end{aligned}$$

The algorithm terminates by returning  $\Delta$  in line 24. The resulting shortest path costs are presented in Table 1. However, if we examine graph  $G'$  (Figure 1), we find that these results are *incorrect*. The correct shortest paths are provided in Table 2. Therefore, the Shoshan-Zwick algorithm is incorrect.

Table 1: Shortest paths for graph  $G'$  based on the Shoshan-Zwick algorithm.

$\Delta$	$a$	$b$	$c$
$a$	4	6	0
$b$	6	4	-2
$c$	0	-2	4

Table 2: Correct shortest paths for graph  $G'$ .

$\Delta$	$a$	$b$	$c$
$a$	0	2	4
$b$	2	0	6
$c$	4	6	0

## 4 The Errors in the Algorithm

In this section, we describe what causes the erroneous behavior of the Shoshan-Zwick algorithm. Recall that  $\Delta$  is the matrix that contains the costs of the shortest paths between all pairs of vertices after the algorithm terminates. Moreover, let  $\delta(i, j)$  denote the cost of the shortest path between nodes  $i$  and  $j$ . After the termination of the algorithm, we must have  $\Delta_{ij} = \delta(i, j)$  for any  $i, j \in \{1, \dots, n\}$ . However, as we showed in the counter-example in Section 3, it may be the case that  $\Delta_{ij} \neq \delta(i, j)$  for some  $i, j$  at termination. The exact errors of the algorithm are as follows:

1.  $\mathbf{R}$  is not computed correctly.
2.  $\mathbf{B}_0$  is not computed correctly.
3.  $\Delta$  is not computed correctly, since  $M \cdot \mathbf{B}_0 + \mathbf{R}$  is part of the sum producing it.

In the rest of this section, we illustrate what causes these errors.

It is clear from line 23 of Algorithm 2.1 that the matrices  $\mathbf{B}_k$  (for  $0 \leq k \leq l$ ) represent the  $\lceil \log_2 n \rceil$  most significant bits of each distance. That is,

$$(\mathbf{B}_k)_{ij} = \begin{cases} 1 & \text{if } 2^k \cdot M \text{ must be added to } \Delta_{ij} \text{ so that } \Delta_{ij} = \delta(i, j) \\ 0 & \text{otherwise} \end{cases},$$

while  $\mathbf{R}$  represents the remainder of each distance modulo  $M$ . This is also illustrated in [22, Lemma 3.6], where for every  $0 \leq k \leq l$ ,  $(\mathbf{B}_k)_{ij} = 1$  if and only if  $\delta(i, j) \bmod 2^{k+m+1} \geq 2^{k+m}$ , while  $\mathbf{R}_{ij} = \delta(i, j) \bmod M$ . Hence, for every  $i, j$ , we must have

$$(M \cdot \mathbf{B}_0 + \mathbf{R})_{ij} = \delta(i, j) \bmod 2^{m+1}. \quad (1)$$

The first error of the algorithm arises immediately from the key observation that  $\mathbf{P}_0$  can have entries with negative values. This means that line 22 (that sets  $\mathbf{R}_{ij} = (\mathbf{P}_0)_{ij} \bmod M$ ) is not correctly calculating  $\mathbf{R}_{ij} = \delta(i, j) \bmod M$  since  $\delta(i, j) \geq 0$  by definition, while  $(\mathbf{P}_0)_{ij}$  can be negative.

A closer examination of how  $\mathbf{P}_0$  obtains its negative values reveals another error of the algorithm. The following definitions are given in [22, Section 3]. Consider a set  $Y \subseteq [0, M \cdot n]$ . Note that  $[0, M \cdot n]$  includes any value that  $\delta(i, j)$  can take, since  $n$  is the number of nodes, and  $M$  is the maximum edge cost. Let  $Y = \cup_{r=1}^p [a_r, b_r]$ , where  $a_r \leq b_r$ , for  $1 \leq r \leq p$  and  $b_r < a_{r+1}$ , for  $1 \leq r < p$ . Let  $\Delta_Y$  be an  $n \times n$  matrix, whose elements are in the range  $\{-M, \dots, M\} \cup \{+\infty\}$ , such that for every  $1 \leq i, j \leq n$ , we have

$$(\Delta_Y)_{ij} = \begin{cases} -M & \text{if } a_r \leq \delta(i, j) \leq b_r - M \text{ for some } 1 \leq r \leq p, \\ \delta(i, j) - b_r & \text{if } b_r - M < \delta(i, j) \leq b_r + M \text{ for some } 1 \leq r \leq p, \\ +\infty & \text{otherwise.} \end{cases} \quad (2)$$

By [22, Lemma 3.5],  $\mathbf{P}_0 = \Delta_{Y_0}$ , where  $Y_0 = \{x | (x \bmod 2^{m+1}) = 0\}$ . Recall that  $2^m = M$ . Note that by definition of  $Y_0$ , when calculating  $\mathbf{P}_0 = \Delta_{Y_0}$ , it can only be the case that  $a_r = b_r$ . Moreover,  $b_r = 2^{m+1} \cdot (r-1)$  for  $1 \leq r \leq p$ , where  $p$  is such that  $2^{m+1} \cdot (p-1) \leq M \cdot n < 2^{m+1} \cdot p$ . But then:

$$(\cup_{r=1}^p [b_r - M, b_r + M]) \supset [0, M \cdot n]$$

That is,  $(\cup_{r=1}^p [b_r - M, b_r + M])$  covers all possible values that  $\delta(i, j)$  may take for any  $i, j$ . Hence,

$$(\mathbf{P}_0)_{ij} = \begin{cases} \delta(i, j) & \text{for } r = 1 \text{ (i.e., if } \delta(i, j) \leq 2^m), \\ \delta(i, j) - b_r & \text{for } 2 \leq r \leq p, \text{ such that } b_r - 2^m < \delta(i, j) \leq b_r + 2^m. \end{cases} \quad (3)$$

Let us examine the values that  $(\mathbf{P}_0)_{ij}$  takes by equation (3):

- For  $0 \leq \delta(i, j) \leq 2^m$ , we have  $(\mathbf{P}_0)_{ij} = \delta(i, j) \bmod 2^{m+1}$ .
- For  $2^m < \delta(i, j) < 2^m + 2^m$ , we have  $(\mathbf{P}_0)_{ij} = (\delta(i, j) \bmod 2^{m+1}) - 2^{m+1}$ .
- For  $2^{m+1} \leq \delta(i, j) \leq 2^{m+1} + 2^m$ , we have  $(\mathbf{P}_0)_{ij} = \delta(i, j) \bmod 2^{m+1}$ .
- For  $2^{m+1} + 2^m < \delta(i, j) < 2^{m+2} + 2^m$ , we have  $(\mathbf{P}_0)_{ij} = (\delta(i, j) \bmod 2^{m+1}) - 2^{m+1}$ .

◦ And so forth...

More formally, equation (3) can be rewritten as follows:

$$(\mathbf{P}_0)_{ij} = \begin{cases} \delta(i, j) \bmod 2^{m+1} & \text{if } \delta(i, j) \bmod 2^{m+1} \leq 2^m, \\ (\delta(i, j) \bmod 2^{m+1}) - 2^{m+1} & \text{if } \delta(i, j) \bmod 2^{m+1} > 2^m. \end{cases} \quad (4)$$

Moreover, equation (4) implies that

$$\text{for } i, j \text{ such that } \delta(i, j) \bmod 2^{m+1} \leq 2^m, \text{ we have } 0 \leq (\mathbf{P}_0)_{ij} \leq M, \quad (5)$$

while

$$\text{for } i, j \text{ such that } \delta(i, j) \bmod 2^{m+1} > 2^m, \text{ we have } -M < (\mathbf{P}_0)_{ij} < 0. \quad (6)$$

Recall now that we must have  $(\mathbf{B}_0)_{ij} = 1$  if and only if  $\delta(i, j) \bmod 2^{m+1} \geq 2^m$ . However, from equations (5) and (6), this does not hold (as claimed in the proof of [22, Lemma 3.6]) for  $\mathbf{B}_0 = (0 \leq \mathbf{P}_0 < M)$  (i.e., line 21 of Algorithm 2.1). Therefore, the algorithm does not compute  $\mathbf{B}_0$  correctly.

It is clear that in the presence of these two identified errors (in calculating  $\mathbf{R}$  and  $\mathbf{B}_0$ ), the algorithm is not computing  $\Delta$  correctly.

To accommodate understanding, let us consider the case of  $\mathbf{P}_0$  for graph  $G'$  (Figure 1) in Section 3. We have  $Y_0 \subseteq [0, M \cdot n]$ , i.e.,  $Y_0 \subseteq [0, 12]$ . Further,  $Y_0 = \{x \mid (x \bmod 2^{2+1}) = 0\}$ . This means that  $Y_0 = \{0, 8\}$ . Thus, with respect to the definition of  $(\Delta_Y)_{ij}$ , we have  $a_1 = b_1 = 0$  and  $a_2 = b_2 = 8$ . Therefore,

$$(\mathbf{P}_0)_{ij} = (\Delta_{Y_0})_{ij} = \begin{cases} \delta(i, j) & \text{if } -4 < \delta(i, j) \leq 4 \text{ (for } r = 1 \text{ and hence } b_r = 0), \\ \delta(i, j) - 8 & \text{if } 4 < \delta(i, j) \leq 12 \text{ (for } r = 2 \text{ and hence } b_r = 8). \end{cases}$$

Let us consider the shortest path cost of nodes  $a$  ( $i = 1$ ) and  $b$  ( $j = 2$ ). We have  $\delta(1, 2) = 2$ , and thus  $(\mathbf{P}_0)_{12} = 2$ . We now consider the shortest path cost of nodes  $b$  ( $i = 2$ ) and  $c$  ( $j = 3$ ). We then have  $\delta(2, 3) = 6$ . This, however, means that  $(\mathbf{P}_0)_{23} = -2$ . Although  $\delta(2, 3) \bmod 2^3 > 2^2$ , we have  $(\mathbf{B}_0)_{23} = 0$ . Moreover,  $\mathbf{R}_{23} = (\mathbf{P}_0)_{23} \bmod 8 = -2$  (instead of  $\mathbf{R}_{23} = \delta(2, 3) \bmod 8 = 6$ ). These two errors lead to  $\Delta_{23} = -2$  instead of 6 (see Table 2).

