天津工业大学

本 科 生 毕 业 设 计 （ 论 文 ）

学　　号: 1710820215

姓　　名: 田汝浩

专　　业: 信息与计算科学

学　　院: 数学科学学院

指 导 教 师: 谭建国

职　　称: 副教授

完 成 日 期: 2021.3.5

本人声明

我声明，本论文及其研究工作是由本人在导师指导下独立完成的，在完成论文时所利用的一切资料均已在参考文献中列出。

作者： 田汝浩

签字：

时间： 2020 年 02 月

Hopfield 神经网络在城市物流路径规划的应用研究

摘　要

物流企业对我国物流运输乃至经济的发展有着了不可磨灭的贡献，有了物流企业其他实体行业的商品才会有生产和大量销售的可能，运输互动作为物流的一个组成部分，近年来收到了广泛的研究与讨论，其较为热门的解决方法，遗传算法，模拟退火算法，Hopfield 网络也成为了各个领域研究的焦点。本文首先介绍了课题的应用背景与其内容安排，然后初步介绍了解决车辆路径运输问题的数学模型 CVRP，并且结合实例将该模型进行优化，紧接着介绍了近年来学界对于运输问题算法的研究情况，由于本文实例研究中中占主导的算法为 Hopfield 神经网络算法，所以在第三章介绍了神经网络的发展以及在发展过程中产生的各个神经网络的原理以及算法，在第三章最后详细介绍了 Hopfield 神经网络的设计与训练过程。为了在后文中检验 GASA-Hopfield 算法的优越性，我们又介绍了常见的解决路径规划问题的其他优化算法，并且在最后一章中将第四章介绍了算法与 GASA-Hopfield 算法进行了比较，第五章为了增加项目的可扩展性，主要介绍了其项目数据的自动化爬取并将其处理为邻接矩阵的的过程。第六章我们主要介绍了 Hopfield 神经网络与其他算法集成的必要性与可行性，并且介绍了与之集成的算法，K-means，GA,SA，Dijkstra, 然后介绍了集成后各个模块的算法以及实现，最终结合实例，我们将 GASA-Hopfield 算法应用到了天津西青大学城附近的物流运输实例中，根据算法得到结果，我们又从最优路径，平均路径，平均时间三个维度将 GASA-Hopfield 算法与第四章介绍的算法进行对比，得出了 GASA-Hopfield 算法的优越性的依据，以及以后值得优化的方向，为以后求解物流运输问题给出了很好的解决方法。

关键词：物流运输；遗传算法；模拟退火算法；TSP；K-means；Hopfield 网络；Dijkstra

Research on Application of Hopfield Neural Network in Urban Logistics Path Planning

Abstract

Logistics companies have made an indelible contribution to the development of my country’s logistics and transportation and even the economy. Only with logistics companies’ products in other physical industries will it be possible to produce and sell in large quantities. Transportation interaction, as an integral part of logistics, has received extensive attention in recent years. Research and discussion, its more popular solutions, genetic algorithm, simulated annealing algorithm, and Hopfield network have also become the focus of research in various fields. This article first introduces the application background and content arrangement of the subject, and then preliminarily introduces the mathematical model CVRP to solve the vehicle routing problem, and optimizes the model with examples, and then introduces the research status of the transportation problem algorithm in the academic circles in recent years. Since the dominant algorithm in the case study of this article is the Hopfield neural network algorithm, the development of neural networks and the principles and algorithms of each neural network generated during the development process are introduced in chapter three. At the end of chapter three, they are introduced in detail. The design and training process of Hopfield neural network. In order to test the superiority of the GASA-Hopfield algorithm in the following, we introduce other common optimization algorithms for solving path planning problems, and compare the algorithm introduced in Chapter 4 with the GASA-Hopfield algorithm in the last chapter. In order to increase the scalability of the project, Chapter 5 mainly introduces the process of automatic crawling of its project data and processing it into an adjacency matrix. In Chapter 6, we mainly introduced the necessity and feasibility of the integration of Hopfield neural network with other algorithms, and introduced the integrated algorithms, K-means, GA, SA, Dijkstra, and then introduced the algorithms of each module after integration. Realization, and finally combined with examples, we applied the GASA-Hopfield algorithm to the logistics and

transportation examples near the Xiqing University Town in Tianjin. According to the algorithm, we obtained the results. We then combined GASA-Hopfield from the three dimensions of the optimal path, the average path, and the average time. The algorithm is compared with the algorithm introduced in Chapter 4, and the basis for the superiority of the GASA-Hopfield algorithm and the direction worthy of optimization in the future are obtained, and a good solution is given for solving the logistics and transportation problems in the future.

