浙江工业大学

本科毕业设计论文

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

浙江工业大学本科毕业设计论文

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

浙江工业大学信息工程学院

2017 年 6 月

Dissertation Submitted to Zhejiang University of Technology for the Degree of Bachelor

Research on Transmission Line Selection Algorithm for Large Scale Network Routing

Student:Li Shaoxiao

Advisor: Wang Xingang

College of Information Engineering

Zhejiang University of Technology

June 2017

浙江工业大学

本科生毕业设计(论文、创作)诚信承诺书

本人慎重承诺和声明：

1. 本人在毕业设计（论文、创作）撰写过程中，严格遵守学校有关规定，恪守学术规范，所呈交的毕业设计（论文、创作）是在指导教师指导下独立完成的；

2. 毕业设计（论文、创作）中无抄袭、剽窃或不正当引用他人学术观点、思想和学术成果，无虚构、篡改试验结果、统计资料、伪造数据和运算程序等情况；

3. 若有违反学术纪律的行为，本人愿意承担一切责任，并接受学校按有关规定给予的处理。

学生（签名）：

年　　月　　日

浙 江 工 业 大 学

本科生毕业设计（论文、创作）任务书

专 业 电子信息工程 班 级 电信1302 学生姓名/学号 李绍晓/201303080511

一、设计（论文、创作）题目：

大规模网络路由的传输线路选择算法研究

二、主要任务与目标：

研究并实现网络路由线路选择算法,利用图论知识，转化为特定模型分析。该算法要求在规定时间内，解决大规模、必经点、备选路径等3个问题。

三、主要内容与基本要求：

1.了解并研究最短路的图论原理;2.设计算法的输入和输出格式;3.研究并实现寻路算法;4.改进算法，能够解决大规模、必经点、备选路问题。 5.提供充足测试样例，并给出结果和说明，证明算法的可靠性。6.撰写毕业论文和提交算法设计文档、测试数据、程序流程图等。

四、计划进度：

2016.12.20~2017.02.20 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； 2017.02.21~2017.03.05 完成开题报告，参加开题交流；2017.03.06~2017.04.23 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；2017.04.24~2017.05.28 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； 2017.05.29~2017.06.12 论文修改，毕业答辩，提交相关文档资料。

五、主要参考文献：

[1] 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. [2]Dijkstra E W.A note on two problems in connexion withgraphs[J].Numerische Mathematik，1959，1：269-271. [3]Goldberg A V，Kaplan H，Werneck R F.Reach for A*：efficientpoint-to-point shortest path algorithms[C]//ALENEX,2006.6（2）：129-143.

任务书下发日期 2016 年 12 月 20 日

设计（论文、创作）工作自 2016 年 12 月 20 日 至 2017 年 6 月 4 日

设计（论文、创作）指导教师 王辛刚

系主任（专业负责人） 王辛刚

主管院长 张兵 印有

大规模网络路由的传输线路选择算法研究

摘　要

近年来，随着通讯和云计算的发展，网络信息在 IP 层传输时，对速度和时间的要求愈发严格。特别是在大规模路由中选择一条尽可能短的通路时，需要一个优良、高效率的算法，以满足工业及商业需要。这个问题可转化为最短路径问题，该问题已经出现了许多经典或成熟的算法。但由于路由传输的一些特性，需要为该问题添加一些限定条件。在这一背景下，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。本文的主要研究工作如下：

（1）设计并给出严谨的概念定义和问题定义，以便理解算法，并给出输入输出的格式，通过一个例子，来便于该问题的理解。

（2）介绍该算法的预备知识。先粗略介绍算法整体思想，再介绍经典寻路算法的概念和用法，接着介绍深度搜索 DFS 的概念和定义，以及需要提及的贪心思想等。

（3）详细介绍算法的原理。先描述算法的原理、流程，并给出伪代码和流程图，接着给出一个例子，演示算法的运行过程。最后给出算法时间复杂度分析。

（4）通过测试结果，分析算法的可行性和普适性。

关键词：路由选择，单源最短路径，旅行商问题，Dijkstra 算法，DFS 深度搜索

RESEARCH ON TRANSMISSION LINE SELECTION ALGORITHM FOR LARGE SCALE NETWORK ROUTING

ABSTRACT

In recent years, with the development of communication and cloud computing, when network information transmit in the IP layer, the speed and time requirements become more stringent. In particular, when selecting a shortest path in a large-scale route, an excellent and efficient algorithm is needed to meet industrial and business demand. This problem can be transformed into the shortest path problem, and the problem has arisen with many classical or mature algorithms. However, we need to add some restrictions for the problem because of some characteristics of routing transmission. In this context, based on the establishment of the graph model in the data structure, the shortest path problem of adding qualified conditions is studied, and the test program is written according to the algorithm, and the algorithm is verified by the test environment. The main works of this paper are as follows:

(1) Design and give a rigorous definition of the concept and the definition of the problem in order to understand the algorithm, and give the input and output format, through an example to facilitate the understanding of the problem.

(2) Introduce the preliminary knowledge of the algorithm. First introduced the concept of the overall idea of the algorithm, and then introduce the classic algorithm Dijkstra's concept and usage, then introduce the concept of depth search tconcept and definition, and finally tell our algorithm how to pre-processing input, which will design their own data structure.

(3) Details the principle of the algorithm. First describe the principle of the algorithm, the process, and give pseudo-code and flow chart, and then give an example of the operation of the demonstration algorithm. Finally, the time complexity analysis of the algorithm is given.

(4) Through the test results, analyse the feasibility and universality of the algorithm.

Key Words: routing choice, single source shortest path, traveling salesman problem, dijkstra algorithm, DFS depth search

目　　录

第 1 章　绪　论

1.1　课题研究的背景和意义

1.1.1　课题的研究背景

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，一片包含一定数量的网络系统称为自制系统（AS，Autonomous System），在同一个自治系统内，采用同一个内部选择协议。

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[1]。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。本课题涉及的路由选择算法为开放最短路径优先协议（OSPF，Open Shortest Path First），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。

1.1.2　课题的研究意义

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[2]。

我们参考如图 1-1 所示的场景图。

首先，由于通信和网络的不断发展，可能会出现一片包含上千路由的自制系统。故我们假定一个大规模的场景，其系统内路由数量最大可达到 1200 个。系统内路由数量

的增大会使得寻路效率变慢。

而对于网络层中路由的传输选择算法，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（图中的大路由器）。

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，以尽可能避开出错线缆。

同时由于传输的时效性特点，为了优先保障网络线路的传输，必须在尽可能快的时间内得到一个可行解。故能否尽快得出第一个结果，也是一个重要考察因素。

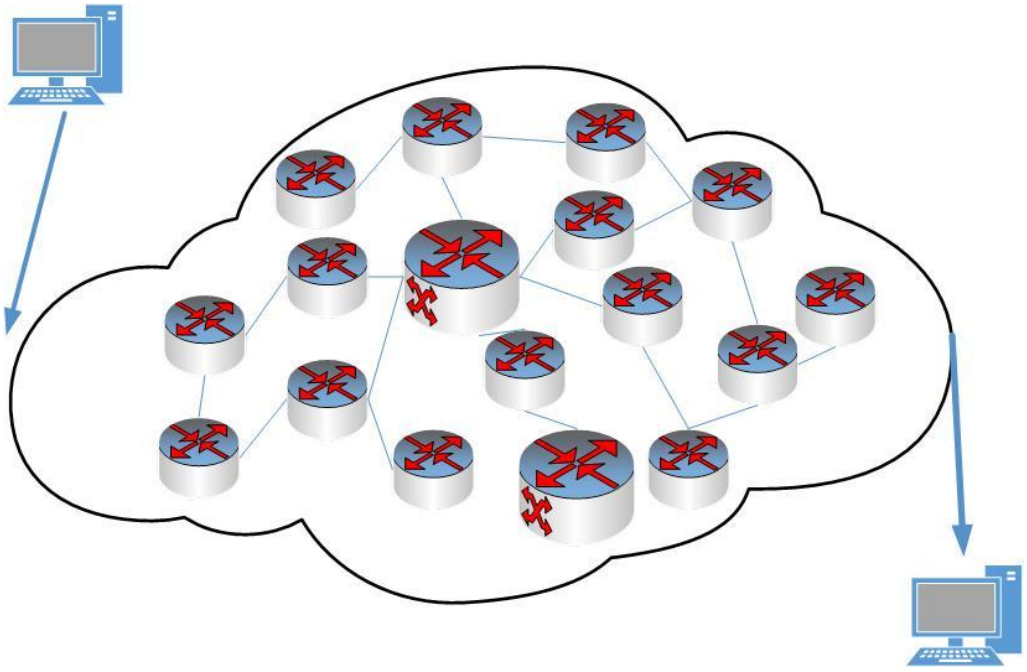

图 1-1　课题应用场景示例图

将这 3 个实际问题中需要考察的因素结合起来，转化为一系列的限定条件，即可成为一个具有特殊性却又实际意义的最短路传输问题。

如果能拥有解决该问题的优良算法，能够给企业提供一定的借鉴意义，或为今后的相关问题提供一些参考。

1.2　类似问题的研究现状

该问题类似于旅行家问题（TSP，Traveling Salesman Problem）。旅行家定义问题如下：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[3]。此问题求解复杂，在特大规模的图中，是一个 NP 完全问题（NP，Non-deterministic Polynomial），即至今所解决该问题的算法都是非

确定性算法。

关于该问题，单独使用 Dijkstra 算法是无法解决的。80 年代初，模拟退火算法被提出[4]。模拟退火其实也是一种贪心算法，但是它的搜索过程引入了随机因素。模拟退火算法以一定的概率来接受一个比当前解要差的解，因此有可能会跳出这个局部的最优解，达到全局的最优解。以图 1-2 为例，模拟退火算法在搜索到局部最优解 A 后，会以一定的概率接受到 E 的移动。也许经过几次这样的不是局部最优的移动后会到达 D 点，于是就跳出了局部最大值 A。