## 5 The Revised Algorithm

In this section, we present a new version of the Shoshan-Zwick algorithm that resolves the problems illustrated in Section 4. Lines 1 to 20 of Algorithm 2.1 remain unchanged. We make the following changes to lines 21 to 24:

1. We replace  $\mathbf{B}_0$  with  $\hat{\mathbf{B}}_0$  and set  $\hat{\mathbf{B}}_0$  to  $(-M < \mathbf{P}_0 < 0)$  in line 21.
2. We replace  $\mathbf{R}$  with  $\hat{\mathbf{R}}$  and set  $\hat{\mathbf{R}}$  to  $\mathbf{P}_0$  in line 22.

3. We set  $\Delta$  to  $M \cdot \sum_{k=1}^l 2^k \cdot \mathbf{B}_k + 2 \cdot M \cdot \hat{\mathbf{B}}_0 + \hat{\mathbf{R}}$  in line 23.

Note that we have replaced  $\mathbf{B}_0$  and  $\mathbf{R}$  with  $\hat{\mathbf{B}}_0$  and  $\hat{\mathbf{R}}$ , respectively. The purpose of the change in notation is to show that these matrices no longer represent the incorrect versions from the original (erroneous) algorithm. Lines 21 to 24 of the revised algorithm are illustrated in Algorithm 5.1.

21:  $\hat{\mathbf{B}}_0 = (-M < \mathbf{P}_0 < 0)$ .  
 22:  $\hat{\mathbf{R}} = \mathbf{P}_0$ .  
 23:  $\Delta = M \cdot \sum_{k=1}^l 2^k \cdot \mathbf{B}_k + 2 \cdot M \cdot \hat{\mathbf{B}}_0 + \hat{\mathbf{R}}$ .  
 24: **return**  $\Delta$ .

**Algorithm 5.1:** Corrected portion of lines 21 to 24.

We refer to our counter-example in Section 3. Since the algorithm is correct up to line 20, we will examine how the algorithm operates in lines 21 to 24. Hence, we set  $\hat{\mathbf{B}}_0 = (-4 < \mathbf{P}_0 < 0)$ ,  $\hat{\mathbf{R}} = \mathbf{P}_0$ , and  $\Delta = 4 \cdot \sum_{k=1}^2 2^k \cdot \mathbf{B}_k + 8 \cdot \hat{\mathbf{B}}_0 + \hat{\mathbf{R}}$ .

$$\hat{\mathbf{B}}_0 = (-4 < \mathbf{P}_0 < 0) = \left( -4 \leq \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & -2 \\ 4 & -2 & 0 \end{bmatrix} < 0 \right) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix},$$

$$\hat{\mathbf{R}} = \mathbf{P}_0 = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & -2 \\ 4 & -2 & 0 \end{bmatrix},$$

$$\begin{aligned} \Delta &= 4 \cdot \sum_{k=1}^2 2^k \cdot \mathbf{B}_k + 8 \cdot \hat{\mathbf{B}}_0 + \hat{\mathbf{R}} \\ &= 4 \cdot \left( 2^1 \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + 2^2 \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right) + 8 \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & -2 \\ 4 & -2 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} \end{aligned}$$

Line 24 returns  $\Delta$ . The elements of  $\Delta$  reflect the correct shortest path costs given in Table 2. Therefore, the revised algorithm works correctly for our counter-example. The corrections in the algorithm are not limited to the counter-example, as shown in the theorem below:

**Theorem 5.1** *The revised Shoshan-Zwick algorithm calculates all the shortest path costs in an undirected graph with integer edge costs in the range  $\{1, \dots, M\}$ .*

**Proof.** It suffices to show that  $2 \cdot M \cdot \hat{\mathbf{B}}_0 + \hat{\mathbf{R}}$  represents what the original algorithm intended to represent with  $M \cdot \mathbf{B}_0 + \mathbf{R}$ . That is, by equation (1), it suffices to show that  $(2 \cdot M \cdot \hat{\mathbf{B}}_0 + \hat{\mathbf{R}})_{ij} = \delta(i, j) \mod 2^{m+1}$ , for every  $1 \leq i, j \leq n$ .

First, we consider the case where  $\delta(i, j) \mod 2^{m+1} \leq 2^m$ . Equation (5) indicates that  $0 \leq (\mathbf{P}_0)_{ij} \leq M$ . Hence,  $(\hat{\mathbf{B}}_0)_{ij} = 0$  (by line 21 of the revised algorithm). Moreover, since  $\hat{\mathbf{R}}_{ij} = (\mathbf{P}_0)_{ij}$  (by line 22 of the



revised algorithm), we have that  $\hat{\mathbf{R}}_{ij} = \delta(i, j) \bmod 2^{m+1}$  by equation (4). Thus,  $(2 \cdot M \cdot \hat{\mathbf{B}}_0 + \hat{\mathbf{R}})_{ij} = \delta(i, j) \bmod 2^{m+1}$ .

We next consider the case where  $\delta(i, j) \bmod 2^{m+1} > 2^m$ . Equation (6) indicates that  $-M < (\mathbf{P}_0)_{ij} < 0$ . Hence,  $(\hat{\mathbf{B}}_0)_{ij} = 1$  (by line 21 of the revised algorithm). Further,  $\hat{\mathbf{R}}_{ij} = (\delta(i, j) \bmod 2^{m+1}) - 2^{m+1}$  by equation (4). Therefore,  $(2 \cdot M \cdot \hat{\mathbf{B}}_0 + \hat{\mathbf{R}})_{ij} = 2 \cdot 2^m \cdot 1 + (\delta(i, j) \bmod 2^{m+1}) - 2^{m+1} = \delta(i, j) \bmod 2^{m+1}$ , which completes the proof.  $\square$

## 6 Efficacy

In this section, we identify issues with implementing the Shoshan-Zwick algorithm. Recall that the algorithm runs in  $\tilde{O}(M \cdot n^\omega)$  time, where  $\omega$  is the exponent for the fastest known matrix multiplication algorithm. This means that the running time of the algorithm depends on which matrix multiplication algorithm is used. We will discuss two subcubic matrix multiplication algorithms and show why it is impractical to use them in the Shoshan-Zwick implementation.

The current fastest matrix multiplication algorithm is provided by [29], where  $\omega < 2.3727$ . This approach tightens the techniques used in [5], which gives a matrix multiplication algorithm where  $\omega < 2.376$ . Although both matrix multiplication algorithms are theoretically efficient, neither one is practical to implement. They both provide an advantage only for matrices that are too large for modern hardware to handle [20].

We next consider Strassen's matrix multiplication algorithm [23], which runs in  $O(n^{2.8074})$  time. Recall that the algorithm computes  $\mathbf{A} \cdot \mathbf{B} = \mathbf{C}$  by partitioning the matrices into equally sized block matrices

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

where

$$\begin{aligned} \mathbf{C}_{1,1} &= \mathbf{A}_{1,1} \cdot \mathbf{B}_{1,1} + \mathbf{A}_{1,2} \cdot \mathbf{B}_{2,1} \\ \mathbf{C}_{1,2} &= \mathbf{A}_{1,1} \cdot \mathbf{B}_{1,2} + \mathbf{A}_{1,2} \cdot \mathbf{B}_{2,2} \\ \mathbf{C}_{2,1} &= \mathbf{A}_{2,1} \cdot \mathbf{B}_{1,1} + \mathbf{A}_{2,2} \cdot \mathbf{B}_{2,1} \\ \mathbf{C}_{2,2} &= \mathbf{A}_{2,1} \cdot \mathbf{B}_{1,2} + \mathbf{A}_{2,2} \cdot \mathbf{B}_{2,2} \end{aligned}$$

To reduce the total number of multiplications, seven new matrices are defined as follows:

$$\begin{aligned} \mathbf{M}_1 &= (\mathbf{A}_{1,1} + \mathbf{A}_{2,2}) \cdot (\mathbf{B}_{1,1} + \mathbf{B}_{2,2}) \\ \mathbf{M}_2 &= (\mathbf{A}_{2,1} + \mathbf{A}_{2,2}) \cdot \mathbf{B}_{1,1} \\ \mathbf{M}_3 &= \mathbf{A}_{1,1} \cdot (\mathbf{B}_{1,2} - \mathbf{B}_{2,2}) \\ \mathbf{M}_4 &= \mathbf{A}_{2,2} \cdot (\mathbf{B}_{2,1} - \mathbf{B}_{1,1}) \\ \mathbf{M}_5 &= (\mathbf{A}_{1,1} + \mathbf{A}_{1,2}) \cdot \mathbf{B}_{2,2} \\ \mathbf{M}_6 &= (\mathbf{A}_{2,1} - \mathbf{A}_{1,1}) \cdot (\mathbf{B}_{1,1} + \mathbf{B}_{1,2}) \\ \mathbf{M}_7 &= (\mathbf{A}_{1,2} - \mathbf{A}_{2,2}) \cdot (\mathbf{B}_{2,1} + \mathbf{B}_{2,2}) \end{aligned}$$

With these new matrices, the block matrices of  $\mathbf{C}$  can be redefined as

$$\begin{aligned}\mathbf{C}_{1,1} &= \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7 \\ \mathbf{C}_{1,2} &= \mathbf{M}_3 + \mathbf{M}_5 \\ \mathbf{C}_{2,1} &= \mathbf{M}_2 + \mathbf{M}_4 \\ \mathbf{C}_{2,2} &= \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6\end{aligned}$$

The process of dividing  $\mathbf{C}$  repeats recursively  $n$  times until the submatrices degenerate into a single number.

Although Strassen's algorithm is faster than the naive  $O(n^3)$  matrix multiplication algorithm, we cannot directly use it in the Shoshan-Zwick algorithm. Recall that the naive approach uses matrix multiplication over the closed semi-ring  $\{+, \cdot\}$ . The Shoshan-Zwick algorithm, on the other hand, actually uses matrix multiplication over the closed semi-ring  $\{\min, +\}$ , which is known as “funny matrix multiplication” or the “distance product” in the literature. Note that the sum operation in the naive approach is equivalent to the min operation in “funny matrix multiplication”. However, Strassen's algorithm requires an additive inverse. This implies that an inverse for the min operation is needed in “funny matrix multiplication”. Such an inverse does not exist. In fact, we cannot multiply matrices with less than  $\Omega(n^3)$  operations when only the min and sum operations are allowed [1, 18]. Thus, Strassen's algorithm cannot directly be used for computing shortest paths.