Key words: logistics and transportation; genetic algorithm; simulated annealing algorithm; TSP; K-means; Hopfield network; Dijkstra

目　　录

第一章 绪论 …… 1
一、 引言 …… 1
二、 课题的应用背景 …… 1
三、 本文的内容及结构安排 …… 2
第二章 环境配置相关技术及其理论基础 …… 3
一、 车辆路径运输问题模型 …… 3
(一) 模型一：带有容量约束的车辆路径问题 (CVRP) …… 3
(二) 模型二 …… 4
二、 运输问题算法研究概况 …… 5
第三章 神经网络原理及其算法 …… 7
一、 神经网络的发展与基本概念 …… 7
二、 神经网络的基本特点 …… 15
(一) M-P …… 15
(二) 感知机 …… 15
(三) Hopfield …… 15
(四) BM …… 15
(五) RBM …… 16
(六) BP …… 16
三、 Hopfield 网络 …… 16
(一) Hopfield 网络模型 …… 17
(二) Hopfield 网络的学习算法 …… 20
(三) Hopfield 网络的稳定性 …… 21
四、 网络的应用设计 …… 22
(一) 问题描述 …… 22
(二) 构造能量函数与动态方程 …… 23

(三) 初始化网络 …… 23
(四) 优化计算 …… 23
第四章 解决路径规划问题的其他优化算法 …… 25
一、 暴力穷举搜索 …… 25
二、 剪枝法 …… 26
(一) 剪枝法初步 …… 26
(二) 启发式算法的剪枝法 …… 27
三、 智能算法简介 …… 28
(一) 粒子群算法 …… 28
(二) 模拟退火算法 …… 30
(三) 遗传算法 …… 33
(四) 蚁群算法 …… 35
四、 动态规划 …… 36
第五章 数据预处理 …… 38
一、 数据爬取 …… 38
(一) 爬取 osm 数据 …… 38
(二) 转化成 shp 数据 …… 40
二、 图数据结构的构建 …… 45
(一) 读取 line 数据 …… 45
(二) 求交点 …… 45
第六章 集成 Hopfield 网络与其他优化算法求解运输问题 …… 49
一、 集成的必要性与可行性 …… 49
二、 集成的方法 …… 49
(一) 单源最短路 …… 49
(二) 主程序 …… 50
(三) 遗传算法子程序 …… 51
(四) 模拟退火算法子程序 …… 51
(五) k-means 算法 …… 51

三、 针对运输问题集成 Hopfield 网络与其他优化算法进行求解 …… 53
(一) 算法实现 …… 53
(二) 实例求解 …… 55
总结 …… 60
致谢 …… 61
参考文献 …… 62
附录 A 常见问题 …… 64
附录 B 联系我们 …… 65

第一章　绪论

一、　引言

运输问题，一版被描述为寻找某一种最优的运输方案使得被运输品从始发地运输到多个目的地, 使得运输方利益最大化。我国的物流运输现状，近年来有着规模扩大, 运量增加的趋势. 截止到 2021 年, 我国近五年的社会物流总额年均增长率超 6.5%，物流业总收入超 10 万亿元. 但现阶段我国物流还存在着效率低，兼容性差，建设滞后，机制障碍的问题。所以在现阶段在面对大量的公路，铁路，海运等物流运输要求时，可以在时间和仓储方面消耗最少的资源无疑会更有优势，这也对现有的物流路径规划算法有了更高的要求。

二、　课题的应用背景

近几年来我国物流企业不断崛起，整体呈现出规模巨大，科技水平提升，物流发展格局不断优化的特点，物流企业对我国物流运输乃至经济的发展有着了不可磨灭的贡献，有了物流企业其他实体行业的商品才会有生产和大量销售的可能而商品生产出来的目的就是为了被消费者所消耗，而大学生作为聚集且消费能力强的群体，导致大学校园以及周边的外卖快递的物流运输十分的错综复杂。但本文以天津西青大学城为例，研究物流运输的路径最优问题。面积近 90 平方公里，入驻师生约 40 万人。入驻高校有天津工业大学、天津师范大学、天津理工大学、天津师范大学津沽学院、天津大学软件学院、天津公安警官职业学院、天津农学院、天津城建大学、天津商业大学宝德学院、天津教育招生考试院等高等教育机构、公安部天津消防研究所等科研机构、天津团泊体育中心等体育远动场所发展起来的郊区新城，近年来物流运输请求的逐年增加，如何让商品流转于各个地方最终到达消费者手中且能够在消费者可接受的时间内送达成为了平台和快递员与外卖员常常思考的问题。但在后文的讨论中我们就会发现物流运输在大学校园的应用其本质就是解决 TSP 问题。为了增加代码的复用性，本文所用的代码可以通过经纬度自动搜集任意地区的地图矢量数据，使用本文的方法对路径进行优化。