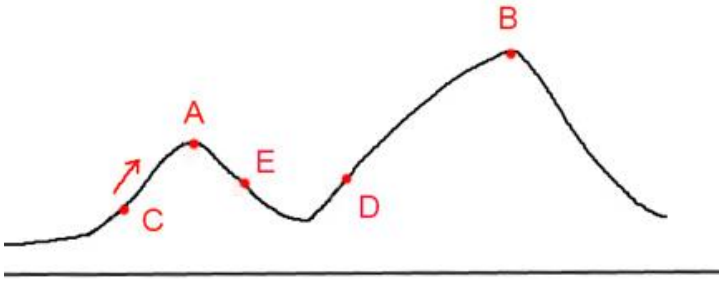

图 1-2　模拟退火算法示例图

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[5]（ACO，ant colony optimization），并成功地用于求解旅行商问题。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，我国最早研究蚁群算法的是东北大学张纪会博士和徐心和教授。

1.3　大规模网络路由寻路算法的前景分析

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，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。本次课题能够为特定条件下的大规模路由转发问题提供一定的参考价值。

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[6]，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。

对于智能化的未来，最短路径问题将始终是计算机科学领域的重要课题。

1.4 本文主要研究内容及结构

本课题主要研究的是以网络路由选路为背景的受限制最短路径问题，包括了算法和测试环境的设计。主要内容如下：

第一章绪论。主要介绍了该问题的研究背景和意义，就其特点进行分析，描述了它在商业及工业领域的潜在价值，对其发展趋势做了预测。并且还介绍了该问题的研究现状，给出了一些经典的求解该问题的方法。

第二章给出该问题的概念和定义，以便理解算法，并给出输入输出的格式，通过一个简单的例子来便于该问题的理解。

第三章介绍该课题中所参考的一些经典算法，描述了 Dijkstra 的概念和用法，接着介绍深度搜索 DFS 的概念和定义，以及搜索过程中所用到的贪心思想。

第四章详细介绍课题自主设计的算法原理。先描述算法的整体思想，给读者一个算法整体性的认识，再给出算法的所需数据结构和预处理过程。接着再介绍算法的核心部分：图压缩和深度搜索，给出详细的解释，阐述其中的算法核心和创新之处，并依次给出伪代码。之后再给出一个简单的例子，演示算法的运行过程。最后给出算法时间复杂度分析。

第五章给出测试环境以及测试流程、工具，先通过一个设计好的例子，详细展示算法的运行过程，展示算法的所有关键之处。其次一个自动化的测试样例生成器，产生足够的测试用例，整理成表格，并以此分析算法的可行性和普适性。

第六章总结了本文的工作内容，分析了做这个课题的原因，以及工作方法和成功之处，并且总结当前该问题的发展情况，提出了文章的不足之处，进一步给出了解决方案，并对今后的工作做出展望。

第 2 章　大规模网络路由寻路问题的描述和定义

2.1　大规模网络路由寻路问题的概念

如图 2-1 所示，对于该网络示例拓扑图，我们进行一些具体概念的定义。

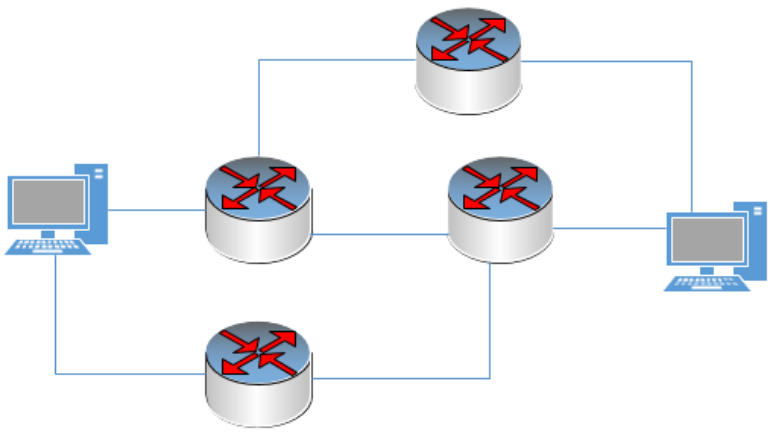

图 2-1　网络拓扑示例图

顶点集V为某网络中的所有路由。有向边集E为路由和路由之间的通路集合。

而顶点s相当于发送请求主机所连接的起始路由，顶点t相当于接受请求主机所连接的路由。

通过算法，得到的从s到t所选择的传输路径为路径P。

必经点集V：相当于必经路由集，即从s到达t的过程中，必须经过V中的所有顶点。路径P必须经过必经点集V。

首选最短路径P'和必经点集V'对应，备选路径P''和必经点集V''对应。

结合抽象出的概念，在下面给出具体的问题定义。

2.2　大规模网络路由寻路问题的定义

已知路由的拓扑图，我们规定了起始路由和终止路由，同时也规定了中间必须经过的路由集。我们要找到从起始路由到终止路由的首选路径和备选路径，要求首选路径和备选路径的重叠边尽可能地少，以便首选路出错时，能替换至备选路保持通讯的畅通。

抽象网络路由概念后的描述如下：

给定一个带权重的有向图$G=(V,E)$，V为顶点集，E为有向边集，每一条有向边均有一个权重l。对于给定的顶点s、t，以及V的子集V'和V''，寻找从s到t的两条不成环的有向路径P'和P''，使得P'经过V'中所有的顶点，而P''经过V''中所有的顶点（对P'经过V'中顶点的顺序以及P''经过V''中顶点的顺序不做要求）。

若不同时存在这样的两条有向路径，则输出无解；若存在，则两条有向路径的重合边数量越少越好。同时尽量能够在较短的时间内先得到一个可行解，再去求更好的解。

说明：

（1）图中所有权重均为[1，100]内的整数。

（2）任一有向边的起点不等于终点。

（3）连接顶点A至顶点B的有向边可能超过一条，权重可能一样，也可能不一样。

（4）该有向图的顶点不会超过 1200 个，每个顶点出度（以该点为起点的有向边的数量）不超过 20。

（5）V'和V''中元素个数均不超过 60 个，交集为空，且不包含起始顶点s和终止顶点t。

（6）从s到t的不成环有向路径P是指，P为由一系列有向边组成的从s至t的有向连通路径，且不允许重复经过任一顶点。

（7）路径的权重是指所有组成该路径的所有有向边的权重之和（重复边的权重应分别在两条路径中各计算一次）。

2.3 大规模网络路由寻路问题的举例

在如图 2-1 所示的网络拓扑图中，我们将信息抽象化，转化为一个有向图，如图 2-2 所示：

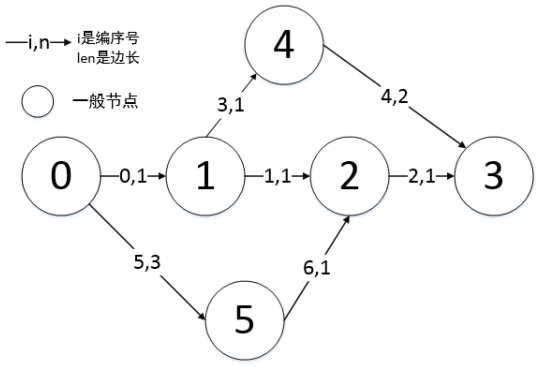

图 2-2　大规模寻路问题的示例图

该有向图转化为数据时的信息，如表 2-1 所示。

表 2-1　示例图信息

边序号	起始点	终止点	边长
0	0	1	1
1	1	2	1
2	2	3	1
3	1	4	2
4	4	3	2
5	0	5	3
6	5	2	1

该信息转化为图的形式如下：

如果此时需要寻找从 0 到 3 的路径P'和P''，且P'必须经过顶点 1，P''必须经过顶点 2，故相应的需求信息文件内容表 2-2 所示：

表 2-2　寻路问题的示例需求信息

序号	起点	终点	必经点
1（首选路）	0	3	1
2（备选路)	0	3	2

对于该用例，可得到两组解。

第一组解为：经过边 0|1|2 的首选路，经过边 5|6|2 的备选路。总长为 9。

第二组解为：经过边 0|3|4 的首选路，经过边为 5|6|2 的备选路。总长为 10。

由于第一组解两条路径的重合边个数为 1，第二组解两条路径的重合边个数为 0。故即使第二组解的总长较长，但此时的最优解应是第二组解。

第 3 章　相关经典算法介绍

本文所提出的算法，是结合了以下 2 种经典算法的变形。故在此先介绍这 2 种经典算法的思想和意义，以及会用到的思想。

3.1　经典算法 Dijkstra

1959 年，Dijkstra EW.A 提出了著名的 Dijkstra 算法[7]，这是最经典的单源点最短路径算法，该算法用于计算一个点到图中其他各点的最短距离。

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。

其算法步骤为：

（1）首先，引入一个辅助向量D，它的每个分量$D[i]$表示当前所找到的 从起始点（即源点 ）到其它每个顶点 的长度。例如，$D[3]=2$表示从起始点到顶点 3 的路径相对最小长度为 2。这里强调相对就是说在算法执行过程中 D 的值是在不断逼近最终结果但在过程中不一定就等于长度。

（2）D的初始状态为：若从v 到$v[i]$有弧（即从v到$v[i]$存在连接边），则$D[i]$为弧上的权值（即为从v到$v[i]$的边的权值）；否则置$D[i]$为∞。显然，长度为 $D[j]=Min\{D\ |v[i]\in V\}$ 的路径就是从v出发到顶点$v[j]$的长度最短的一条路径，此路径为$(v,v[j])$。

（3）找到从源点v到下一个顶点的最短路径长度所对应的顶点，且这条最短路径长度仅次于从源点v到顶点$v[j]$的最短路径长度。假设该次短路径的终点是$v[k]$，则可想而知，这条路径要么是$(v,v[k])$，或者是$(v,v[j],v[k])$。它的长度或者是从v到$v[k]$的弧上的权值，或者是$D[j]$加上从$v[j]$到$v[k]$的弧上的权值。