One potential solution is to encode a matrix used for distance products such that regular matrix multiplication works. [1] provides an approach for this conversion as follows: Suppose we want to convert an  $n \times n$  matrix  $\mathbf{A}$  to  $\mathbf{A}'$ . We set

$$a'_{ij} = (n + 1)^{a_{ij} - x},$$

where  $x$  is the smallest value in  $\mathbf{A}$ . We perform the same conversion from  $\mathbf{B}$  to  $\mathbf{B}'$ . We then obtain  $\mathbf{C}' = \mathbf{A}' \cdot \mathbf{B}'$  by:

$$c'_{ij} = \sum_{k=1}^n (n + 1)^{a_{ik} + b_{kj} - 2 \cdot x}.$$

We then use binary search to find the largest  $s_{ij}$  such that  $s_{ij} \leq a'_{ik} + b'_{kj} - 2 \cdot x$ , and we set  $c_{ij} = s_{ij} + 2 \cdot x$ . This gives us  $\mathbf{C}$ , which is the distance product of matrices  $\mathbf{A}$  and  $\mathbf{B}$ .

[1] states that the algorithm performs  $O(n^\omega)$  operations on integers which are  $\leq n \cdot (n + 1)^{2 \cdot M}$ , where  $M$  is the magnitude of the largest number. For large numbers, we would need  $O(M \cdot \log M)$  operations on  $O(\log n)$ -bit numbers. As a result, the total time to compute the distance product is  $O(M \cdot n^\omega \cdot \log M)$ . If we apply this to the Shoshan-Zwick algorithm, the algorithm takes  $\tilde{O}(M^2 \cdot n^\omega \cdot \log M)$  time.

Although the above algorithm provides a subcubic approach for implementing the Shoshan-Zwick algorithm, it is not necessarily the most efficient implementation. This is because there exist other efficient APSP algorithms for integer edge costs. For instance, we can implement Goldberg's  $O(m + n \cdot \log N)$  time single source shortest path algorithm [14], where  $N$  is the largest edge cost, and run it  $n$  times; one for each vertex. Goldberg's implementation is one of the currently known fastest implementations available. The resulting implementation is substantially faster compared to the Shoshan-Zwick algorithm.

## 7 Conclusion

In this paper, we revised the Shoshan-Zwick algorithm to resolve issues related to its correctness. We first provided a counter-example which shows that the algorithm is incorrect. We then identified the exact cause of the problem and presented a modified algorithm that resolves the problem. We also explained the efficacy issues of the algorithm. An interesting study would be to implement the Shoshan-Zwick algorithm and profile it with efficient APSP algorithms for graphs with integer edge costs.

## References

- [1] Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *J. Comput. Syst. Sci.*, 54(2):255–262, 1997.
- [2] Noga Alon and Raphael Yuster. Fast algorithms for maximum subset matching and all-pairs shortest paths in graphs with a (not so) small vertex cover. In *ESA*, pages 175–186, 2007.
- [3] Timothy M. Chan. All-pairs shortest paths with real weights in  $O(n^3 \log n)$  time. *Algorithmica*, 50(2):236–243, 2008.
- [4] Timothy M. Chan. All-pairs shortest paths for unweighted undirected graphs in  $o(mn)$  time. *ACM Transactions on Algorithms*, 8(4):34, 2012.
- [5] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 1–6, New York, NY, USA, 1987. ACM.
- [6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.
- [7] Marek Cygan, Harold N. Gabow, and Piotr Sankowski. Algorithmic applications of Baur-Strassen’s theorem: shortest cycles, diameter and matchings. In *FOCS*, pages 531–540, 2012.
- [8] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
- [9] W. Dobosiewicz. A more efficient algorithm for the min-plus multiplication. *Int. J. Comput. Math.*, 32:251–280, 1990.
- [10] Robert W. Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.
- [11] M. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34:596–615, 1987.
- [12] Michael L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM J. Comput.*, 5(1):83–89, 1976.

- [13] Zvi Galil and Oded Margalit. All pairs shortest paths for graphs with small integer length edges. *Journal of Computer and System Sciences*, 54(2):243–254, 1997.
- [14] A.V. Goldberg. A simple shortest path algorithm with linear average time. In *9th Ann. European Symp. on Algorithms (ESA 2001)*, volume 2161 of *Lecture Notes in Comput. Sci.*, pages 230–241, Aachen, Germany, 2001. Springer.
- [15] Y. Han. Improved algorithm for all pairs shortest paths. *Inform. Process. Lett.*, 91(5):245–250, 2004.
- [16] Yijie Han and Tadao Takaoka. An  $O(n^3 \log \log n / \log^2 n)$  time algorithm for all pairs shortest paths. In *SWAT*, pages 131–141, 2012.
- [17] D. Johnson. Efficient algorithms for shortest paths in sparse graphs. *Journal of the ACM*, 24:1–13, 1977.
- [18] L.R. Kerr. The effect of algebraic structure on the computational complexity of matrix multiplication. Ph.D. Thesis, Cornell University, 1970.
- [19] W. Liu, D. Wang, H. Jiang, W. Liu, and Chonggang Wang. Approximate convex decomposition based localization in wireless sensor networks. In *Proc. of IEEE INFOCOM*, pages 1853–1861, 2012.
- [20] S. Robinson. Toward an optimal algorithm for matrix multiplication. *SIAM News*, 38(9), 2005.
- [21] Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51(3):400–403, December 1995.
- [22] A. Shoshan and U. Zwick. All pairs shortest paths in undirected graphs with integer weights. In *FOCS*, pages 605–614, 1999.
- [23] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 14(3):354–356, 1969.
- [24] Tadao Takaoka. A new upper bound on the complexity of the all pairs shortest path problem. *Inform. Process. Lett.*, 43(4):195–199, 1992.
- [25] Tadao Takaoka. A faster algorithm for the all-pairs shortest path problem and its application. In *COCOON*, pages 278–289, 2004.
- [26] Tadao Takaoka. An  $O(n^3 \log \log n / \log n)$  time algorithm for the all-pairs shortest path problem. *Inf. Process. Lett.*, 96(5):155–161, 2005.
- [27] Mikkel Thorup. Undirected single source shortest path in linear time. In *FOCS*, pages 12–21, 1997.
- [28] Mikkel Thorup. Floats, integers, and single source shortest paths. In *STACS*, pages 14–24, 1998.
- [29] Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *STOC*, pages 887–898, 2012.

- [30] Raphael Yuster. A shortest cycle for each vertex of a graph. *Inf. Process. Lett.*, 111(21-22):1057–1061, 2011.
- [31] Uri Zwick. A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths. *Algorithmica*, 46(2):181–192, 2006.

# 用 Shoshan-Zwick 算法求解所有点对点最短路径

Pavlos Eirinakis , Matthew Williamson , K. Subramani

翻译人 李绍晓 专业班级 电信 1302

**摘要:** Shoshan-Zwick 算法解决无向图中的所有点对点最短路径问题, 边长为整数, 范围为 $\{1, 2, \dots, M\}$ 。它运行在 $\tilde{O}(M \cdot n^\omega)$ 时间, 其中 $n$ 是顶点的数量,  $M$ 是最大整数边长, 并且 $\omega < 2.3727$ 是矩阵乘法的指数。这是所知道的算法中最快的。本文将指出 Shoshan-Zwick 算法中的错误之处, 并修改算法以解决此问题。此外, 通过使用当前存在的子立方矩阵乘法算法, 来讨论 Shoshan-Zwick 算法的各方面实施情况。

**关键词:** 所有点对点最短路径、Shoshan-Zwick

## 1. 引言

Shoshan-Zwick 算法<sup>[22]</sup>解决了无向图中的所有点对点最短路径(APSP)问题, 其中边长的范围为整数 $\{1, 2, \dots, M\}$ 。当 $n \times n$ 矩阵的边长范围为 $\{1, 2, \dots, M\}$ 时, 通过计算距离积得到结果为 $O(\log(M \cdot n))$ 。算法运行在 $\tilde{O}(M \cdot n^\omega)$ 的时间内, 其中 $\omega < 2.3727$ 是已知的最快矩阵乘法算法的指数<sup>[29]</sup>。

APSP 问题是图论算法中的一个基础问题。考虑具有 $n$ 个节点、 $m$ 条边的图。对于边权重为正的有向或者无向图情况, 我们可以使用已知的计算方法运行在 $O(m \cdot n + n^2 \cdot \log n)$ 时间<sup>[8,11,17]</sup>和 $O(n^3)$ 时间<sup>[10]</sup>。另一种 APSP 算法可运行在 $O\left(n^3 \cdot \left(\frac{\log \log n}{\log n}\right)^{\frac{1}{2}}\right)$ 时间<sup>[12]</sup>,  $O(n^3 \cdot \sqrt{\log \log n / \log n})$ 时间<sup>[24]</sup>,  $O(n^3 / \sqrt{\log n})$ 时间<sup>[9]</sup>,  $O(n^3 \cdot (\log \log n / \log n)^{5/7})$ 时间<sup>[15]</sup>,  $O(n^3 \cdot (\log \log n)^2 \log n / \log n)$ 时间<sup>[26]</sup>,  $O(n^3 \cdot \log \log n / \log n)$ 时间<sup>[31]</sup>,  $O(n^3 / \log n)$ 时间<sup>[3]</sup>和 $O(n^3 \cdot \log \log n / \log^2 n)$ <sup>[16]</sup>时间内。对于具有整数边权重的无向图, 该问题可以在 $O(m \cdot n)$ 时间

[27,28]中求解。快矩阵乘法算法也可以用于解决具有小整数边权重的密集图 APSP 问题。[13,21]提供一个算法，可在未加权的无向图中以  $\tilde{O}(n^\omega)$  时间运行，其中  $\omega < 2.3727$  是矩阵乘法的指数<sup>[29]</sup>。[4]改进了这个算法，能够在  $O(m \cdot n)$  时间内运行。

在本文中，我们修改 Shoshan-Zwick 算法，来解决算法中的错误之处。在算法程序的实例实施中发生错误情况时(例如识别无向图中的负权重)，我们发现算法没有产生正确的结果。由于 Shoshan-Zwick 算法在当前版本中不正确，任何使用此算法的程序也不正确。例如，由 Alon 和 Yuster<sup>[2]</sup>，Cygan et. al<sup>[7]</sup>，W. Liu et al<sup>[19]</sup>和 Yuster<sup>[30]</sup>提供的结果都取决于 Shoshan-Zwick 算法。因此，他们结果当前都是不正确的。通过应用我们的修订版本的算法，上述结果的问题将被解决。我们还讨论关于使用已知 Shoshan-Zwick 算法解决的子例程矩阵乘法问题。

本文的主要贡献如下：

- (i) 展现当前 Shoshan-Zwick 算法中引发错误情况的反例。
- (ii) 给出算法在何处出错，为何出错的详细解释。
- (iii) Shoshan-Zwick 算法的修改版本，修正先前版本的问题。校正不影响算法的时间复杂度。
- (iv) 关于算法中使用的矩阵乘法子例程的讨论。