三、 本文的内容及结构安排

本文一共分为六章。

第一章绪论。主要介绍了本文的研究背景与相关领域的介绍。

第二章相关技术及其理论基础。介绍了运输问题的理论模型以及解决常见运输问题的研究算法。

第三章神经网络原理及其算法。介绍了神经网络的基本概念，以及详细介绍了 hopfield 神经网络的模型，工作方式以及稳定性。

第四章解决路径规划问题的其他优化算法。介绍了除神经网络外其余已经成熟的解决路径规划的算法。

第五章数据爬取介绍了本文例子如何从网站爬取并且对数据进行转化清洗最终转化为图的数据结构。

第六章集成 Hopfield 网络与其他优化算法算法求解运输问题。实现本文的例子，并且将集成的 Hopfield 路径规划算法与其余算法在时间空间性能稳定性上进行比较。

第二章　环境配置相关技术及其理论基础

一、　车辆路径运输问题模型

Dantzig 和 Ramser 于 1959 年首次提出了车辆路径规划问题 (VRP)，这个问题是指不定量的顾客，每个顾客有不同的需求，一个或多个物流中心向客户提供所需商品，并且由一个或多个配送员分配顾客的货物，并寻找合适的配送路径，在满足客户需求的大前提下，能够完成例如路径最短，时间花费最少等成本消耗最小的目的。

(一)　模型一：带有容量约束的车辆路径问题 (CVRP)

记顾客集合 $M = \{1, 2, \cdots, n\}$

记顶点集合 $V = M \cup \{0, n+1\}$

记车辆数量 D

记车辆总容量 K

记顾客 j 的需求 R_j

记路径成本 x_{ij}

目标函数如下：

$$min \quad f = \sum_{i=0}^{n} +1 \sum_{j=0}^{n} +1 x_{ij}\omega_{ij} \tag{2.1}$$

$$
st.\begin{cases}
\sum_{\substack{j=1\\ j\neq i}}^{n+1}\omega_{ij}=1 \quad \forall i\in M & (2.2)\\
\sum_{\substack{i=0\\ k\neq i}}^{n+1}\omega_{ik}=\sum_{\substack{j=1\\ k\neq i}}^{n}+1\omega_{kj} \quad \forall k\in M & (2.3)\\
\sum_{j=1}^{n}\omega_{0j}\Subset D & (2.4)\\
y_i+\omega_{ij}q_j-k(1-\omega_{ij})\leq y_i \quad \forall i,j\in V & (2.5)\\
q_j\leq y_i\leq Q \quad \forall i\in N & (2.6)\\
x_{ij}\in\{0,1\} \quad \forall i,j\in N & (2.7)
\end{cases}
$$

上式中 (2.1) 是目标函数，代表合法路径的成本最小化，(2.2) 代表每个点除起点外都要出一次，(2.3) 代表流量约束，指每个点出一次必须进一次，(2.4) 代表起点出弧数小于车辆数量，而 (2.5) 代表合法路径不存在回路，(2.6) 表示需求不超过车的容量。(2.7) 表示 ω_{ij} 满足 0，1 约束。

对于本文例子的应用，经过调查走访发现大学城南门各饭店小吃街的各个商户的外卖商品样式质量相似，所以各个顾客的需求可近似为 1，其店铺规模普遍偏小，每家商户通常配备 1 名配送员，一名配送员通常服务 1 家门店所以 D = 1，K = 1, 故模型可以进一步简化为模型二

(二) 模型二

记赋权图 G=(V,E)，V 为顶点集 $V=\{1,2,\cdots,n\}$，E 为边集，各顶点间的距离 d_{ij} 已知 n 为地点数量。车辆路径运输模型如下：

$$
min \quad f=\sum_{i=1}^{n}\sum_{j=1}^{n}\omega_{ij}x_{ij} \tag{2.8}
$$

$$
st.\begin{cases}
\sum\limits_{\substack{i=1\\1\le j\le n}}^{n}\omega_{ij}=1 \quad \forall i\in V & (2.9)\\
\sum\limits_{\substack{j=1\\1\le i\le n}}^{n}\omega_{ij}=1 \quad \forall j\in V & (2.10)\\
\sum\limits_{i=1}^{n}\omega_{ij}=\sum\limits_{k=1}^{n}\omega_{jk} \quad \forall j\in V & (2.11)\\
\sum\limits_{i\in S}\sum\limits_{j\in\bar{S}}\omega_{ij}\ge 1 \quad \forall S\Subset V, S\neq\varnothing & (2.12)\\
\omega_{ij}\in\{0,1\} \quad \forall i,j\in V & (2.13)\\
\omega_{ii}\neq 1 \quad \forall i\in V & (2.14)
\end{cases}
$$