（4）一般情况下，假设S为已求得的从源点v出发的最短路径长度的顶点的集合，则可证明：下一条次最短路径（设其终点为x）要么是弧(v,x)，或者是从源点 v 出发的中间只经过 S 中的顶点而最后到达顶点 的路径。 因此，下一条长度次短的的最短路径长度必是$D[j]=\ Min\{D[i]\ |v[i]\in V-S\}$，其中$D$ 要么是弧$(v,v[i])$上的权值，或者是$D[i](v[k]\in S)$和弧$(v[k],v[i])$上的权值之和。

以图 3-1 为例：

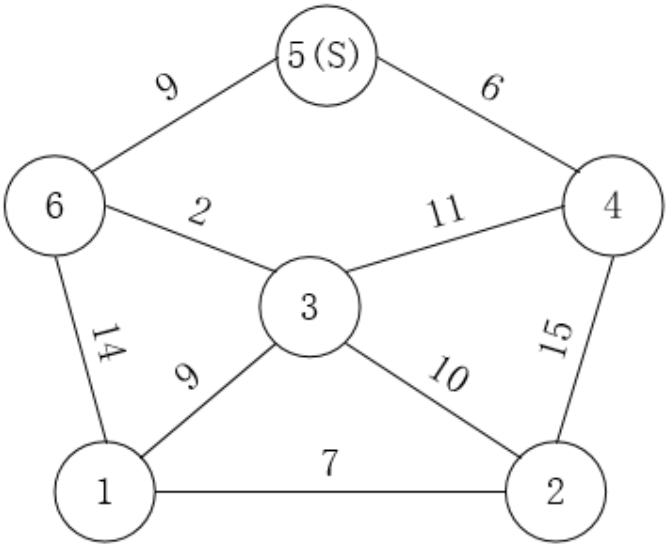

图 3-1　Dijkstra 算法的演示图

起点为首，先初始化所有$d[i]=\infty$（即不可达）。

因为 5 号点为起点，故$d[5]=0$。故第一次先选取 5 号点，然后进行更新，因为与 5 号点相邻的是 6 号点和 4 号点，故更新$d[6]=9$，$d[4]=6$。

第二次，$d[4]$此时最小，选取 4 号点进行更新，使得$d[3]=17$，$d[2]=21$。

第三次，$d[6]=9$在此时最小，故选取 6 号点，更新得$d[3]=11$，$d[1]=23$。

第四次，$d[3]=11$在此时最小，故选取 3 号点，更新$d[1]=20$，$d[2]=21$。

第五次，$d[1]=20$在此时最小，故选取 1 号点，不更新 2 号点。

第五次即选择最后的 2 号点，$d[2]=21$。

故最后结果为$d[1]=20$，$d[2]=21$，$d[3]=11$，$d[4]=6$，$d[5]=0$，$d[6]=9$，即 5 号点到各个点的距离。

该经典算法仅仅以长度为判断指标，后面将对该算法进行相应的改进，以求解本文所提出的问题。

3.2　深度遍历搜索 DFS

事实上，深度优先搜索（DFS，Depth First Search）属于图算法的一种[8]，其过程简要来说是对每一个可能的分支路径深入到不能再深入为止，而且每个节点只能访问一次。该算法常用于暴力搜索、迷宫搜索以及树的遍历等。

举例说明之：如图 3-2 是一个无向图，如果我们从 A 点发起深度优先搜索（以下的

访问次序并不是唯一的，第二个点既可以是 B 也可以是 C，D)，则我们可能得到如下的一个访问过程：A→B→E（没有路了！回溯到 A）→C→F→H→G→D（没有路，最终回溯到 A，A 也没有未访问的相邻节点，本次搜索结束)。

对于 DFS 算法，其很容易陷入局部搜索，尤其是对应规模特别大的图。故需要增加一些剪枝、路径选择等优化算法，增加搜索的效率。.

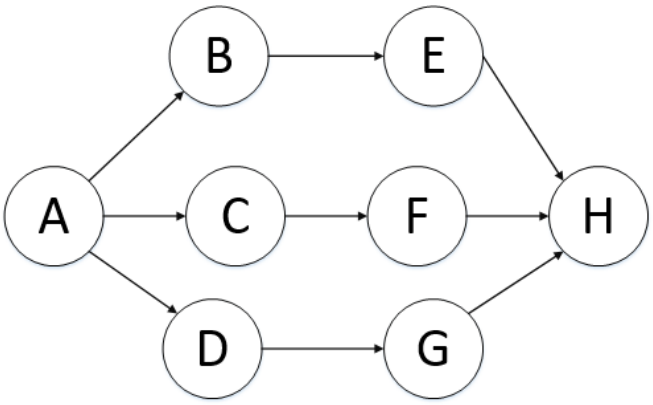

图 3-2　DFS 算法的演示图

3.3　贪心算法

贪心算法（又称贪婪算法）是指，在对问题求解时，总是做出在当前看来是最好的选择[9]的一种策略。也就是说，不从整体最优上加以考虑，他所做出的是在某种意义上的局部最优解。

贪心算法不是对所有问题都能得到整体最优解，关键是贪心策略的选择，选择的贪心策略必须具备无后效性，即某个状态以前的过程不会影响以后的状态，只与当前状态有关[1]。

第 4 章　经过指定节点集且重叠边尽可能少的寻路算法

4.1　寻路算法描述

针对课题所需解决的问题，我们提出了一个算法，并命名该算法为“经过指定节点集且重叠边尽可能少的寻路算法”。为了方便，以下皆简称寻路算法。

4.1.1　寻路算法的整体思想

我们考虑到研究所要求的图中总节点数较大，最多有 1200 个，而必经点最多仅 60 个。针对这个差异，我们想到可以先进行压缩，压缩为必经点之后，子图的规模相对于原图来说较小，再利用深度搜索，能够在一个较快的时间内，得到一个可行解。

该算法思想可描述为四步：

对于图$G = (V, E)$，　必经点集为V'，步骤为：

（1）将拓扑图数据读入相应数据结构，并进行预处理。

（2）将原图 G 通过处理，压缩简化为一个子图[9]，该子图的顶点集 V 仅包含起点、终点、必经点集，通过 CompressRoad 操作（修改的 Dijkstra 算法），建立各点间的边。