本文的其余部分组织如下。我们在第 2 节中描述 Shoshan-Zwick 算法。第 3 节中通过提供简单的反例，证明当前版本算法的不正确。第 4 节中标识出错之处和原因。在第 5 节中，我们给出一个修订版本算法，并提供正确性的证明。在第 6 节，我们详细讨论这个修正版 Shoshan-Zwick 算法的效率。第 7 节总结了本文，总结了我们的贡献，讨论未来的研究方向。

## 2. Shoshan-Zwick 算法

在本节中，我们将回顾使用 Shoshan-Zwick 算法解决具有整数边权重的无向图 APSP 问题。考虑图  $G = (V, E)$ ，其中  $V = \{0, 1, 2, \dots, n\}$  是顶点集合， $E$  是边集合。该图表示为  $n \times n$  的矩阵  $D$ ，其中  $d_{ij}$  是  $edge(i, j)$  的权重。如果  $(i, j) \in E$ ，则有  $d_{ii} = 0$ 。否则  $d_{ij} = +\infty$ 。注意，在不失一般性的情况下，边权重取自范围

$\{1, 2, \dots, M\}$ , 其中当  $m \geq 1$  时,  $M = 2^m$ 。

该算法计算距离积的对数以确定最短路径. 令  $\mathbf{A}$  和  $\mathbf{B}$  为两个  $n \times n$  矩阵。它们的距离积  $\mathbf{A} \star \mathbf{B}$  是一个  $n \times n$  矩阵, 使得:

$$(\mathbf{A} \star \mathbf{B})_{ij} = \min_{k=1}^n \{a_{ik} + b_{kj}\}, 1 \leq i, j \leq n.$$

两个  $n \times n$  矩阵的元素范围为  $\{1, 2, \dots, N\}$ , 其距离乘积  $\{1, 2, \dots, M, +\infty\}$  的计算在  $\tilde{O}(Mn^w)$  时间内<sup>[1]</sup>。范围为  $\{1, 2, \dots, M\}$  的两个矩阵的距离积也为一个矩阵, 其范围为  $\{1, 2, \dots, 2 \cdot M\}$ 。然而, Shoshan-Zwick 算法不允许其矩阵中的元素进行范围增加。因此, 如果  $A$  是  $n \times n$  矩阵,  $a, b$  是两个数, 且  $a \leq b$ , 则该算法利用两个操作, 即  $clip(A, a, b)$  和  $chop(A, a, b)$ , 来产生如下对应的  $n \times n$  矩阵:

$$(clip(\mathbf{A}, a, b))_{ij} = \begin{cases} a & \text{if } a_{ij} < a \\ a_{ij} & \text{if } a \leq a_{ij} \leq b \\ +\infty & \text{if } a_{ij} > b \end{cases}$$

$$(chop(\mathbf{A}, a, b))_{ij} = \begin{cases} a_{ij} & \text{if } a \leq a_{ij} \leq b \\ +\infty & \text{otherwise} \end{cases}$$

该算法也定义了  $n \times n$  矩阵的  $A \wedge B, A \wedge^- B, A \vee B$  的计算, 如下:

$$(\mathbf{A} \wedge \mathbf{B})_{ij} = \begin{cases} a_{ij} & \text{if } b_{ij} < 0 \\ +\infty & \text{otherwise} \end{cases}$$

$$(\mathbf{A} \wedge^- \mathbf{B})_{ij} = \begin{cases} a_{ij} & \text{if } b_{ij} \geq 0 \\ +\infty & \text{otherwise} \end{cases}$$

$$(\mathbf{A} \vee \mathbf{B})_{ij} = \begin{cases} a_{ij} & \text{if } a_{ij} \neq +\infty \\ b_{ij} & \text{if } a_{ij} = +\infty, b_{ij} \neq +\infty \\ +\infty & \text{if } a_{ij} = b_{ij} = +\infty \end{cases}$$

最后, 如果  $\mathbf{C} = (c_{ij})$  和  $\mathbf{P} = (p_{ij})$  是矩阵的话, 这个算法将会生成一个如下的布尔矩阵  $(\mathbf{C} \geq 0)$  和  $(0 \leq \mathbf{P} \leq M)$ 。

$$(\mathbf{C} \geq 0)_{ij} = \begin{cases} 1 & \text{if } c_{ij} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$(0 \leq \mathbf{P} \leq M)_{ij} = \begin{cases} 1 & \text{if } 0 \leq p_{ij} \leq M \\ 0 & \text{otherwise} \end{cases}$$

使用上面的定义, Shoshan-Zwick 算法通过在  $\lceil \log_2 n \rceil$  的时间内找到每个距离的最短路径和每个距离对  $M$  取余, 来得到足够的信息来计算并重建距离。程序展示在算法 2.1 中, 证明正确性在[22, 引理 3.6]中给出。



该算法在  $O(\log n + \log M) = O(\log M \cdot n)$  时间内计算产生的距离矩阵，其范围在  $\{0, 1, 2, \dots, 2 \cdot M\}$  或者  $\{-M, \dots, M\}$ 。注意每个距离矩阵的乘积可以减少到一个恒定的数量，其矩阵的范围为  $\{1, 2, \dots, M\}$ 。算法中的所有其他操作都需要  $O(n^2 \cdot (\log n + \log M))$  时间。因此，算法的总运行时间为  $\tilde{O}(Mn^w)$  [22, 定理 3.7]。

Function SHOSHAN-ZWICK-APSP(**D**)

```

1:  $l = \lceil \log_2 n \rceil$ 
2:  $m = \log_2 M$ 
3: for ( $k = 1$  to  $m + 1$ ) do
4:    $\mathbf{D} = \text{clip}(\mathbf{D} * \mathbf{D}, 0, 2 \cdot M)$ 
5: end for
6:  $\mathbf{A}_0 = \mathbf{D} - M$ 
7: for ( $k = 1$  to  $l$ ) do
8:    $\mathbf{A}_k = \text{clip}(\mathbf{A}_{k-1} * \mathbf{A}_{k-1}, -M, M)$ 
9: end for
10:  $\mathbf{C}_l = -M$ 
11:  $\mathbf{P}_l = \text{clip}(D, 0, M)$ 
12:  $\mathbf{Q}_l = +\infty$ 
13: for ( $k = l - 1$  down to  $0$ ) do
14:    $\mathbf{C}_k = [\text{clip}(\mathbf{P}_{k+1} * \mathbf{A}_k, -M, M) \wedge \mathbf{C}_{k+1}] \vee [\text{clip}(\mathbf{Q}_{k+1} * \mathbf{A}_k, -M, M) \wedge \mathbf{C}_{k+1}]$ 
15:    $\mathbf{P}_k = \mathbf{P}_{k+1} \vee \mathbf{Q}_{k+1}$ 
16:    $\mathbf{Q}_k = \text{chop}(\mathbf{C}_k, 1 - M, M)$ 
17: end for
18: for ( $k = 1$  to  $l$ ) do
19:    $\mathbf{B}_k = (\mathbf{C}_k \geq 0)$ 
20: end for
21:  $\mathbf{B}_0 = (0 \leq \mathbf{P}_0 \leq M)$ 
22:  $\mathbf{R} = \mathbf{P}_0 \bmod M$ 
23:  $\Delta = M \cdot \sum_{k=0}^f 2^k \cdot \mathbf{B}_k + \mathbf{R}$ 
24: return  $\Delta$ 

```

算法 2.1: Shoshan-Zwick APSP 的算法实现

### 3. 一个反例

在本节中，我们提供了一个反例的详细介绍，来展示 Shoshan-Zwick 算法的错误之处，如[22]所示。考虑图  $G'$  的 APSP 问题，如图 1。

图  $G'$  可以表示为  $3 \times 3$  的矩阵：

$$\mathbf{D} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix}$$

我们现在开始运行我们的算法。首先,  $l = \lceil \log_2 3 \rceil = 2$ ,  $m = \lceil \log_2 4 \rceil = 2$ 。下面将运行程序的第 3 行到 5 行, 我们将会令  $\mathbf{D} = \text{clip}(\mathbf{D} * \mathbf{D}, 0, 8)$  这个步骤执行三次。依此我们会得到:

$$\begin{aligned}\mathbf{D} * \mathbf{D} &= \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix} * \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix}, \quad \mathbf{D} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} \text{ (for } k = 1) \\ \mathbf{D} * \mathbf{D} &= \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix}, \quad \mathbf{D} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} \text{ (for } k = 2) \\ \mathbf{D} * \mathbf{D} &= \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix}, \quad \mathbf{D} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} \text{ (for } k = 3)\end{aligned}$$

下一步令  $\mathbf{A}_0 = \mathbf{D} - \mathbf{M}$ , 我们将会得到

$$\mathbf{A}_0 = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} - 4 = \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix}$$

在第 7 行到第 9 行的循环中, 我们计算  $\mathbf{A}_k = \text{clip}(\mathbf{A}_{k-1} * \mathbf{A}_{k-1}, -4, 4)$ , 在  $k$  等于 1 或 2 的时候。我们将会得到

$$\begin{aligned}\mathbf{A}_0 * \mathbf{A}_0 &= \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix} * \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix} = \begin{bmatrix} -8 & -6 & -4 \\ -6 & -8 & -2 \\ -4 & -2 & -8 \end{bmatrix} \\ \mathbf{A}_1 &= \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} \\ \mathbf{A}_1 * \mathbf{A}_1 &= \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} * \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} = \begin{bmatrix} -8 & -8 & -8 \\ -8 & -8 & -8 \\ -8 & -8 & -8 \end{bmatrix} \\ \mathbf{A}_2 &= \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -4 \\ -4 & -4 & -4 \end{bmatrix}\end{aligned}$$

在第 10 行到第 12 行, 我们令  $\mathbf{C}_2 = -4$ ,  $\mathbf{P}_2 = \text{clip}(\mathbf{D}, 0, 4)$ ,  $\mathbf{Q}_2 = +\infty$ 。我们将会得到

$$\mathbf{C}_2 = \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -4 \\ -4 & -4 & -4 \end{bmatrix}, \quad \mathbf{P}_2 = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix}, \quad \mathbf{Q}_2 = \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix}$$