上式中 ω_{ij} 代表最终路径中是否存在从 i 到 j 的通路，x_{ij} 代表从 i 到 j 所需代价。(2.8) 是目标函数，即通过所有节点后代价最小。目标函数下面为各个约束条件，其中 (2.9) 说明任意节点的进度为 1, (2.10) 说明任意节点的出度为 1，(2.9) 和 (2.10) 规定合法路径中不存在分叉，(2.11) 是流量平衡约束，即合法路径流量从一个节点到另外一个节点不会存在断流情况。(2.12) 是消除子回路约束，其中 n 个地点被任意分为 S 与 $\bar{S}$，合法路径中应满足以存在于 S 中的 i 点为起点，以存在于 $\bar{S}$ 中的 j 点为终点的情况。(2.13) 说明 ω_{ij} 满足 0，1 约束。(2.14) 说明路径任意节点并不存在一条自己到自己的通路。

二、 运输问题算法研究概况

目前对于路径规划问题的算法分为四部分

(1) 精确式算法:

其精确式算法主要分为三部分即动态规划法，线性规划法，与分支定界法，但此三种方法的运行时间会随着问题的规模指数上升，对于大型问题普遍求解效率偏低，经过近几年算法的发展，学者们对其进行了改进，王晓琨利用混合启发式快速算法与整数规划解决了电车充电需求问题，张鹏乐提出了一种求解大规模 VCVRP 问题的快速动态规划模型即 DPM-MST 模型，并且改进后的算法求解质量受节点规模波动较小。

(2) 启发式算法:

启发式算法相对于最优化算法而被定义，通常是一个基于直观或经验构造

的算法，它可以在规定时间内给出一个问题的可行解，但求得的解与最优解的偏差无法被估计，可行解的区间也无法被确切估计。其在 VRP 的应用主要由三类算法构成，即插入检测法，节约法，与改进节约法。启发式算法能够解决大规模 VRP 问题，尽管往往求出的并非是最优解。此类算法的应用研究在十余年前较为流行，近几年研究文献逐步下降，例如孔媛设计了新的评价因子即最小评价因子并且结合了 VSPA(接送顾客到机场的车辆调度问题) 问题的特点，基于顺序插入的方法构造路径。李兵采用了重新优化法方法即将车辆的出发点看作任务点使得问题可以看作可以使用静态问题算法求解的普通车辆路径问题。但此阶段的启发式算法对于问题最优解的寻找作用较为薄弱。

(3) 元启发式算法:

元启发式算法是启发式算法的改进，它是随机算法与局部搜索算法相结合的产物，相对于启发式算法，它求得的可行解质量有了较大的提升，它由遗传算法，蚁群算法，模拟退火算法，粒子群算法，禁忌搜索算法与混合算法等六类算法组成。元启发式算法也是近几年研究成果最为丰富的一类算法。穆东等基于并行模型退火算法求解时间依赖型车辆路径问题，罗勇等针对遗传算法的选择交叉变异步骤，提出了基于序的选择算子、基于最小代价树的交叉算子和基于随机点长度控制的变异算子。其改进后的遗传算法收敛速度与全局搜索能力有了较大的提升。

(4) 神经网络: 除了以上常用的几种元启发式算法以外, 还有一些文献也提到了神经网络算法, 学者们将人工智能的方法应用到 VRP 问题中, 取得了较好的效果. 路径规划更加合理, 速度也更快. 随着环境的动态性和信息的未知性, 通过神经网络学习的方式, 对未知环境进行训练式探索, 是一种较好的求解方法。而本文的研究方法主要为神经网络，下面对该类方法进行较为详细的介绍。

第三章　神经网络原理及其算法

一、　神经网络的发展与基本概念

神经学说为西班牙解刨学家 Cajal 于十九世纪末创造并建立，20 世纪 40 年代美国心理学家 McCulloch 以及数学家 Pitts 在论文《神经活动中所蕴涵思想的逻辑活动》中提出了 M-P 模型。M-P 模型也为后续神经网络的研究奠定了基础，1949 年 Hebb 在《行为的组织》中对各个神经元的规则进行了分析改进，Hebb 规则被以此提出。其学习规则如下所示：