（3）对该图进行 DFS 深度搜索，利用贪心策略以及相关优化搜寻符合要求的路径，并在规定时间内继续搜索更优的解。

（4）在首选路的搜索结束之后，再进行备选路的搜索。通过路径重复值策略，能够使备选路尽量避开首选路中的边。

我们以如图 4-1 所示的拓扑图，介绍第（1）步的压缩思想。

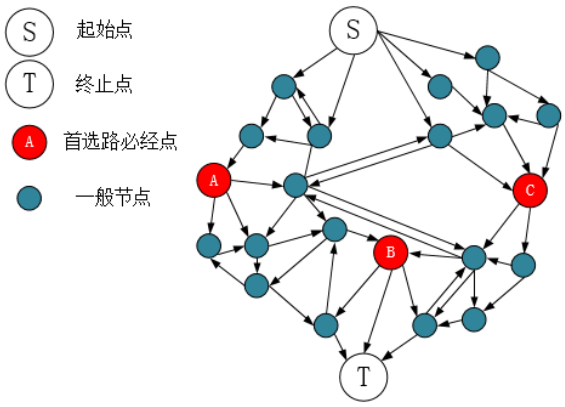

图 4-1　寻路算法整体思想的示例拓扑图

该图的起点为 S，终点为 T，必经点为 A、B、C 三点。

我们以 S 点为起点，寻找 S 点分别到 A、B、C 三点的最短路径，并将这 3 条最短路径整合为 S 点的边，如图 4-2 所示。

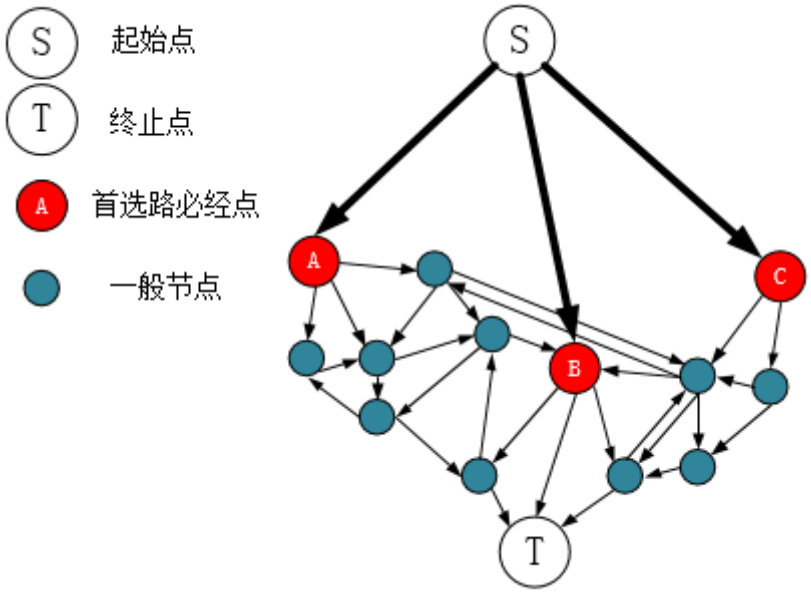

图 4-2　S 点对 ABC 三点寻最短路并整合后的拓扑图

接着，我们以 A 点为起点，寻找 A 点分别到 S、B、C、T 三点的最短路径，并整合，如图 4-3 所示。

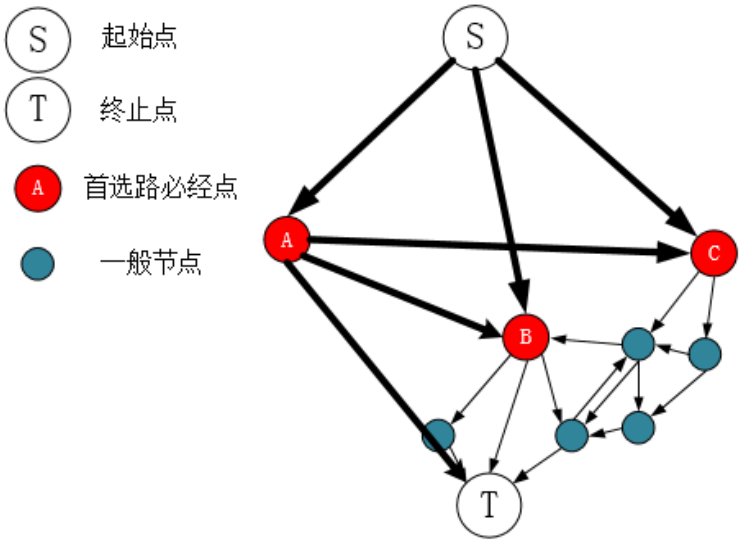

图 4-3　A 点对 BCT 三点寻最短路并整合后的拓扑图

我们以同样方法处理 B 点和 C 点，T 点由于是终点，不进行处理。图 4-4 即压缩后的子图。

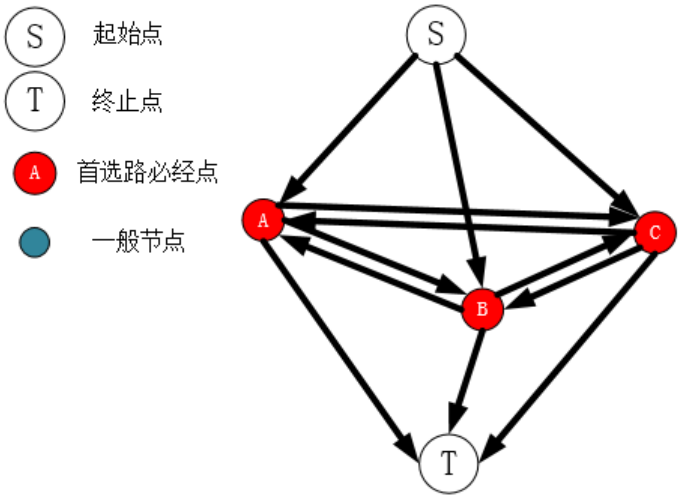

图 4-4　各自寻完最短路后的压缩子图

压缩子图后，我们能够将注意力集中在必经点的选择上，排除其他无关点的干扰。随后便可进行 DFS 搜索，具体搜索方法见 4.1.3。

4.1.2　寻路算法的数据结构和预处理

关于数据结构，我们主要体现于图的存储和必经点的存储。

对于图的存储，由于各点的出度最高为 20，但是顶点数最高达 2000 多，若使用邻接矩阵存储，则会增加不必要的空间和时间。故使用邻接表。$Head(v)$是每个顶点的头节点，$road(i)$是一个结构体，存储每条路 i 的信息，每条路通过$road(i).next$与 Head(v)进行连接。故若要知道从顶点 v 为起点有多少条边，只需要从$Head(v)$开始，通过$next$进行遍历即可。

对于必经点集的存储，我们用$get_need(v')$存储必经点v'在必经点集中的序号$need_n$。$need_juge(need_n)$存储序号所对应的必经点v''。备选路的v''存储同理。这样做方便在算法中判断必经点的排序序号，以及查找必经点的图序号。

当针对备选路的必经点集存储结束后，我们会针对备选路的必经点v''进行依次判断，当某必经点p的出度为 1 时，增加该点那条边的重复值权重，并引申至前后皆为出度为 1 的点和边。

我们可以参考图 4-5，点 B 为备选路的必经点，点 B 的出度为 1，故入边和出边的重复值权重皆加 1。同时由于点 B 的下一个点 C 的出度也为 1，故 C 的出边的重复值权重也加 1。

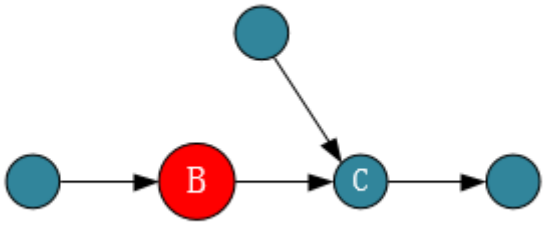

图 4-5　需要预处理的情况示例图

该操作的意义为：由于备选路的必经点限制，导致被备选路必须经过这些边，故首选路在搜索时，需要尽量避开这些边。这是该算法预处理部分相当关键的一个步骤，避免了非常多不必要的重边情况。

4.1.3　CompressRoad 算法（修改版 Dijkstra）

该算法名为 CompressRoad (s,findn)，s 为算法中的扩散起始点，findn 为所要查找的必经点数量。

该算法用于求从 s 点到其他各点 v 的最短距离，并将路径信息存入 NeedRoad(s,v)，找到 findn 个点即停止。

由于压缩时，需要确定的是某点到其他所有各点的最短距离，且要保证一个较快的速度，Dijkstra 算法正好能够解决这个问题，但必须进行相应的修改和优化。

该算法需要考虑到 2 点，一是如何在 Dijkstra 的基础上进行路径压缩，二是如何按照课题所给条件，进行特定需求的处理。

描述为文字时的具体流程如下：

（1）初始化，令 s 到其他点的所有距离和重复值为无穷。

（2）找到重复值、距离最小的点 select。

（3）如果该点 v 为必经点，则为 s->v 生成一条最短路径并保存进 NeedRoad(s,v) 中。若该算法已生成 findn 条路，则退出算法。

（4）若是必经点，直接跳转到（2）。否则进行松弛和更新操作，判断条件依旧为重复值和距离。跳转到（2）。

该算法的伪代码如下：

```
CompressRoad(s,findn)
1:for each v 属于 V do
2:    dis(v)=无穷，renum(v)=无穷
3:end for
4:dis(s)=0 ； renum(s)=0
5:for(v 属于 V)
//优先判断重叠权值 re 是否最小，再判断 dis 是否最小
6:    find min(dis(v) and renum(v))
7:    select=v
8:end for
//若找到的该点在必经点集中，进行增路操作
9:if ( need(select) ) then
10:    add_road(s,select)
//如果 findn 为 0，说明需要搜索的点已经搜完，则结束该算法
16:    findn=findn-1
17:    if (findn = 0) then
18:        return;
19:end if
20:outr=Head(select)
21:for (outr.next 不等于 -1)
//根据 outp 点的重复权值和距离进行更新，权值优先。
22: outp=road[outr]
22: if ( juge(s,outp) )
24:    update(outr)
25：end if
26:end for
```

与传统 Dijkstra 算法的不同之处在于：

（1）当被选中的最短长度点为必经点 p 时，将进行“生成路”的操作。即将 s 到 p

之间所经过的边，以及路径长，存入NeedRoad(s,v)中，即 s 到 v 之间的路多一条。

（2）在松弛和更新操作中，不再仅仅以路的总长来判断是否更新路径，而是通过路径的重叠权值 re 和路径距离 dis 来综合判断。

（3）不更新已经过必经点的路径，减少重复率，方便图的压缩。

该算法的流程图如图 4-6 所示：

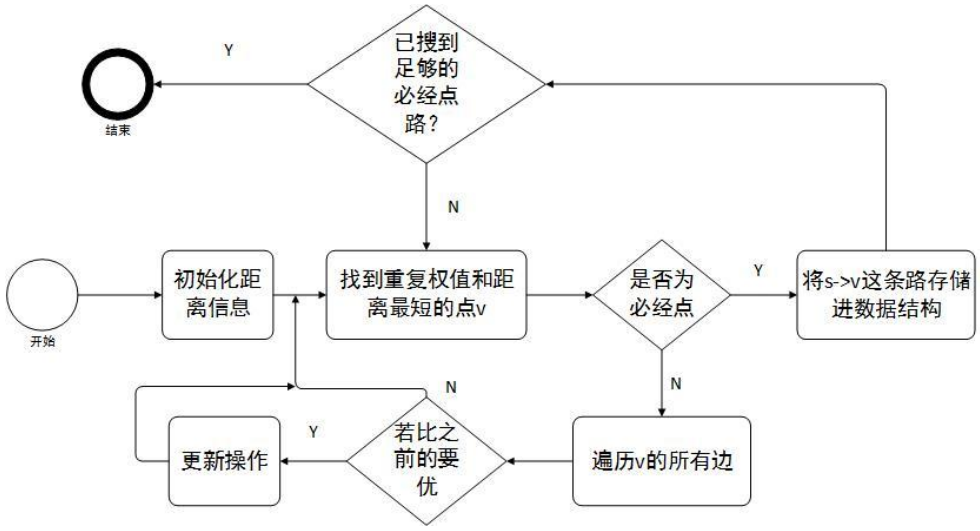

图 4-6　必经点寻路 CompressRoad 算法流程图

4.1.4　深度搜索及剪枝

在子图建立后，对于应采用何种搜索方式，我们需要考虑研究课题的实际意义。对于网络传输，通讯连通性的意义大于通讯速度。故在一个较短时间内尽快求出一个可行解较为重要。如果采用广度遍历、蚁群算法等依靠层层扩撒来进行搜索的方法，能够得到一个较为优质的结果，但所需要的空间和时间消耗都非常大。故我们采用 DFS 深度搜索。因为子图中都是必经点，按照任意方向搜索下去，有较大概率搜到最终点，而避开了其他无用搜索。

当考虑搜索时，不可直接进行深搜，直接深搜的搜索效果和效率都较差。我们通过贪心思想，每次选择“局部最优”的那个选择向下搜索，这样有助于在图的规模过大时，尽快搜得一条“较优”的所需路。同时为了避免陷入“局部最优”，当搜得一条所需路时，我们进行回溯，回溯到x层进行下 1 次搜索。