我们现在计算第 13 行到第 17 行的循环。我们运行的操作为

$$\mathbf{C}_k = [\text{clip}(\mathbf{P}_{k+1} * \mathbf{A}_k, -4, 4) \wedge \mathbf{C}_{k+1}] \vee [\text{clip}(\mathbf{Q}_{k+1} * \mathbf{A}_k, -4, 4) \wedge \mathbf{C}_{k+1}]$$

$$\mathbf{P}_k = \mathbf{P}_{k+1} \bigvee \mathbf{Q}_{k+1}$$

$$\mathbf{Q}_k = \text{chop}(\mathbf{C}_k, -3, 4)$$

在循环到 $k = 1$ 和 $k = 0$ 时，我们将会得到

$$(\text{for } k = 1) \mathbf{P}_2 * \mathbf{A}_1 = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix} * \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} = \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix},$$

$$\text{clip}(\mathbf{P}_2 * \mathbf{A}_1, -4, 4) = \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix},$$

$$\begin{aligned} \text{clip}(\mathbf{P}_2 * \mathbf{A}_1, -4, 4) \bigwedge \mathbf{C}_2 &= \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} \bigwedge \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -4 \\ -4 & -4 & -4 \end{bmatrix} \\ &= \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix}, \end{aligned}$$

$$\mathbf{Q}_2 * \mathbf{A}_1 = \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix} * \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} = \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix},$$

$$\begin{aligned} \text{clip}(\mathbf{Q}_2 * \mathbf{A}_1, -4, 4) \bigwedge \mathbf{C}_2 &= \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix} \bigwedge \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -4 \\ -4 & -4 & -4 \end{bmatrix} \\ &= \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix}, \end{aligned}$$

$$\mathbf{C}_1 = \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} \vee \mathbf{Q}_2 = \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix},$$

$$\mathbf{P}_1 = \mathbf{P}_2 * \mathbf{Q}_2 = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix} * \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix},$$

$$\mathbf{Q}_1 = \text{chop}(\mathbf{C}_1, -3, 4) = \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & -2 \\ \infty & -2 & \infty \end{bmatrix},$$

$$(\text{for } k = 0) \mathbf{P}_1 * \mathbf{A}_0 = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix} * \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix} = \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & -2 \\ 0 & 2 & -4 \end{bmatrix},$$

$$\text{clip}(\mathbf{P}_1 * \mathbf{A}_0, -4, 4) = \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix},$$

$$\text{clip}(\mathbf{P}_1 * \mathbf{A}_0, -4, 4) \bigwedge \mathbf{C}_1 = \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix} \bigwedge \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix}$$

$$= \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix},$$

$$\mathbf{Q}_1 * \mathbf{A}_0 = \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & -2 \\ \infty & -2 & \infty \end{bmatrix} * \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix} = \begin{bmatrix} \infty & \infty & \infty \\ -2 & 0 & -6 \\ -4 & -6 & -4 \end{bmatrix},$$

$$\begin{aligned}
clip(\mathbf{Q}_1 * \mathbf{A}_0, -4, 4) \bigwedge \mathbf{C}_1 &= \begin{bmatrix} \infty & \infty & \infty \\ -2 & 0 & -4 \\ -4 & -4 & -4 \end{bmatrix} \bigwedge \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} \\
&= \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix}, \\
\mathbf{C}_0 &= \begin{bmatrix} -4 & -2 & -4 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix} \vee \mathbf{Q}_2 = \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix}, \\
\mathbf{P}_0 = \mathbf{P}_1 * \mathbf{Q}_1 &= \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix} * \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & -2 \\ \infty & -2 & \infty \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & -2 \\ 4 & -2 & 0 \end{bmatrix}, \\
\mathbf{Q}_0 = chop(\mathbf{C}_0, -3, 4) &= \begin{bmatrix} \infty & -2 & \infty \\ -2 & \infty & 2 \\ \infty & 2 & \infty \end{bmatrix}
\end{aligned}$$

在第 18 到 20 行的循环中，我们让  $\mathbf{B}_k = (\mathbf{C}_k \geq 0)$  当  $k = 1$  和  $k = 2$  时。这意味着

$$\begin{aligned}
\mathbf{B}_1 = (\mathbf{C}_1 \geq 0) &= \left( \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} \geq 0 \right) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \\
\mathbf{B}_2 = (\mathbf{C}_2 \geq 0) &= \left( \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -4 \\ -4 & -4 & -4 \end{bmatrix} \geq 0 \right) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},
\end{aligned}$$

我们接着通过 22 到 23 行的操作，令  $\mathbf{B}_0 = (\mathbf{0} \leq \mathbf{P}_0 \leq 4)$ ,  $\mathbf{R} = \mathbf{P}_0 \bmod 4$ ,  $\Delta = 4 \cdot \sum_{k=0}^2 2^k \cdot \mathbf{B}_k + \mathbf{R}$ 。

$$\begin{aligned}
\mathbf{B}_0 = (\mathbf{0} \leq \mathbf{P}_0 \leq 4) &= \left( \mathbf{0} \leq \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & -2 \\ 4 & -2 & 0 \end{bmatrix} \leq 4 \right) = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \\
\mathbf{R} = \mathbf{P}_0 \bmod 4 &= \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & -2 \\ 4 & -2 & 0 \end{bmatrix} \bmod 4 = \begin{bmatrix} 0 & 2 & 0 \\ 2 & 0 & -2 \\ 0 & -2 & 0 \end{bmatrix}
\end{aligned}$$

$$\begin{aligned}
\Delta &= 4 \cdot \sum_{k=0}^2 2^k \cdot \mathbf{B}_k + \mathbf{R} \\
&= 4 \cdot \left( 2^0 \cdot \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + 2^1 \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + 2^2 \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right) + \begin{bmatrix} 0 & 2 & 0 \\ 2 & 0 & -2 \\ 0 & -2 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 4 & 4 & 0 \\ 4 & 4 & 0 \\ 0 & 0 & 4 \end{bmatrix} + \begin{bmatrix} 0 & 2 & 0 \\ 2 & 0 & -2 \\ 0 & -2 & 0 \end{bmatrix} = \begin{bmatrix} 4 & 6 & 0 \\ 6 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix}
\end{aligned}$$

算法结束，所得的最短路径结果在表 1 中给出。然而，如果我们检查图  $G'$  (图 1)，我们发现这些结果是不正确的。正确的最短路径在表 2 中给出。因此，当

前的 Shoshan-Zwick 算法是不正确的。

表 1 基于 Shoshan-Zwick 算法得出的  $G'$  图最短路径

$\triangle$	$a$	$b$	$c$
$a$	4	6	0
$b$	6	4	-2
$c$	0	2	-4

表 2  $G'$  图最短路径的正确结果

$\triangle$	$a$	$b$	$c$
$a$	4	6	0
$b$	6	4	-2
$c$	0	2	-4

## 5. 改进算法

在本节中，我们提出了一个新版本的 Shoshan-Zwick 算法来解决在第 4 节中发现的问题。算法 2.1 的第 1 到 20 行保持不变。我们对第 21 行至第 24 行进行以下更改。

1. 我们在第 21 行中将用  $\hat{\mathbf{B}}_0$  替换  $\mathbf{B}_0$ ，设置  $\hat{\mathbf{B}}_0$  为  $(-M < \mathbf{P}_0 < 0)$ 。
2. 我们在第 22 行中用  $\hat{\mathbf{R}}$  代替  $\mathbf{R}$ ，在第 22 行中将  $\mathbf{R}$  设置为  $\mathbf{P}_0$ 。
3. 我们在第 23 行中将  $\Delta$  设置为  $M \cdot \sum_{k=1}^2 2^k \cdot \mathbf{B}_k + 2 \cdot M \cdot \hat{\mathbf{B}}_0 + \hat{\mathbf{R}}$

注意，我们分别用  $\hat{\mathbf{B}}_0$  和  $\hat{\mathbf{R}}$  替换  $\mathbf{B}_0$  和  $\mathbf{R}$ 。改变符号的目的，是表明这些矩阵不再表示来自原始（错误）算法的不正确版本。在算法 5.1 中示出了改进算法的第 21 行到 24 行。

```

21:  $\hat{\mathbf{B}}_0 = (-M < \mathbf{P}_0 < 0)$ 
22:  $\hat{\mathbf{R}} = \mathbf{P}_0$ 
23:  $\Delta = M \cdot \sum_{k=1}^2 2^k \cdot \mathbf{B}_k + 2 \cdot M \cdot \hat{\mathbf{B}}_0 + \hat{\mathbf{R}}$ 
24: return  $\Delta$ 

```

算法 5.1 改进算法的第 21 行到 24 行

我们参考第 3 节中的反例。由于算法在第 20 行之前是正确的，故我们将检查算法在第 21 行至 24 行中的运行情况。我们令  $\hat{\mathbf{B}}_0 = (-4 < \mathbf{P}_0 < 0)$ ， $\hat{\mathbf{R}} = \mathbf{P}_0$ ，

以及

$$\Delta = 4 \cdot \sum_{k=1}^2 2^k \cdot \mathbf{B}_k + 8 \hat{\mathbf{B}}_0 + \hat{\mathbf{R}}$$

$$\mathbf{B}_0 = (-4 \leq \mathbf{P}_0 \leq 0) = \left( -4 \leq \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & -2 \\ 4 & -2 & 0 \end{bmatrix} < 0 \right) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix},$$

$$\mathbf{R} = \mathbf{P}_0 = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & -2 \\ 4 & -2 & 0 \end{bmatrix}$$

$$\Delta = 4 \cdot \sum_{k=0}^2 2^k \cdot \mathbf{B}_k + \mathbf{R}$$

$$= 4 \cdot \left( 2^1 \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + 2^2 \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right) + 8 \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix}$$

第 24 行返回了  $\Delta$ 。元素  $\Delta$  反映了表 2 中给出的正确最短路径。故修正算法正确地解出了我们的反例。算法中的校正不限于此反例，如下面的定理所示：

**定理 5.1** 修正的 Shoshan-Zwick 算法可计算无向图中的所有最短路径。其中整数边权重范围为  $\{1, \dots, M\}$ 。

**证明** 显而易见，我们用  $2 \cdot M \cdot \hat{\mathbf{B}}_0 + \hat{\mathbf{R}}$  替代了  $M \cdot \mathbf{B}_0 + \mathbf{R}$ 。也就是说，通过式 (1)，对于每个  $1 \leq i, j \leq n$ ，有  $(2 \cdot M \cdot \hat{\mathbf{B}}_0 + \hat{\mathbf{R}})_{ij} = \delta(i, j) \bmod 2^{m+1}$ 。

首先，我们考虑  $\delta(i, j) \bmod 2^{m+1} \leq 2^m$  的情况。等式 (5) 意味着  $(0 \leq (\mathbf{P}_0)_{ij} \leq M)$ 。因此， $(\hat{\mathbf{B}}_0)_{ij} = 0$ （在算法第 21 行的修正）。因此，当  $(\hat{\mathbf{R}})_{ij} = \mathbf{P}_{0ij}$  时，我们通过等式 (4) 可知  $(\hat{\mathbf{R}})_{ij} = \delta(i, j) \bmod 2^{m+1}$ 。因此， $(2 \cdot M \cdot \hat{\mathbf{B}}_0 + \hat{\mathbf{R}})_{ij} = \delta(i, j) \bmod 2^{m+1}$ 。