$$\omega_{ij}(t+1) = \omega_{ij}(t) + \alpha y_i(t) y_j(t) \tag{3.1}$$

其中 $\omega_{ij}(t)$ 表示在时间为 t 时，i 与 j 两个神经元的连接强度，y_i 表示 j 神经元输出的值，y_j 表示 j 神经元输出的值继属于无监督学习的 Hebb 规则之后，属于有监督学习范畴的 Delta 被创立，Delta 对权值不断迭代改进使其神经网络的训练精度提高，其改进公式如下：

$$\omega_{ij}(t+1) = \omega_{ij}(t) + \alpha(d_i - y_i) x_j(t) \tag{3.2}$$

其中，α 表示神经网络的学习速度，$x_i(t)$ 指第 i 个神经元在时间为 t 时的值，一般用 0，1 来表示神经元抑制或激活的状态，y_i 表示第 i 个神经元的实际输出，而 d_i 表示第 i 个神经元的期望输出。值得注意的是前面介绍的两种学习规则都是对单一神经元进行操作。虽然单一神经元的效能少之又少，但是两位学者的工作为后来神经网络的发展奠定了基础。20 世纪 60 年代，Rosenblatt 等将神经网络的学习功能首次应用于模式识别，这是神经网络研究的一个里程碑式的成果，该算法的损失函数基于误分类，并且对损失函数使用梯度下降算法进行极小化求解。模型如下所示：

$$y = f(x) = sign(\omega \cdot x + b) \tag{3.3}$$

y 的空间为 $y = \{-1, +1\}$，ω 表示权值，它与 x 均为向量，b 解释为神经元的偏置值；$\omega \cdot x$ 即二者的内积；而 *sign* 函数的定义如下：

$$sign(x) = \begin{cases} +1, & x > 0 \\ -1, & x < 0 \end{cases} \tag{3.4}$$

给定数据集 $T = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$，设超平面误分类点集 N，损失函数如下定义：

$$L(\omega, b) = -\sum_{x_i \in N} y_i \quad (\omega \cdot x_i + b) \tag{3.5}$$

而感知机算法目的即为求解损失函数的极小值即：

$$\min L(\omega, b) = -\sum_{x_i \in N} y_i \quad (\omega \cdot x_i + b) \tag{3.6}$$

通过使用梯度下降法来求解极小值，具体算法步骤如下：

Step 1: 求解损失函数对 ω 与 b 的偏导数即

$$\nabla_\omega L(\omega, b) = -\sum_{x_i \in N} x_i \cdot y_i \tag{3.7}$$

$$\nabla_b L(\omega, b) = -\sum_{x_i \in N} y_i \tag{3.8}$$

Step 2: 对某个误分类点不妨设为 (x_i, y_i), 以此来更新 ω 与 b

$$\omega^{new} = \omega^{old} + \eta y_i x_i \tag{3.9}$$

$$b^{new} = b^{old} + \eta y_i \tag{3.10}$$

其中 η 代表步长，从上面的感知机原始形式还存在感知机的另一种形式即对偶形式我们假设某一样本点设为 (x_i, y_i) 在一共更新了 n_i 次，所以从 (3.9) 与 (3.10)

两式中我们可以将 ω 与 b 的更新公式改进为：

$$\omega = \sum_{i=1}^{N} n_i \eta y_i x_i \tag{3.11}$$

$$b = \sum_{i=1}^{N} n_i \eta y_i \tag{3.12}$$

显然，n_i 的值越大，越意味着该样本点容易被误分类，该点是离超平面很近的点，随着超平面的不断更新，该点的种类被不断改变，在 SVM 中，该样本点很可能是样本点。所以感知机的模型更新为：

$$f(x) = sign(\omega \cdot x + b) = sign(\sum_{j=1}^{N} n_j \eta y_i x_j \cdot x + \sum_{j=1}^{N} n_j \eta y_j) \tag{3.13}$$

此时我们发现训练的目标是为了求解 n_i，而不是 ω 与 b，$i = \{1, 2, \cdots, N\}$。所以感知机对偶形式的训练过程为：

$Step$ 1: 初始化 $\forall n_i = 0$。

$Step$ 2: 选取数据 (x_i, y_i)。

$Step$ 3: 若 $y_i(\sum_{j=1}^{N} n_j \eta y_j x_j \cdot x_i + \sum_{j=1}^{N} n_j \eta y_j) \leq 0 \quad n_i \leftarrow n_i + 1$