另外，当图中的各点出度不高时，会导致各点之间有足够的关联性，故压缩图时所选择的路径，会相互之间存在冲突。如图 4-7 所示。

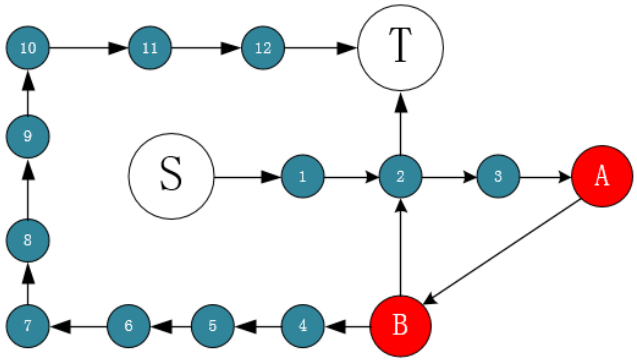

图 4-7　需要重寻路时的样例图

假设此时已经搜索到必经点 B，之前所行走的路线为 S→1→2→3→A→B。对于必经点 B，所应走的路是 B→2→T，因为这条路较短，在压缩时选择了这条路。但由于 2 号点已经被走过，故存在走不通的情况。所以我们进行重寻路操作，最终选择了 B→4→5→6→7→…→12→T 这种较远的走法。在算法中我们假定当此时可选择的路中有一半以上的路不可走时，我们进行针对该点进行重寻路。

故搜索算法的具体描述如下：

（1）若当前最短路已找到，退出，向上回溯。

（2）若搜索到终点，且满足条件“经过所有必经点”，则和当前最短路进行比较，若重复值小且长度短，则将这条路更新至当前最短路，并退出。若不是终点，则进入（3）。

（3）提取出接下来要可选择的路（压缩之后），可选择路的数量为 AR（all road）。检查每条路是否可走（即判断该路所经过的点是否已被访问），若不可走，统计值 NPR（no pass road）加 1。

（4）检测 NPR 是否大于等于 AR 的一半。若大于，说明不可走的路过多，进行$CompressRoad$操作，更新可选路的集合。

（5）对可选择路进行排序，排序规则为重复值优先，其次是路长度优先。

（6）从长度最短的路开始遍历，选择这条路后，得知该路终点为e，进行dfs(e)，进入下一层搜索。

（7）当从dfs(e)的这层搜索退出时，检查是否已退回第x层，若当前是x层，则重置 minlen，继续向下搜索，直到下次搜到更短的路后再回溯。

这里总结一下该深搜的剪枝和优化要点为：

1.当在某一点可选择的路中，有超过一半已经“不可走”时（一般是因为深搜过程

中，已经过的点占据了这些路中的点），针对该点进行再次路径压缩，称之为“点间路重寻”。

2.将某一点可选择的路进行排序，选择最优的那条，最优的判断为重复权值、长度总和考虑。即利用“贪心”思想进行向下搜索。

3.搜到可行解时，选择回溯到第x层再进行继续搜索，避免陷入局部最优。这里的x取值可以设置为随着图形规模调整，但重点还是在于尽快求出可行解。

4.利用重复边数和路径总长综合考虑可行解是否应被设置为当前最优解。

伪代码如下：

```
dfs(v,levelinfo)
1：if (minlen 不等于无穷) then //说明以及找到最短路，需要回溯
2：    return
3：end if
4：if (v is endv && juge()) //若 v 是终点且满足条件
5：      getup(minlen,anslen); //更新 minlen 和 anslen
6：end if
7：for ( r in needroad(v))
8：      if( FailCheck(r) ) then
9：           NPR=NPR+1 else save(road)
10：     end if
11：end for
12:if (NPR > APR/2) then    //若不可走的路多于一般，针对该点再次压缩
13:       CompressRoad (v)
14:end if
15:sort(road)     //对可选择的方向进行排序
16:for(r in road)
17:     dfs(r.e,NextLevelInfo)
18:     if(levelInfo.level  等于  x) then
19:          minlen=无穷,stop  回溯，接着搜下一条
20：     end if
21:end for
```

4.1.5 总算法流程图

如图 4-8 所示，给出整个算法的全部求解流程图。

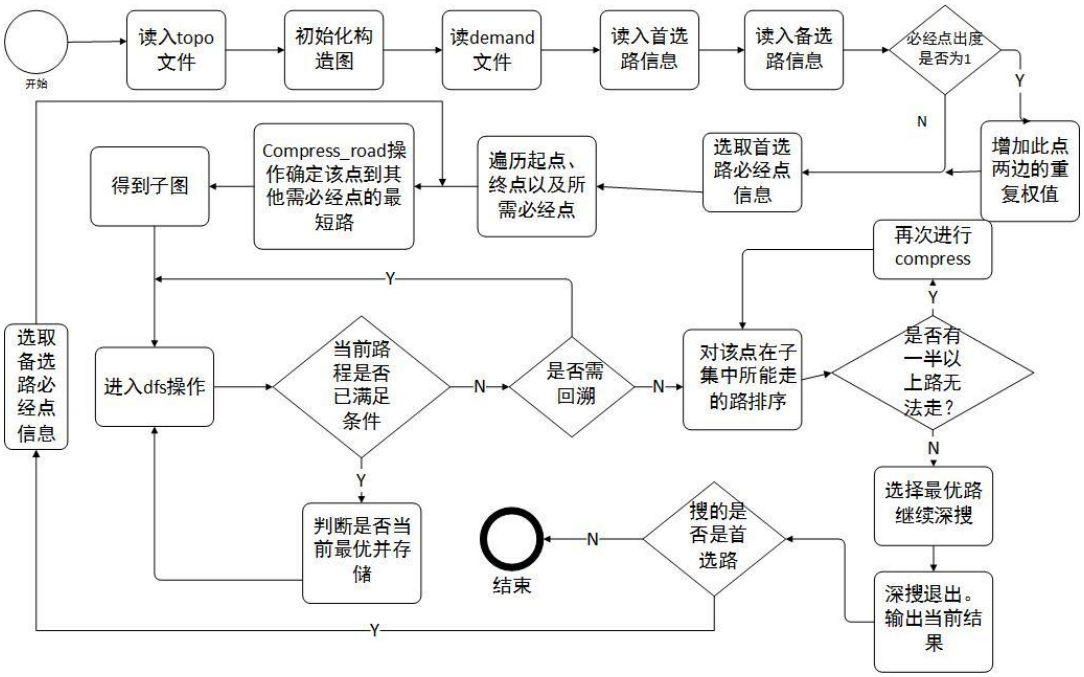

图 4-8　寻路算法的总流程图

4.2 寻路算法演示

下面以图 4-6 所示的拓扑结构图为例，简单演示一次这个寻路算法。

图 4-9 中，起点为 0，终点为 3。首选路的必经点为 1。备选路的必经点为 2。

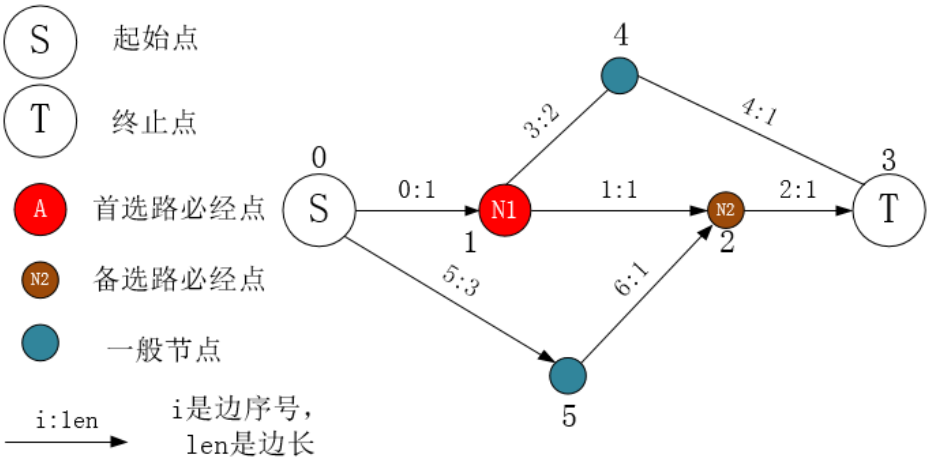

图 4-9　寻路算法的演示图

步骤 1：读取输入，变量和数据结构初始化。

由于备选路的必经点 2 的出度值只有 1，备选路一定会走 i=2 这条路，故首选路应尽量避开 2 点，由处理 4.1.2 可知，在搜索首选路时，1、2、6 这三条边的重复值被初始化为 1。

步骤 2： 首先，搜索首选路。需先对各必经点进行 CompressRoad 操作。

步骤 3：针对起点 0，进行 CompressRoad。0 只能达到 1，且所经过边为 0。

步骤 4：针对必经点 1，进行 CompressRoad。1 能够达到 3，其边为 3、4。之所以没选择 1、2，是因为后者的重复值大于前者（原因见步骤 1）。压缩成如图 4-10 所示：