我们接下来考虑  $\delta(i, j) \bmod 2^{m+1} > 2^m$  的情况。等式 (6) 意味着  $-M < (\mathbf{P}_0)_{ij} < 0$ 。因此， $(\hat{\mathbf{B}}_0)_{ij} = 1$ （算法第 21 行的修正）。此外，我们通过等式 (4) 可知  $(\hat{\mathbf{R}})_{ij} = \delta(i, j) \bmod 2^{m+1} - 2^{m+1}$ 。因此， $(2 \cdot M \cdot \hat{\mathbf{B}}_0 + \hat{\mathbf{R}})_{ij} = 2 \cdot 2^m \delta(i, j) \bmod 2^{m+1}$ 。自此我们结束了证明。

## 6. 功效讨论

在本节中，我们确定 Shoshan-Zwick 算法的实现效率。回忆一下算法运行在  $\tilde{O}(M \cdot n^\omega)$  时间，其中  $\omega$  是最快已知的矩阵乘法指数。这个意味着算法的运行

时间取决于使用哪种矩阵乘法算法。我们将讨论两个子矩阵乘法算法，并说明为什么在 Shoshan-Zwick 中使用它们是不切实际的。

当前最快的矩阵乘法算法由[29]提供，其中 $\omega < 2.3727$ 。这种方法用的是在[5]中出现的技术，其给出了矩阵乘法算法，其中 $\omega < 2.376$ 。虽然两个矩阵乘法算法在理论上是高效的，但是没有一个实现的。他们两个只为那些使用现代硬件计算设备，进行大规模矩阵运算提供了优势。

我们接下来考虑 Strassen 的矩阵乘法算法<sup>[23]</sup>，它在 $O(n^{2.8074})$ 时间运行。回顾那个算法，是通过将矩阵分割成大小相等的块矩阵，来计算 $\mathbf{A} \cdot \mathbf{B} = \mathbf{C}$ 的。

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

其中

$$\mathbf{C}_{1,1} = \mathbf{A}_{1,1} \cdot \mathbf{B}_{1,1} + \mathbf{A}_{1,2} \cdot \mathbf{B}_{2,1}$$

$$\mathbf{C}_{1,2} = \mathbf{A}_{1,1} \cdot \mathbf{B}_{1,2} + \mathbf{A}_{1,2} \cdot \mathbf{B}_{2,2}$$

$$\mathbf{C}_{2,1} = \mathbf{A}_{2,1} \cdot \mathbf{B}_{1,1} + \mathbf{A}_{2,2} \cdot \mathbf{B}_{2,1}$$

$$\mathbf{C}_{2,2} = \mathbf{A}_{2,1} \cdot \mathbf{B}_{1,2} + \mathbf{A}_{2,2} \cdot \mathbf{B}_{2,2}$$

为了减少乘法的总数，定义如下的七个新矩阵：

$$\mathbf{M}_1 = (\mathbf{A}_{1,1} + \mathbf{A}_{2,2}) \cdot (\mathbf{B}_{1,1} + \mathbf{B}_{2,2})$$

$$\mathbf{M}_2 = (\mathbf{A}_{2,1} + \mathbf{A}_{2,2}) \cdot \mathbf{B}_{1,1}$$

$$\mathbf{M}_3 = \mathbf{A}_{1,1} \cdot (\mathbf{B}_{1,2} - \mathbf{B}_{2,2})$$

$$\mathbf{M}_4 = \mathbf{A}_{2,2} \cdot (\mathbf{B}_{2,1} - \mathbf{B}_{1,1})$$

$$\mathbf{M}_5 = (\mathbf{A}_{1,1} + \mathbf{A}_{1,2}) \cdot \mathbf{B}_{2,2}$$

$$\mathbf{M}_6 = (\mathbf{A}_{2,1} - \mathbf{A}_{1,1}) \cdot (\mathbf{B}_{1,1} + \mathbf{B}_{1,2})$$

$$\mathbf{M}_7 = (\mathbf{A}_{1,2} - \mathbf{A}_{2,2}) \cdot (\mathbf{B}_{2,1} + \mathbf{B}_{2,2})$$

使用以上这些新矩阵，我们可以重新定义 C 矩阵为：

$$\mathbf{C}_{1,1} = \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7$$

$$\mathbf{C}_{1,2} = \mathbf{M}_3 + \mathbf{M}_5$$

$$\mathbf{C}_{2,1} = \mathbf{M}_2 + \mathbf{M}_4$$

$$\mathbf{C}_{2,2} = \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6$$

将 C 矩阵进行重复递归地分割 $n$ 次，直到子矩阵退化为单个数字。

虽然 Strassen 的算法比常见的 $O(n^3)$ 矩阵乘法算法更快，但我们不能直接在 Shoshan-Zwick 算法中使用它。回想一下，常见方法使用矩阵乘法闭合半环

$\{+, \cdot\}$ 。另一方面，Shoshan-Zwick 算法实际上使用矩阵乘法在闭合半环  $\{\min, +\}$  上，这被称为“有趣矩阵乘法”或“距离乘积”。在文献中。注意，在常见方法中的求和运算相当于“有趣”中的最小运算矩阵乘法。然而，Strassen 的算法需要一个加法逆运算。这意味着在“有趣矩阵乘法”中需要最小操作。这样的逆不存在。事实上，我们不能乘以小于的矩阵  $O(n^3)$  矩操作，只允许  $\min$  和  $\text{sum}$  操作<sup>[1,18]</sup>。因此，Strassen 的算法不能直接用于计算最短路径。

一个可能的解决方案是对用于距离乘积的矩阵进行编码，获得规则矩阵乘法。[1]提供了这种转换的方法如下：假设我们想转换一个  $n \times n$  矩阵  $\mathbf{A}$  到  $\mathbf{A}'$ 。我们令

$$a'_{ij} = (n + 1)^{a_{ij} - x}$$

$x$  是在  $\mathbf{A}$  中最小的值。我们执行从  $\mathbf{B}$  到  $\mathbf{B}'$  的相同转换。然后通过下面的式子得到

$$c'_{ij} = \sum_{k=1}^n (n + 1)^{a_{ik} + b_{kj} - 2 \cdot x}$$

然后我们使用二分搜索找到最大的  $s_{ij}$ ，使  $s_{ij} \leq a'_{ik} + b'_{kj} - 2 \cdot x$ ，我们设  $c_{ij} = s_{ij} + 2 \cdot x$ 。矩阵  $\mathbf{C}$  是矩阵  $\mathbf{A}$  和  $\mathbf{B}$  的距离乘积。

[1]指出算法对的整数执行  $O(n^w)$  时间的运算，其值  $\leq n \cdot (n + 1)^{2 \cdot M}$ ，其中  $M$  是边长最大的数字。对于大数，我们需要在  $O(\log n)$  位上进行  $O(M \cdot \log M)$  运算数字。结果，计算距离积的总时间为  $O(M \cdot n^w \cdot \log M)$ 。如果我们应用这个 Shoshan-Zwick 算法，该算法需要  $\tilde{O}(M^2 \cdot n^w \cdot \log M)$  时间。

虽然上述算法提供了用于实现 Shoshan-Zwick 算法的方法，但它不一定是最有效的实现。这是因为存在其他有效的 APSP 算法用于整数边权重。例如，我们可以实现 Goldberg 的  $O(m + n \cdot \log N)$  时间单源最短路径算法<sup>[14]</sup>，其中  $N$  是最大边权重，对于每对定点运行  $n$  次。戈德堡的算法实现是当前已知最快的可用实现之一。结果基本上比 Shoshan-Zwick 算法更快。

## 7. 结论

在本文中，我们修改 Shoshan-Zwick 算法来解决与其正确性相关的问题。我们在第一节提供了算法不正确的反例。然后我们确定了确切的问题原因，并



提出了一个解决问题的修改算法。我们还解释了算法的效率问题。一个有趣的研究将是用 Shoshan-Zwick 算法和高效的配置，实现拥有整数边权重的 APSP 算法。

## 参考文献

- [1] Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *J. Comput. Syst. Sci.*, 54(2):255–262, 1997.
- [2] Noga Alon and Raphael Yuster. Fast algorithms for maximum subset matching and all-pairs shortest paths in graphs with a (not so) small vertex cover. In *ESA*, pages 175–186, 2007.
- [3] Timothy M. Chan. All-pairs shortest paths with real weights in  $O(n^3 \log n)$  time. *Algorithmica*, 50(2):236–243, 2008.
- [4] Timothy M. Chan. All-pairs shortest paths for unweighted undirected graphs in  $O(mn)$  time. *ACM Transactions on Algorithms*, 8(4):34, 2012.
- [5] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 1–6, New York, NY, USA, 1987. ACM.
- [6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3<sup>rd</sup> edition, 2009.
- [7] Marek Cygan, Harold N. Gabow, and Piotr Sankowski. Algorithmic applications of Baur-Strassen's theorem: shortest cycles, diameter and matchings. In *FOCS*, pages 531–540, 2012.
- [8] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
- [9] W. Dobosiewicz. A more efficient algorithm for the min-plus multiplication. *Int. J. Comput. Math.*, 32:251–280, 1990.
- [10] Robert W. Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.
- [11] M. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34:596–615, 1987.
- [12] Michael L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM J. Comput.*, 5(1):83–89, 1976. 14

- [13]Zvi Galil and Oded Margalit. All pairs shortest paths for graphs with small integer length edges. *Journal of Computer and System Sciences*, 54(2):243–254, 1997.
- [14]A.V. Goldberg. A simple shortest path algorithm with linear average time. In *9th Ann. European Symp. On Algorithms (ESA 2001)*, volume 2161 of *Lecture Notes in Comput. Sci.*, pages 230–241, Aachen, Germany,2001. Springer.
- [15]Y. Han. Improved algorithm for all pairs shortest paths. *Inform. Process. Lett.*, 91(5):245–250, 2004.
- [16]Yijie Han and Tadao Takaoka. An  $O(n^3 \log \log n / \log^2 n)$  time algorithm for all pairs shortest paths. In *SWAT*, pages 131–141, 2012.
- [17]D. Johnson. Efficient algorithms for shortest paths in sparse graphs. *Journal of the ACM*, 24:1–13, 1977.
- [18]L.R. Kerr. The effect of algebraic structure on the computational complexity of matrix multiplication. Ph.D.Thesis, Cornell University, 1970.
- [19]W. Liu, D. Wang, H. Jiang, W. Liu, and Chonggang Wang. Approximate convex decomposition based localization in wireless sensor networks. In *Proc. of IEEE INFOCOM*, pages 1853–1861, 2012.
- [20]S. Robinson. Toward an optimal algorithm for matrix multiplication. *SIAM News*, 38(9), 2005.
- [21]Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51(3):400–403, December 1995.
- [22]A. Shoshan and U. Zwick. All pairs shortest paths in undirected graphs with integer weights. In *FOCS*,pages 605–614, 1999.
- [23]V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 14(3):354–356, 1969.
- [24]Tadao Takaoka. A new upper bound on the complexity of the all pairs shortest path problem. *Inform.Process. Lett.*, 43(4):195–199, 1992.
- [25]Tadao Takaoka. A faster algorithm for the all-pairs shortest path problem and its application. In *COCOON*,pages 278–289, 2004.
- [26]Tadao Takaoka. An  $O(n^3 \log \log n / \log^2 n)$  time algorithm for the all-pairs shortest path problem. *Inf.Process. Lett.*, 96(5):155–161, 2005.
- [27]Mikkel Thorup. Undirected single source shortest path in linear time. In *FOCS*,

pages 12–21, 1997.