$Step$ 4：转 Step2，如果没有误分类数据则完成并退出训练。

感知机的对偶与原始形式本质目的并无区别，但感知机的对偶形式可以事先算好 Gram 矩阵，即所有的内积，这样大大加快了感知机的训练速度。

1969 年，从数学角度上 Papert 和 Minsky 证明了单层神经网络的局限性，尤其即使是面对简单的逻辑问题是也很难解决。但现实中很多复杂函数是无法通过单层神经网络进行拟合的。所以之后的很长一段时间时间美国与苏联对神经网络的工作的资助近乎停滞。1982 年加州理工学院的 Hopfield 提出了离散与连续的神经网络，被称为 Hopfield 神经网络。该神经网络采用全互联型网络对 NP 难度的问题旅行商（Travelling Salesman Problem,TSP）求解。Hopfield 神经网络的提出对神经网络的研究打入了催化剂，神经网络的研究再次进入了快速发展时期。

1983 年，Sejnowski 和 Hinton 首次提出了“隐单元”的概念。玻尔兹曼机（Boltzmann Machine，BM）也以此被设计出来，BM 是一种全连接随即神经元构成的反馈神经网络，由统计概率规则决定每个神经元的状态即激发或未激发。

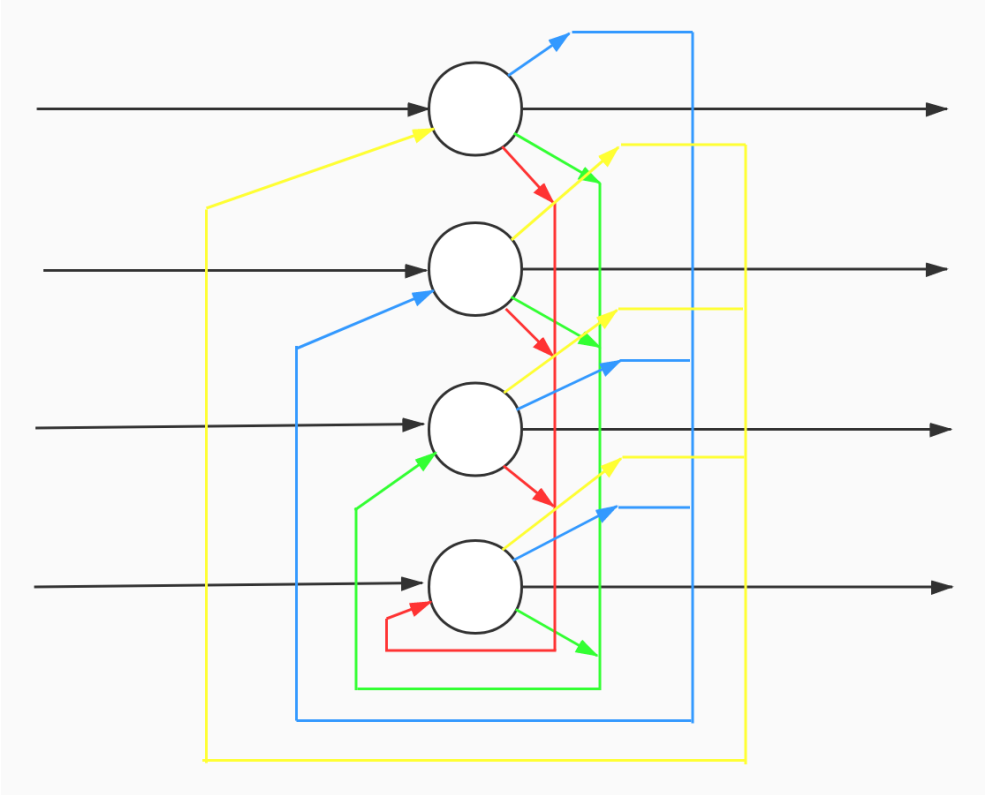

图 3-1　Hopfield 网络示意图

BM 可以从数据中寻找规律属于无监督学习范畴，但缺点是学习时间普遍较长，并且计算 BM 的分布存在着困难。因此限制玻尔兹曼机（Restricted Boltzmann Machine, RBM）被创立，他是 BM 的改进型相对于 BM 的特点是层间全连接，层内无连接，并且求 RBM 的分布可以使用 Gibbs 采样法。RBM 现被应用于语音识别，协同过滤推荐，网络故障诊断，分类，特征提取，图像识别等领域，RBM 阶段的神经网络已从单层进化成了双层结构，隐含层使得网络的数据表达能力大大加强。

RBM 阶段解决了网络灵活性问题，但参数训练是制约神经网络发展的一个因素.1974 年 Werbos 首次提出了 BP（Back Propagation）算法，证明了一个连续函数在任意区间内都可以用一个隐含层的 BP 网络来逼近，该算法旨在解决神经网络的参数训练问题，之后 Rumelhart 与 McCelland 在 1986 年对反向传播算法进行了分析。其网络结构如下所示：BP 除了隐含层，输入输出层的节点个数固