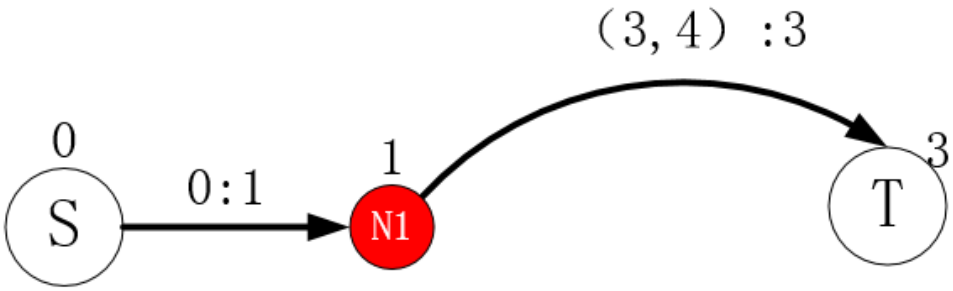

图 4-10 搜首选路时压缩后的图

步骤 5：开始 dfs 搜索。由于中途不需要进行重搜，故可搜得首选路为 0、3、4。

步骤 6：开始搜备选路。备选路的必经点为 0、2、3。重置之前的数据结构，再次进行 CompressRoad。

步骤 7：由于边 1 的重复值为 1，故 0→2 选择的边为 5、6。

步骤 8：2→3 选择的边显然为 2。故备选路的图集被压缩为如图 4-11 所示。

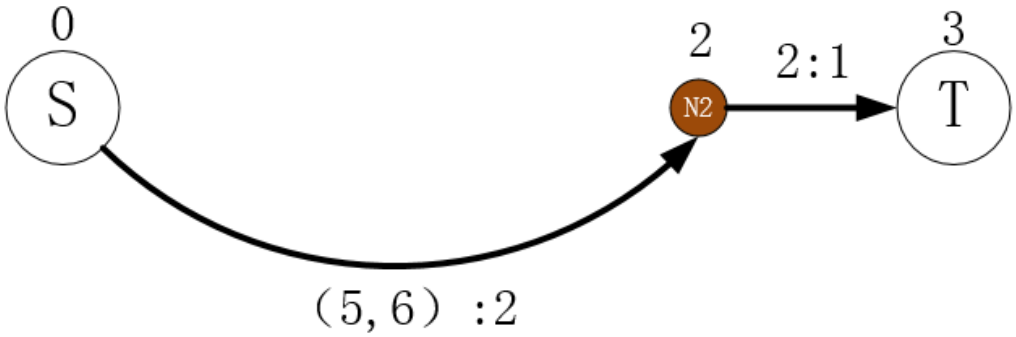

图 4-11 搜备选路压缩后的图

步骤 9：dfs 搜索后，备选路的边为 5、6、2。

步骤 10：搜索结束，得到结果为：

首选路所选择的边为 0，3，4。

备选路所选择的边为 1，5，6。

首选路和备选路的重复值为 0，路总长为 9。

本次演示只简单展现了压缩和 dfs 搜索的过程。关于其中各处优化情况的体现，请参见 5.3.1 中的详细例子以及数据跟踪情况。

4.3 寻路算法的复杂度分析

4.3.1 寻路算法的时间复杂度

一个算法由其控制结构和固有类型操作构成，其算法时间取决于两者的综合效果。为了分析算法的时间效率，通常的做法是，从算法中选取一种基本操作的原操作，以该基本操作重复执行的次数作为算法的时间复杂度[8]。

一般情况下，算法中基本操作重复执行的次数是问题规模n的某个函数$f(n)$，其时间量度记作

$$T(n) = O(f(n)) \tag{4-1}$$

这个量度表示它随着问题规模n的增大，增长率和函数$f(n)$的增长率相同，这称作算法的渐近时间复杂度，简称时间复杂度。

这里我们用这个概念分析一下该课题所提出算法的效率。

设n为节点总数，平均每个节点的出度为e，必经点个数为m。每个必经点的 CompressRoad 操作的最坏时间复杂度为$O(n^2 + ne)$，故总的 CompressRoad 操作最坏时间复杂度为$O(mn^2 + mne)$。由于n的规模限制远大于m，故其约等于$O(mn^2)$。

由于题设条件的特殊性，在短时间内得到一个可行解较为重要。故这里我们分析一下得到一个可行解的时间复杂度。

搜到可行解，相当于从起点搜到最底层，相当于遍历到底部，共经历m层。每一层需要进行排序取优，故每一层消耗$O(mlogm)$的时间。则到达底部所消耗时间为

$$E_t = m \cdot mlogm \tag{4-2}$$

若假设整个过程搜索需要平均经历k次重寻点间路，而一次重寻路需要$O(n^2)$时间。则搜索这个过程所花复杂度为$O(m^2 \log m + kn^2)$。

故总算法的时间复杂度为$O(mn^2 + m^2 \log m + kn^2)$。若考虑到图点$n$远大于$m$，且

k假设为 0（即无需进行重寻点间路），则时间复杂度简化为$O(mn^2)$。

可以看出，在假设的理想情况下，时间复杂度在一个可接受的范围内。但并不保证搜得的路是最优解，只能确保在短时间内搜得一个可行解。

若不考虑搜索时间的限制，以所能得到的最佳结果为标准进行搜索，则需考虑回溯层数x，则 DFS 过程能约能搜得结果次数为

$$R_n = m(m-1)\dots(m-x) = A_m^x \tag{4-3}$$

搜得结果中，取其中最优的作为答案。添加这一考察因素后，搜得最佳结果的总算法时间复杂度为$O(A_m^x * (mn^2 + m^2 \log m + kn^2))$。同理约等于$O(A_m^x mn^2)$。

可以看出，当以求出尽可能好的结果为目标时，x的值设置不能太大。若设$x = m$即将所有可能结果都搜一遍，时间复杂度为$O(m!\, mn^2)$。当m大于 10 时，其时间消耗将非常大。

但当图的复杂度不高时，重寻路次数才会较低。实际上当点的出度不高时，会导致各点之间有足够的关联性，故压缩图时所选择的路径，会相互之间存在冲突。例如路径 1 需要经过 1、3、5 这 3 个点，而路径 2 需要经过 3 这个点，导致路径 2 在路径 1 被选择后，存在走不通的情况。通过 30 次测试，得到k的范围约为 1~5，个别图的k值大于 10。

4.3.1 寻路算法的空间复杂度

类似于算法的时间复杂度，空间复杂度描述了算法所需的存储空间的量度，记作

$$S(n) = O(f(n)) \tag{4-4}$$

若输入数据所占空间只取决于问题本身，和算法无关，则只需要分析除输入和程序之外的额外空间，否则应同时考虑输入本身所需的空间，这和输入数据的表示形式有关。

存储初始图结构需要n^2的空间，压缩时需要m^2条路，但由于m远小于n，故空间复杂度近似为$O(n^2)$。

第 5 章　算法性能测试

5.1　寻路算法的测试方法

本课题将大规模网络拓扑图抽象为 topo.csv 文件，必经路由信息抽象为 demand.csv 文件。仿真程序的编程语言为 C++，编译环境为 VC++6.0。实验环境为 windows7 64 位系统，2.60GHz，i5-3230MCPU，4GB 内存。

以两个.csv 文件（csv 是以逗号为分隔符的文本文件）给出输入数据，一个为图的数据(G)，一个为需要计算的路径信息(s,t,V',V'')。文件每行以换行符（ASCII’\n’即 0x0a）为结尾。

（1）图的数据中，每一行包含如下的信息：

LinkID，SourceID，DestinationID，Cost

其中，LinkID 为该有向边的索引，SourceID 为该有向边的起始顶点的索引，DestinationID 为该有向边的终止顶点的索引，Cost 为该有向边的权重。顶点与有向边的索引均从 0 开始编号(不一定连续，但用例保证索引不重复)，顶点索引范围在[0，2000)，有向边索引范围在[0，40000)。

（2）路径信息中，只有一行如下数据：

DemandID，SourceID，DestinationID，IncludingSet

其中，DemandID 里面第一行为 1，第二行为 2，表示路径索引，1 表示P'，2 表示P''，SourceID 为起始顶点s的索引，DestinationID 为终止顶点t的索引，IncludingSet 表示必须经过的顶点集合V'或V''，其中不同的顶点索引之间用“ | ”分割。

输出文件同样为一个.csv 文件。

（1）如果该测试用例存在满足要求的有向路径P'和P''，则输出两行信息，第一行按P'经过的有向边顺序，依次输出有向边的索引，索引之间用“ | ”分割；第二行按P''经过的有向边顺序依次输出有向边的索引，索引之间用“ | ”分割；

（2）如果该测试用例不同时存在两条满足要求的有向路径P'和P''，则只输出一行信息：NA。

（3）每一行只允许输出最多一条有向路径，以换行符 0x0a 结尾。

写好的算法仿真程序命名为 FindTowShortRoad.exe，该程序需要读入 2 个参数：网

络拓扑图文件 topo.csv 和需求文件 demand.csv。（关于这 2 个文件的含义见第 2 章）。通过 start.bat 脚本启动算法程序，仿真结束后生成 result 文件。该文件中包含以下信息：

（1）首选路的起点，终点，必经点信息。

（2）首选路的总长，搜索所用时间。

（3）首选路经过的所有路径，用边序号 i 来表示。

（4）备选路的起点，终点，必经点信息。

（5）备选路的总长，搜索所用时间。

（6）备选路经过的所有路径，用边序号 i 来表示。

（7）两条路的长度之和，边重叠数，总搜索时间。

5.2 寻路算法的测试结果

5.2.1 实验 1 过程追踪，验证正确性

该实验的测试数据为自主设计，通过跟踪，能够观察该算法的详细求解过程，以及能够体现算法的所有关键之处，便于理解第 4 章的内容。

网络节点数据如图 5-1 所示。

其数据格式含义为：边序号，边起始点，边终止点，边长。

总共 36 条边，30 个节点。

0, 0, 1, 1	12, 12, 13, 10	24, 12, 23, 20
1, 1, 2, 3	13, 13, 14, 1	25, 23, 24, 1
2, 2, 3, 1	14, 14, 15, 1	26, 24, 25, 1
3, 3, 4, 1	15, 15, 16, 1	27, 25, 17, 1
4, 4, 5, 1	16, 16, 17, 1	28, 12, 26, 1
5, 5, 6, 5	17, 1, 18, 1	29, 26, 10, 1
6, 6, 7, 1	18, 18, 19, 1	30, 10, 27, 1
7, 7, 8, 1	19, 19, 5, 1	31, 27, 28, 1
8, 8, 9, 1	20, 5, 20, 1	32, 28, 17, 1
9, 9, 10, 1	21, 20, 21, 1	33, 9, 29, 1
10, 10, 11, 1	22, 21, 22, 1	34, 29, 30, 1
11, 11, 12, 1	23, 22, 12, 1	35, 30, 17, 30