[28]Mikkel Thorup. Floats, integers, and single source shortest paths. In *STACS*, pages 14–24, 1998.

[29]Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *STOC*, pages 887–898, 2012.

[30]Raphael Yuster. A shortest cycle for each vertex of a graph. *Inf. Process. Lett.*, 111(21-22):1057–1061,2011.

[31]Uri Zwick. A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths. *Algorithmica*, 46(2):181–192, 2006.

### 三、开题报告

# 大规模网络路由的传输线路选择算法研究

姓名：李绍晓 班级：电信 1302

## 1 课题研究背景

在当今社会，网络无处不在，网络信息通过路由之间的不断转发，传达到使用者主机处。转发的过程在 IP 层进行，选择转发到哪个路由通过路由选择表来确定，而路由选择表在确定最佳路径的过程中被初始化和维护。常见的路由选择协议有 RIP、OSPF 以及 IGRP 等<sup>[1]</sup>。RIP 路由协议通过和相邻路由进行信息的交换，获取下一跳距离来动态更新确定下一条该选择的路由。OSPF 协议指开放最短路径优先协议，每个路由器通过存放链路状态的数据库，获得全网的拓扑结构，并用相关的最短路径优先（SPF，Shortes Path First）寻路算法，求出整个拓扑结构中的最短路，并给相应路由确定该转发的路径。当链路状态发生变化时，通过泛洪法来转发更新信息并构建新的数据库信息。

而随着网络通信和云计算的不断发展，企业对于网络路由的选路策略愈发重视。当确定起始路由和目标路由之后，需要通过上述介绍的路由转发过程到达目标点。假设使用的是 OSPF 协议，则需要确定好其中的 SPF 选择算法。当传输网络中的路由数达到上千时，为了减小用户时延，对选路策略的速度要求会非常高。同时因为各种条件的限制，需要在算法中考虑各种特殊情况，例如指定转发路由、通信线路出现故障等。故国内外公司投入了很大的精力，来研发更合适更高效的网络转发算法。

## 2 课题研究意义

在设计大规模网络路由的线路选择算法时，不仅仅要确定最短路径，还需要考虑其他的限制，在此过程中不断改进最短路径算法<sup>[2]</sup>。例如有时候需要规定特定的必经路由集，即转发过程必须经过必经路由集中的路由。同时，考虑到当线路上某段线缆出现问题，且无法确定是哪一段出现问题时，必须在用户

察觉之前，启动备选通路，该备选通路要求与故障线路的重边尽可能地少，尽量避开出错线缆。

由于上述 2 个问题的存在，设计一个适用的最短路径算法非常重要。

该课题在经济意义上来说，能够给企业设计新的路由选择算法提供新的参考思路，能够增加传输速率，减少传输损耗，避开通信阻塞问题。

另外，该课题考虑了 2 种特殊情况结合的单源最短路径问题，在计算机科学或图论研究中，具有一定的参考和借鉴价值。

### 3 课题研究内容与方法

#### 3.1 研究内容

本课题将研究大规模的网络路由传输线路选择问题。该问题给定一个起始路由，给定一个终点路由，要求找到最短路径。限定条件为

(i) 必须经过指定的路由集。

(ii) 找到最短路径的同时，设计一条备选线路，要求备选线路上的边尽可能地避开最短路上的边。

(iii) 总路由数最多为 2000 个，每个路由的输出线路为 20 条，必经路由数量最多为 150 个。要求极限条件下能够在 10s 内求出相应解。

关于计算机网络知识中路由是如何转发的细节将不被考虑，则该课题被建模为图论中的单源最短路径问题。故研究内容如下：

首先，对网络路由传输问题进行抽象，按照图的结构，来设计相应的数据格式和数据结构，并给出具体解释。同时根据数据格式，设计输入和输出的内容。

其次，学习求解最短路径的相关算法，并加以改进，能够在找到最短路径的同时，解决上述给出的 2 个限定条件。需要编写供该算法的测试程序，能够正确地输入和输出。开发环境为 Microsoft Visual C++ 6.0，语言为 C++。

另外，需要提供规模更大更丰富的测试样例，要求算法能够在 10s 得出结果，并统计算法和数据规模之间的规律。测试样例要具有一定的正确性和合理性。

最后，分析算法的可行性、优劣势，并编写出一篇严谨的论文。

### 3.2 研究难点

单源最短路径问题是图论算法中的经典问题，而新增的 2 个限制条件使得该问题变为一个完全 NP 问题<sup>[8]</sup>（NP，Non-deterministic Polynomial）。关于该问题，暂时没有能够求出最优解的算法，适用算法种类繁多，其中许多算法需要一定深度的数学知识和计算机科学领域知识。

同时该问题的建模、数据结构和输入输出的设计较难，对于研究者的编码功底和算法基础都有一定的要求和挑战。在设计结束后，还需要进行算法的复杂度分析，该分析过程必须要严谨且准确。

另外，当寻路问题的规模逐渐变大时，需要为算法增加许多优化、调参和剪枝，这对研究者的耐心和创造力都是一定的考验。

当算法设计完成后，还需要编写测试程序和测试样例，来检测算法的正确性和可靠性。这个过程需要进行大量的代码编写和错误调整。

### 3.3 研究方法

课题需要设计一定的数据结构。考虑到图的规模较大，用邻接矩阵存储图的结构会影响效率，故使用邻接表<sup>[3]</sup>来存储图的结构。

为了解决旅行商问题（TSP，Traveling Salesman Problem），在学术界上常用蚁群算法、模拟退火算法等<sup>[6]</sup>，蚁群算法在TSP问题领域是一种新型的优化算法，受到了各国学者的关注<sup>[7]</sup>。但由于该问题的特殊条件限制，我们采用比较经典的解决方法进行结合并加以改进，采用压缩子图的思想。<sup>[8]</sup>

单源最短路径问题的经典解法为 Dijkstra E.W.提出的著名 Dijkstra 算法<sup>[4]</sup>，这是最经典的单源点最短路径算法，用于计算一个节点到其他所有节点的最短路径。该算法的核心思想为贪心。从起始点开始，层层向外扩散，每次选出距离最短的一个点，则从起始点到该点即为最短距离，同时通过该点，对所有点进行长度的松弛和更新。

由于课题所限定的 2 个条件，单纯使用 Dijkstra 算法无法解决问题。决定采用 DFS 深度搜索<sup>[5]</sup>和 Dijkstra 相结合的方案解决该问题。DFS 深度搜索指不断对一个可能的分支进行不断地深入，直到不能深入时再进行回溯。

#### 4 课题研究工作计划

时间	课题研究工作计划
2016.12.20~2017.2.20	收集相关资料文献，学习图论相关知识；完成外文翻译、文献综述；熟悉课题，做好开题准备，有初步设计方案。
2017.2.21~3.12	参加开题交流，完成开题报告。
2017.3.13~4.30	实现算法的输入输出格式，完成算法的设计和实现，接受中期检查。
2017.5.1~5.31	系统的测试与改进，满足算法特殊要求，做出最终设计成品。撰写毕业论文初稿。
2017.6.1~6.20	论文修改，毕业答辩，提交相关文档资料。




## 参考文献

- [1] 谢希仁. 计算机网络[M]. 第六版. 北京:电子工业出版社, 2013.
- [2] 张毅, 张猛, 梁艳春. 改进的最短路径算法在多点路由上的应用[J]. 计算机科学, 2009, 36(8):205-207.
- [3] 严蔚敏, 吴伟民. 数据结构(C语言版)[M]. 北京:清华大学出版社, 2006.
- [4] Dijkstra E W. A note on two problems in connexion with graphs[J]. Numerische Mathematik, 1959, 1:269-271.
- [5] Ausiello G, Franciosa P G, Italiano G F, et al. Incremental DFS Trees on Arbitrary Directed Graphs[J]. Computer Science, 2015.
- [6] 宋青, 汪小帆. 最短路径算法加速技术研究综述[J]. 电子科技大学学报, 2012, 41(2):176-184.
- [7] 张纪会, 徐心和, 等. 一种新的进化算法--蚁群算法[J]. 系统工程理论与实践, 1999, 19(3):84-87.
- [8] 石海林. NP 完全问题多项式时间算法研究[J]. 应用数学, 2001, 5(s1):111-116.
- [9] 黄书力, 胡大裘, 蒋玉明. 经过指定的中间节点集的最短路径算法[J]. 计算机工程与应用, 2015, 51(11):41-46.