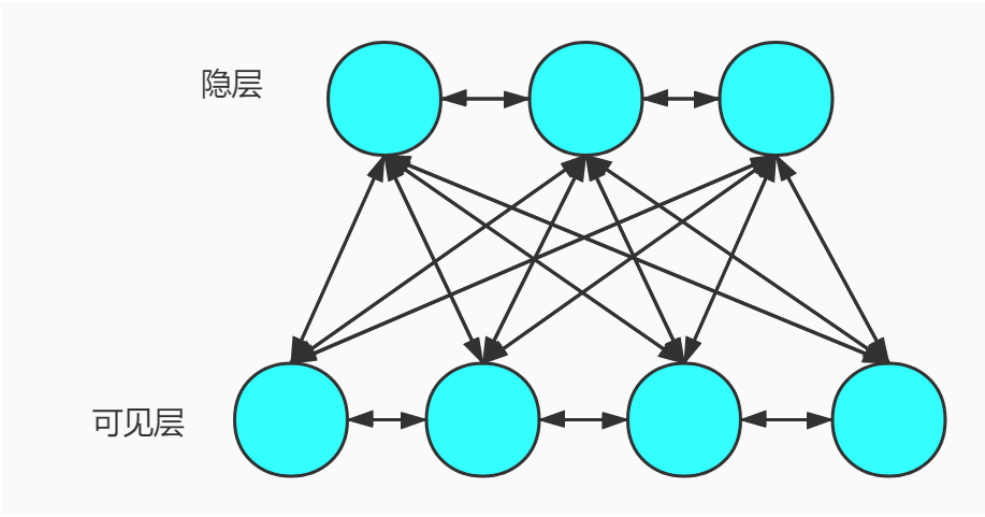

图 3-2　玻尔兹曼机结构示意图

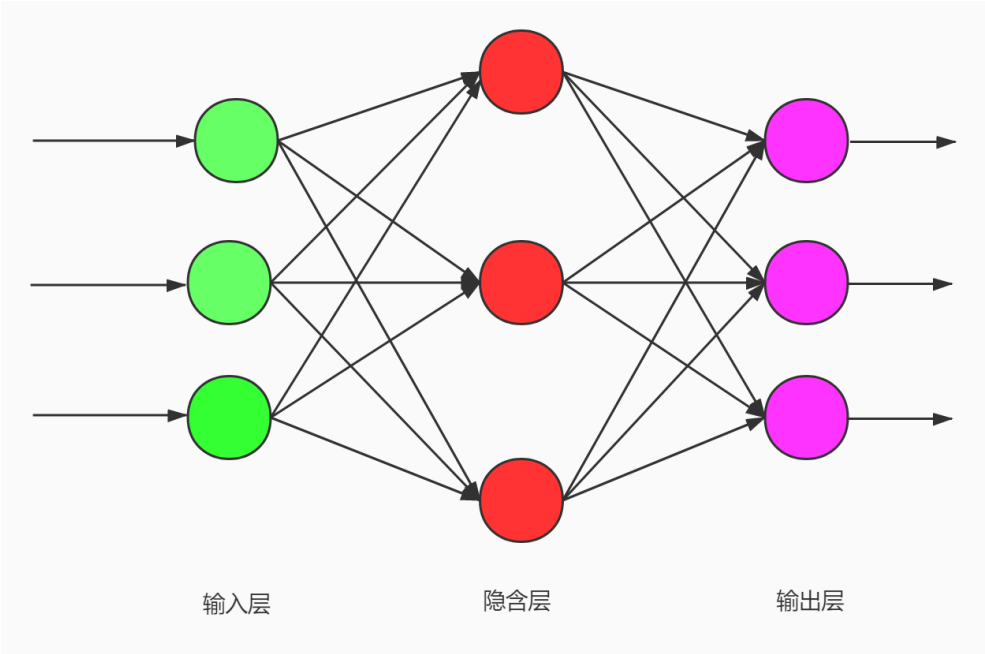

图 3-3　BP 神经网络结构示意图

定，而隐含层节点的选取可以使用经验公式确定即：

$$h = \sqrt{m+n} + a \tag{3.14}$$

隐含层节点数目为 h，输入层节点数目记为 m，输出节点数目记为 n，a 记为调节常数，$1 \le a \le 10$。

BP 正向传播过程：

ω_{ij} 代表第 i 个节点与第 j 个节点的权值，b_j 表示第 j 个节点的阈值。x_j 表示第 j 个节点的输出值。具体算法如下：

$$S_j = \sum_{i=0}^{m-1} \omega_{ij} x_i + b_j \tag{3.15}$$

$$x_j = f(S_j) \tag{3.16}$$

f 代表激活函数，激活函数定义了该节点输入对应的输出，BP 中选取线性或 S 型函数作为激活函数。BP 反向传播过程：设输出层输出为 d_j，则误差函数如下定义：