图 5-1　网络节点信息图

需求情况，设起始点为 0。终止点为 17。

首选路的必经点为 5，9，12。

备选路的必经点为 18，21，24。

上述网络节点所形成的网络拓扑结构如图 5-2 所示。

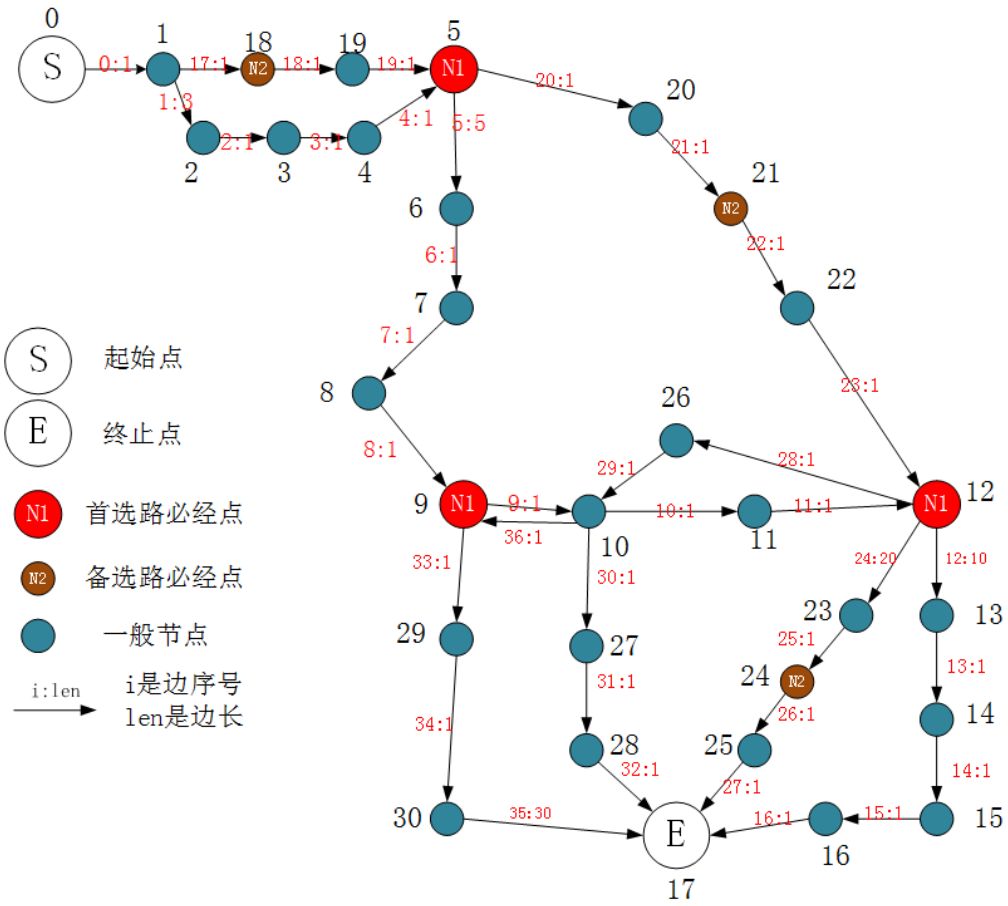

图 5-2　寻路算法测试样例图

为了能够完整展示程序的执行过程，我们针对一些关键之处，添加输出日志。

首先，针对预处理部分，输出当读取备选路必经点时，所进行的增权重操作。输出结果如图 5-3 所示：

```
备选路中发现出度为1的点：18
边18 的重复值加1
边19 的重复值加1
边17 的重复值加1

备选路中发现出度为1的点：21
边22 的重复值加1
边23 的重复值加1
边21 的重复值加1
边20 的重复值加1

备选路中发现出度为1的点：24
边26 的重复值加1
边25 的重复值加1
边24 的重复值加1
```

图 5-3　预处理情况图

对首选路进行压缩，所得到的各路情况：

A	B	C	D
压缩路径	长度	重复权重	路线(所经边的顺序)
0->5	7	0	0|1|2|3|4
5->12	4	4	20|21|22|23
5->9	8	0	5|6|7|8
9->17	4	0	9|30|31|32
9->12	3	0	9|10|11
12->17	5	0	28|29|30|31|32

图 5-4　压缩记录图

dfs 搜索首选路的过程如下：

```
当前为第1层，正处于必经点0
当前为0点,选择5这个点走
当前为第2层，正处于必经点5
当前为5点,选择9这个点走
当前为第3层，正处于必经点9
当前为9点,选择12这个点走
当前为第4层，正处于必经点12
有1个选择已不可走，选择重新进行压缩
当前为12点,选择17这个点走
当前为第5层，正处于必经点17
此为终点且符合条件，需检查是否可更新当前最短路
重复权值为0,总长度为32
当前最短路的重复权值为9999999,总长度为9999999
比当前最短路优，更新！所经过路为：
0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15|16|
回到第4层
回到第3层
当前为9点,选择17这个点走
回到第2层
当前为5点,选择12这个点走
当前为第3层，正处于必经点12
当前为12点,选择9这个点走
当前为第4层，正处于必经点9
有1个选择已不可走，选择重新进行压缩
当前为9点,选择17这个点走
当前为第5层，正处于必经点17
此为终点且符合条件，需检查是否可更新当前最短路
重复权值为4,总长度为46
当前最短路的重复权值为0,总长度为32
回到第4层
回到第3层
当前为12点,选择17这个点走
回到第2层
回到第1层
```

图 5-5　首选路搜索记录图

对备选路进行压缩，所得到的各路情况如图 5-6 所示：

压缩路径	长度	重复权重	路线(所经边的顺序)
0->24	39	14	0|1|2|3|4|5|6|7|8|9|10|11|24|25
0->21	9	7	0|1|2|3|4|20|21
0->18	2	2	0|17
18->24	34	11	18|19|5|6|7|8|9|10|11|24|25
18->17	42	6	18|19|5|6|7|8|33|34|35
18->21	4	4	18|19|20|21
21->24	23	4	22|23|24|25
21->17	7	2	22|23|28|29|30|31|32
24->17	2	1	26|27

图 5-6　备选路压缩结果

dfs 搜索备选路的过程如图 5-7 所示：

```
当前为第1层，正处于必经点0
当前为0点,选择18这个点走
当前为第2层，正处于必经点18
当前为18点,选择21这个点走
当前为第3层，正处于必经点21
当前为21点,选择17这个点走
当前为21点,选择24这个点走
当前为第4层，正处于必经点24
当前为24点,选择17这个点走
当前为第5层，正处于必经点17
此为终点且符合条件，需检查是否可更新当前最短路
重复权值为1,总长度为31
当前最短路的重复权值为9999999,总长度为9999999
比当前最短路优，更新！所经过路为：
0|17|18|19|20|21|22|23|24|25|26|27|
回到第4层
回到第3层
回到第2层
当前为18点,选择17这个点走
当前为18点,选择24这个点走
当前为第3层，正处于必经点24
当前为24点,选择17这个点走
回到第2层
回到第1层
当前为0点,选择21这个点走
当前为第2层，正处于必经点21
当前为21点,选择17这个点走
当前为21点,选择24这个点走
当前为第3层，正处于必经点24
当前为24点,选择17这个点走
回到第2层
回到第1层
当前为0点,选择24这个点走
当前为第2层，正处于必经点24
当前为24点,选择17这个点走
回到第1层
```

图 5-7　备选路搜索过程记录图

最终所得结果如图 5-8 所示：

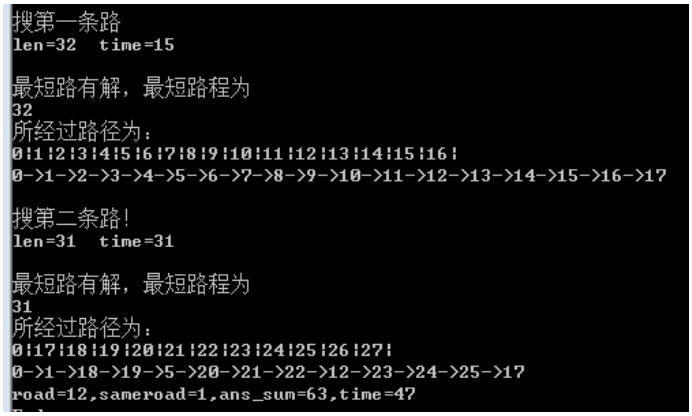

图 5-8　测试所得结果图

通过对图 5-2 中进行人工分析可知，该答案为最佳答案。

5.2.2　实验 2 探索算法在不同规模下的时间效率和结果值

本次实验需提供多种随机生成的测试样例，便于测试算法在各种规模样例下的搜索效果。

测试样例的生成使用自己编写的生成程序 GraphBuilding.exe 进行生成。该程序源码见附件。

起点和终点默认为 0 和 1。

需读取 GraphInfo.txt 文件，来输入图的总节点数 n，必经点数 m1（首选路和备选路必经点数量默认相同），首选及备选路上期望节点 m2，以及各出度范围。

该程序会根据总节点数、各点的出度范围，随机生成一副图文件 topo.csv。同时将必经点数 m1 和路上的期望节点数（路上其他点数）m2 相加，获得一个参考路节点数 M，利用 folyd 插点法[11]，以$O(n^3 \cdot M)$的时间复杂度，在图上寻得一条仅有 M 个节点的“最短路”，选路结束后，在该路上随机选取 m1 个点当作必经点，生成需求文件 demand.csv 和参考答案文件 result.csv。首选路生成结束后，去掉首选路所经过的边，以相同方法，进行备选路的选择，保证注意：该参考答案文件并非正确答案，仅仅是一个可行解，用来和我们的算法进行效果比对所用。

下面给出各测试结果。

表 5-1　50 点规模图的测试

点总数	50	50	50
必经点数	2	5	10
路上其他点数	5	5	5
参考答案长度	52	37	104
测试结果长度	52	36	117
测试结果重边数	0	0	0

由表 5-1 可以看出，在点数较少时，测得的结果能够较为贴近参考答案长度。

表 5-2　200 点规模图的测试

点总数	200	200	200
必经点数	2	5	10
路上其他点数	5	5	5
参考答案长度	52	37	104
测试结果长度	52	36	124
测试结果重边数	0	0	0

表 5-3　500 点规模图的测试

点总数	500	500	500
必经点数	20	40	60
路上其他点数	20	20	20
参考答案长度	156	250	287
测试结果长度	224	392	399
测试结果重边数	0	0	0