## 四、指导教师评语

浙江工业大学本科生  
毕业设计（论文、创作）指导教师评语

专业班级	电信 1302	学生姓名	李绍晓	学号	201303080511
题 目	大规模网络路由的传输线路选择算法研究				
设计（论文、创作）指导教师评语： 本文在建立数据结构中图模型的基础上，对添加限定条件的最短路径问题进行研究，并根据算法编写出测试程序，通过测试环境对算法进行验证。 作者能够较好地理解课题任务并提出合理的解决方案，理论分析准确，实验设计巧妙，见解独特。说明作者具备了较强的理论分析和实际动手能力。 论文格式严谨，图表使用合理，格式规范，达到毕业设计的要求。					
建议成绩：优					
指导教师（签字）： 					
2017 年 6 月 11 日					

注：此表一式一份，各学院自行归档，保留5年。

## 五、论文评阅人评语

# 浙 江 工 业 大 学

## 本科生毕业设计（论文、创作）评阅人评语

专业班级	电信 1302	学生姓名	李绍晓	学号	201303080511
题 目	大规模网络路由的传输线路选择算法研究				
<p>设计（论文、创作）评阅人评语：</p> <p>作者以网络路由为背景，提出了条件约束下最短路径的求解方法。论文概念定义严谨，问题描述准确，算法设计合理。</p> <p>表明作者具有较强的理论分析和数学建模能力。在算法实现上，程序设计熟练。在结果分析上，实验设计巧妙。</p> <p>论文格式规范，语言准确，文字流畅。</p> <p>论文达到了毕业设计的标准。</p> <p>设计（论文、创作）评阅人：</p> <p>2017 年 6 月 11 日</p>					

注：此表一式一份，各学院自行归档，保留 5 年。

## 六、答辩记录

# 浙江工业大学2017届毕业设计(论文)答辩记录

学生姓名	李绍晓	性别	男	专业班级	电子信息工程
指导教师姓名	王平刚	职称	副教授	学科	控制科学与工程
设计(论文)题目	大规模网络路由传输线路选择算法研究				
答辩时间	6月8日10时45分-11时15分			答辩地点	JC513
<p>答辩提问及回答情况:</p> <ol style="list-style-type: none"> <li>1. 首选路径必经节点的确定? 通过一个 demand 文件确定, 必经点是属于问题之中的条件。</li> <li>2. 实验过程中空间复杂度? 测试过程中空间足够用, 论文中分析为 <math>O(n^2)</math>。</li> <li>3. 大规模是什么样的定义? 课题所设上限为1200个。对于一个自治系统来说, 属于大规模范畴, 简单寻路在该规模下不够快。</li> <li>4. 备选路径有什么要求? 与首选路径尽可能不重叠, 再考虑尽可能短, 同时求解要快</li> <li>5. 最优算法是如何实现的? 先对总图进行压缩形成子图, 再对子图进行深度搜索, 保证能尽快得到一个可行解, 并通过相应细节处理使结果尽可能好。</li> </ol>					

答辩委员会(小组)主任(组长)签字: 王平刚 记录员: 叶耀威 日期: 2017.6.8



## 七、毕业设计成果演示记录表



# 浙 江 工 业 大 学

## 毕业设计（论文）成果演示记录表

专业班级	电信 1302	学生姓名	李绍晓	指导教师	王辛刚
题 目	大规模网络路由的传输线路选择算法研究				
根据设计任务书要求，毕业设计（论文）应完成的内容和应达到的功能	<ol style="list-style-type: none"> <li>1. 算法设计完成，并编写成论文来描述算法。</li> <li>2. 编写出测试程序，程序要求能解决大规模、必经点、备选路的问题。</li> <li>3. 提供足够多的测试用例，检验算法的性能和正确性。</li> </ol>				
成果演示时间	2017.6.6	成果演示地点	广 C517		
实际演示结果	<ol style="list-style-type: none"> <li>1. 已经将算法的思想都设计好，论文里将算法描述的很清楚。</li> <li>2. 算法测试程序能够读入 2 个文件作为条件，生成一个文件作为结果。</li> <li>3. 有能够生成测试用例的程序，并演示了该程序的使用，能够生成各种规模和条件的测试用例。</li> </ol>				

答辩小组长签名：王辛刚

注：此表一式一份，各学院（系）自行归档，保留三年。

## 八、教师指导记录表

**浙江工业大学**  
**毕业设计（论文）教师指导记录表**

专业班级	电信 1302	学生姓名	李绍晓
题    目	大规模网络路由的传输线路选择算法研究		
时    间	2016.12.20~2016.12.27		
指 导 内 容 记 录	介绍指导课题内容，解释课题难度； 规划进程安排；		
时    间	2016.12.28~2017.1.3		
指 导 内 容 记 录	指导在网上进行相关文献检索；		
时    间	2017.1.4~2017.1.10		
指 导 内 容 记 录	指导搜集下载外文论文，挑选其中两篇； 指导翻译外文文献；		
时    间	2017.1.11~2017.1.17		

指导 内容 记录	指导撰写文献综述；
时间	2017.2.20~2017.2.26
指导 内容 记录	指导撰写开题报告；
时间	2017.2.27~2017.3.5
指导 内容 记录	指导修改文献综述和开题报告；
时间	2017.3.6~2017.3.12
指导 内容 记录	指导学术信息检索技能，明确算法的分析方向
时间	2017.3.13~2017.3.19

指导 内容 记录	指导寻路算法的设计思路；
时间	2017.3.20~2017.3.26
指导 内容 记录	指导寻路算法的实现细节
时间	2017.3.27~2017.4.2
指导 内容 记录	指导测试程序的输入输出形式
时间	2017.4.3~2017. 4.9
指导 内容 记录	指导熟悉框图绘制软件 Microsoft Office Visio 的使用； 指导对算法时间复杂度进行学习,指导分析算法性能；
时间	2017.4.10~2017. 4.16

指导 内容 记录	指导算法的修改方向；
时间	2017.4.17~2017.4.23
指导 内容 记录	指导相关程序框图进行修改； 指导完成剩余部分程序编写；
时间	2017.4.24~2017.4.30
指导 内容 记录	指导算法关键部分的描述方式；
时间	2017.5.1~2015.5.7
指导 内容 记录	指导算法测试的测试思路； 指导算法测试的规模设计；
时间	2017.5.8~2017.5.14

指导 内容 记录	指导测试环境的展示、组织方式；
时间	2017.5.15~2017.5.21
指导 内容 记录	指导成果展示的方式方法；
时间	2017.5.22~2017.5.28
指导 内容 记录	指导对答辩资料和 PPT 的准备；
时间	2017.5.29~2017.6.4
指导 内容 记录	指导对毕业论文格式的修改；
时间	2017.6.5~2017.6.11
指导 内容	指导完成论文终稿；

记 录	
--------	--

指导教师签名：

王辛刚

注：根据教师实际指导情况，添加相应栏目。  
此表一式一份，各学院（系）自行归档，保留三年。



## 九、毕业设计进程考核表

# 浙江工业大学本科生毕业设计（论文、创作）进程安排与考核表

班 级	电信 1302	学生姓名	李绍晓	学号	201303080511	
题 目	大规模网络路由的传输线路选择算法研究			总进程	2016 年 12 月—2017 年 6 月 总计 20 周	
安 排 与 考 核						
起止时间		阶 段 任 务 要 点		完 成 情 况	*阶段成绩	备 注
2016.12.20~2016.12.26		熟悉课题内容		全面完成	A	
2016.12.27~2015.1.2		进行相关文献检索		全面完成	A	
2017.1.3~2017.1.9		翻译外文文献		全面完成	A	
2017.1.10~2015.1.16		撰写文献综述		全面完成	A	
2017.2.20~2017.2.26		撰写开题报告		全面完成	A	
2017.2.27~2017.3.5		修改文献综述和开题报告		全面完成	A	
2017.3.6~2017.3.12		寻路算法设计		全面完成	A	
2017.3.13~2017.3.19		设计算法思路		全面完成	A	
2017.3.20~2017.3.26		修正算法实现细节		全面完成	A	
2017.3.27~2017.4.2		实现算法输入输出		全面完成	A	
2017.4.3~2017. 4.9		绘制示例图		全面完成	A	
2017.4.10~2017. 4.16		完成算法描述		全面完成	A	
2017.4.17~2017.4.23		完成测试环境编写		全面完成	A	
2017.4.24~2017.4.30		对大规模样例调优		全面完成	A	
2017.5.1~2017.5.7		开始编写毕业论文		全面完成	A	
2017.5.8~2017.5.14		补充测试样例		全面完成	A	
2017.5.15~2017.5.21		完成毕业论文		全面完成	A	
2017.5.22~2017.5.28		准备答辩资料和 PPT		全面完成	A	
2017.5.29~2017.6.4		修改毕业论文格式		全面完成	A	
2017.6.5~2017.6.11		根据老师意见完成终稿		全面完成	A	

\*注：阶段成绩分 A、B、C 三级：A 为全面完成任务、B 为完成任务、C 为完成任务不好

指导教师 王幸刚

2017 年 6 月 11 日

## 十、毕业论文的“知网”检测结果



## 文本复制检测报告单(全文标明引文)

ADBD2017R\_20170527170902427927192025

检测时间: 2017-05-27 17:09:02

检测文献: 大规模网络路由的传输线路选择算法研究

作者: 李绍晓

检测范围:

中国学术期刊网络出版总库  
中国博士学位论文全文数据库/中国优秀硕士学位论文全文数据库  
中国重要会议论文全文数据库  
中国重要报纸全文数据库 中国专利全文数据库 互联网资源(包含贴吧等论坛资源)  
英文数据库(涵盖期刊、博硕、会议的英文数据以及德国 Springer、英国 Taylor&Francis 期刊数据库等)  
港澳台学术文献库  
优先出版文献库  
互联网文档资源  
图书资源  
CNKI 大成编客-原创作品库  
大学生论文联合比对库  
人比对库

时间范围: 1900-01-01 至 2017-05-27

指导教师: 王辛刚

### 检测结果

总文字复制比: 9.7% 跨语言检测结果: 0%

去除引用文献复制比: 9.4% 去除本人已发表文献复制比: 9.7%

单篇最大文字复制比: 3.6%

重复字数: [2154] 总字数: [22195] 单篇最大重复字数: [800]  
总段落数: [7] 前部重合字数: [535] 疑似段落最大重合字数: [1078]  
疑似段落数: [4] 后部重合字数: [1619] 疑似段落最小重合字数: [218]

指标: ☐ 疑似剽窃观点 ☒ 疑似剽窃文字表述 ☐ 疑似自我剽窃 ☐ 疑似整体剽窃 ☐ 过度引用

表格: 0 脚注与尾注: 0

6.9% (218) 中英文摘要等 (总 3141 字)  
21.2% (513) 第 1 章绪论 (总 2422 字)  
0% (0) 第 2 章问题的描述和定义 (总 1378 字)  
58.4% (1078) 第 3 章所用经典算法介绍 (总 1846 字)  
5.8% (345) 第 4 章经过指定节点集且重叠边尽可能少的寻路算法 (总 5967 字)  
0% (0) 第 5 章算法测试 (总 3006 字)  
0% (0) 第 6 章总结 (总 4435 字)



(注释: 无问题部分 文字复制比部分 引用部分)

### 1. 中英文摘要等

总字数: 3141

相似文献列表 文字复制比: 6.9%(218) 疑似剽窃观点: (0)

1	薛思威_200903110323_通信工程(徐志江, 毕业论文) 薛思威 - 《大学生论文联合比对库》- 2013-05-27	6.9% (218) 是否引证: 否
2	薛思威_200903110323_通信工程(徐志江, 毕业论文) 修改版 薛思威 - 《大学生论文联合比对库》- 2013-06-06	6.9% (218) 是否引证: 否