表 5-1、表 5-2、表 5-3 的出度范围均为 2~8，每张表在相同的点总数情况下，增加

必经点数的需求。可以看出，随着必经点数的增加，测试结果越差。

下面给出了表 5-4，该测试确定了必经点数和路上其他点数，逐步增加图的总点数。出度范围为 3~10。

表 5-4　确定必经点数的测试情况

点总数	50	100	200	400	600	800	1000
必经点数	10	10	10	10	10	10	10
路上其他点数	10	10	10	10	10	10	10
参考答案长度	115	79	92	94	83	79	84
测试结果长度	114	76	92	83	83	113	84
测试结果重边数	0	0	0	0	0	0	0

从表 5-4 中可以看出，随着图规模的增大，搜索效果并没有变差，说明图的规模对算法结果的影响并不大，这是因为算法的核心在于压缩，图的规模对算法的影响是能够被减小的。

从上面的测试可以看出，影响算法效率的关键点在于必经点m的个数，因为当图的压缩结束后，影响搜索效率和结果的主要因素体现在m上，这与 4.3 节的算法分析结果符合。

最后我们测试极限情况 1200 点，60 必经点的情况，结果如表 5-5 所示。

表 5-5　1200 点规模，60 必经点的测试情况

点总数	必经点数	路上其他点数	参考答案长度	测试结果长度	所耗时间（s）
1200	60	30	307	466	12.02

可以看出，即使是如此大的规模，也能在一个较短的时间内得到一个较为可行解。

第 6 章　总 结

由于自身对大规模图的寻路问题有强烈的兴趣，故结合网络路由的选择背景，选择了这个课题，并进行了一定的研究。本文借鉴并结合了 2 种经典算法和思想，对该问题进行了对应的处理，并加以优化和改进，能够满足要求地完成结果，在文中也给出了算法。

本文主要的研究成果如下：

（1）将一个实际的工程问题抽象为一个图论问题。首先根据提出的实际问题，结合图论的描述，转化为图和寻找最短路的形式，并用合理的语言来描述问题。

（2）探讨了 3 个经典的算法和思想。我们介绍了寻找点对点最短路的 Dijkstra 算法，分析了其流程和复杂度，同时描述了 DFS 深度搜索的运作理念，阐述了贪心思想。

（3）在一定的知识储备上，根据压缩的思想，结合上面提出的算法，提出了“经过指定节点集且重叠边尽可能少的寻路算法”。该算法能够在一定时间内，搜到 2 个符合条件的路。

（4）利用测试程序和样例生成器，完成了对算法的结果检测。

本文所提出的问题是实际的一个工程问题，虽然本质上和 TSP 旅行商问题有些类似，但加入了首选路和备选路的条件限制，并且根据工程要求，必须在尽可能短的时间内得到一个可行解。故其应用场景具有一定的实际意义。

本次课题也根据压缩的思想进行了自己的一些优化和创新。虽然算法的准确度并不算太高，和一些成熟的搜索算法相比差距甚远，但相信在本文的特定背景和问题下，该算法能够给予一些的参考。

在当今人工智能和信息技术愈发火热的时代，对算法问题的需求不断增加，特别是在涉及高性能网络的情况下。想必在不久的将来，云计算以及信息技术能够在算法的支持下，使人类社会进入一个全新的时代。

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] 高尚, 韩斌, 吴小俊, 杨静宇. 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.

[4] 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, 3(4):381-384.

[5] Dorigo M, Maniezzo V, Clolorni A. The ant systern:Optimization by a colony of cooperationg agents[J]. IEEE Trans on Systems, Man and Cybernetics, Part B, 1996, 26(1):29-41.

[6] 9th DIMACS Implementation Challenge. Shortest Paths [OL]. 2006.

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

[8] 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.

[9] 方红, 杨海蓉. 贪婪算法与压缩感知理论[J]. Acta Automatica Sinica, 2011, 37(12):1413-1421.

[10] 黄书力, 胡大裟, 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.

[11] 郝自军, 何尚录. 最短路问题的Floyd算法的若干讨论[J]. 重庆理工大学学报自然科学版, 2008, 22(5):156-159.

[12] 李望超. 神经网络中的最短路径问题[J]. 电子与信息学报, 1996(s1):147-150.

[13] 9th DIMACS Implementation Challenge. Shortest Paths [OL]. 2006.

[14] 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.

[15] 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.

[16] 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.

附 录

这里给出测试图生成器程序，以表示测试样例的合理性。程序的主要代码如下：

```
//储存路信息
void setroad(TopoLine T)
{
    int i = T.i;
    road[i].s = T.s;
    road[i].e = T.e;
    road[i].l = T.l;
    road[i].next = Head[T.s];
    Head[T.s] = i;
}
//用 BFS 建立一个随机图，保证起点到终点一定有路径可达
int graph_build(struct info PicInfo,vector<TopoLine> &vecTopo)
{
    int i=0, OutDegree,Nodenum = PicInfo.AllNodeNum;
    queue<struct TopoLine> tq;
    struct TopoLine T,TT;
    T.s = 0 ;
    tq.push(T);
    while(Nodenum--)
    {
        if(tq.empty())
        {
            do
            {
                T.s = rand()%PicInfo.AllNodeNum;
            }
            while(Head[T.s] != -1);
        }
        else
```

```
        {
            T = tq.front();
            tq.pop();
            if(Head[T.s] != -1)
            {
                Nodenum++;
                continue;
            }
        }
        OutDegree = rand() % (PicInfo.MaxOutDegree - PicInfo.MinOutDegree + 1) +
PicInfo.MinOutDegree;
        while(OutDegree--)
        {
            do
            {
                T.e = rand() % PicInfo.AllNodeNum;
            }
            while(T.s == T.e);
            T.l = rand() % ELMax + 1;
            T.i = i++;
            setroad(T);
            vecTopo.push_back(T);
            if(Head[T.e] == -1)
            {
                TT = T;
                TT.s = T.e;
                tq.push(TT);
            }
        }
    }
    return 0;
}
//存储图
```

```
void getmap(info PicInfo)
{
    int i,j,s,e,diss;
    vector<int> passroad;
    for(i = 0;i<PicInfo.AllNodeNum ;i++)
    {
        for(j = 0;j < PicInfo.AllNodeNum ; j++)
        {
            MapRoad[i][j].dlen=INF;
            MapRoad[i][j].PassRoad.clear();
        }
    }
    for(i = 0;i< PicInfo.AllNodeNum ; i++)
    {
        for(j = Head[i]; j != -1; j = road[j].next)
        {
            e = road[j].e;
            s = road[j].s;
            diss = MapRoad[s][e].dlen;
            if(deflag[j]==0 && road[j].l < diss)
            {
                MapRoad[s][e].dlen = road[j].l;
                MapRoad[s][e].PassRoad.clear();
                MapRoad[s][e].PassRoad.push_back(j);
            }
        }
    }
}
//folyd 插点
void folyd(info PicInfo)
{
     int cnt = PicInfo.AllNodeNum;
    vector<int> passb,passc;
```

```
    int a,b,c,sameflag;
    for(int k=0; k < cnt ;k++)
     for(int i=0; i < cnt ;i++)
     for(int j=0; j < cnt ;j++)
    {
        if(tmp[i][j].dlen > AnsRoad[i][k].dlen + MapRoad[k][j].dlen)
        {
            passb = AnsRoad[i][k].PassRoad;
            passc = MapRoad[k][j].PassRoad;
            sameflag = 0;
            for( b = 0; b < passb.size() && !sameflag; b++)
            {
                for( c = 0; c < passc.size() && !sameflag; c++)
                {
                    if(road[passb[b]].s == road[passc[c]].e)
                    {
                        sameflag = 1;
                    }
                }
            }
            if(sameflag == 1)
                continue;
            passb.insert(passb.end(),passc.begin(),passc.end());
            tmp[i][j].dlen = AnsRoad[i][k].dlen + MapRoad[k][j].dlen;
            tmp[i][j].PassRoad = passb;
        }
    }
}
//复制一个子图
void map_copy(info PicInfo)
{
    int cnt = PicInfo.AllNodeNum;
     for(int i = 0;i < cnt; i++)
```

```
        for(int j = 0;j < cnt; j++)
        {
            AnsRoad[i][j].dlen = tmp[i][j].dlen;
            AnsRoad[i][j].PassRoad = tmp[i][j].PassRoad;
            tmp[i][j].dlen = INF;
        }
}
//找到 N 个点的结果路
void FindNRoad(info PicInfo)
{
    int n = PicInfo.DemandNodeNum + PicInfo.FirstRoadNodeNum;
    for(int i = 0;i < PicInfo.AllNodeNum; i++)
    {
        for(int j=0;j < PicInfo.AllNodeNum;j++)
        {
            if(i==j)
            {
                AnsRoad[i][j].dlen = 0;
            }
            else
            {
                AnsRoad[i][j].dlen = INF;
            }
            AnsRoad[i][j].PassRoad.clear();
            tmp[i][j].dlen = INF;
        }
    }
    while(n--)
    {
        folyd(PicInfo);
        map_copy(PicInfo);
    }
}
```

致　谢

美好的时光总是转瞬即逝。在大学的四年生涯即将过去，回想这些时光，留下的是美好的回忆和奋斗的青春。这段日子我会铭记于心。

首先，我要感谢我的导师王辛刚副教授，在他的细心督促和教导下，我能够明确课题的研究方向，完善课题研究中的不足之处。王老师为人和蔼和善，工作务实认真，良好的学术思维给了我很大的帮助，为我在毕设的工作了提供了很多的思路和纠正，让我受益终生。在此，对辛勤教导我的王老师表示由衷的敬意和谢意。

最后，感谢我的同学，感谢姜晓睿老师，他上的数据结构课程给了我很多的灵感。感谢所有在此次毕设的任务中给予我帮助的人。这段制作毕业设计的生涯让我感到做研究的乐趣，希望今后能有机会再次回到学校，从事算法类的研究工作。