$$E(\omega, b) = \frac{1}{2} \sum_{j=1}^{n-1} (d_j - y_j)^2 \tag{3.17}$$

Widrow-Hoff 学习算法选取误差平方和下降最快的方向作为迭代方向，所以任意一个参数变化大小为：

$$\triangle\theta = -\eta \frac{\partial E_k}{\partial \theta} \tag{3.18}$$

其中，θ 代表 BP 中的任意一个参数，下面我们根据 (3.18) 来求解各个参数的更新公式：

假设激活函数为：

$$f(x) = \frac{A}{1 + e^{-\frac{C}{B}}} \tag{3.19}$$

$$\tag{3.20}$$

对激活函数求导：

$$f^{'}(x) = \frac{Ae^{-\frac{C}{B}}}{B(1 + e^{-\frac{C}{B}})^2} \tag{3.21}$$

$$= \frac{1}{AB} \cdot \frac{A}{1 + e^{-\frac{C}{B}}} \cdot (A - \frac{A}{1 + e^{-\frac{C}{B}}}) \tag{3.22}$$

$$= \frac{f(x)[A - f(x)]}{AB} \tag{3.23}$$

则对于 ω_{ij}：

$$\frac{\partial E(\omega, b)}{\partial \omega_{ij}} = \frac{1}{\partial \omega_{ij}} \cdot \frac{1}{2} \sum_{j=0}^{n-1} (d_j - y_j)^2 \tag{3.24}$$

$$= (d_j - y_j) \cdot \frac{\partial d_j}{\partial \omega_{ij}} \tag{3.25}$$

$$= (d_j - y_j) \cdot f^{'}(S_j) \cdot \frac{\partial S_j}{\partial \omega_{ij}} \tag{3.26}$$

$$= (d_j - y_j) \cdot \frac{f(S_j)[A - f(S_j)]}{AB} \cdot \frac{\partial S_j}{\partial \omega_{ij}} \tag{3.27}$$

$$= (d_j - y_j) \cdot \frac{f(S_j)[A - f(S_j)]}{AB} \cdot x_i \tag{3.28}$$

$$= \delta \cdot x_i \tag{3.29}$$

其中

$$\delta = (d_j - y_j) \cdot \frac{f(S_j)[A - f(S_j)]}{AB} \tag{3.30}$$

对于 b_j 也有：

$$\frac{\partial E(\omega, b)}{\partial b_j} = \delta_{ij} \tag{3.31}$$

根据上述学习规则，每次迭代，神经元连接的权值进行改变，进而系统输出的误差逐渐减小，这个学习规则被称为纠错学习规则。上面是针对隐含层与输出层之间权值与输出层阈值的迭代，下面介绍针对输入层和隐含层之间的权值与隐含层的阈值调整，下面设 ω_{ki} 代表隐含层 i 节点与输入层 k 节点的权值：

$$\frac{\partial E(\omega,b)}{\partial \omega_{ki}} = \frac{1}{\partial \omega_{ki}} \cdot \frac{1}{2}\sum_{j=0}^{n-1}(d_j - y_j)^2 \tag{3.32}$$

$$= \sum_{j=0}^{n-1}(d_j - y_j)\cdot f'(S_j)\cdot \frac{\partial S_j}{\partial \omega_{ki}} \tag{3.33}$$

$$= \sum_{j=0}^{n-1}(d_j - y_j)\cdot f'(S_j)\cdot \frac{\partial S_j}{\partial x_i}\cdot \frac{\partial x_i}{\partial S_i}\cdot \frac{\partial S_i}{\partial \omega_{ki}} \tag{3.34}$$

$$= \sum_{j=0}^{n-1}\delta_{ij}\cdot \omega_{ij}\cdot \frac{f(S_i)[A - f(S_i)]}{AB}\cdot x_k \tag{3.35}$$

$$= x_k \cdot \sum_{j=0}^{n-1}\delta_{ij}\cdot \omega_{ij}\cdot \frac{f(S_i)[A - f(S_i)]}{AB} \tag{3.36}$$

$$= \delta \cdot x_k \tag{3.37}$$

其中：

$$\delta = \sum_{j=0}^{n-1}\delta_{ij}\cdot \omega_{ij}\cdot \frac{f(S_i)[A - f(S_i)]}{AB} \tag{3.38}$$

所以根据上述公式，并结合梯度下降算法，隐含层与输出层之间的权值和阈值修改为：

$$\omega_{ij} = \omega_{ij} - \eta_1 \cdot \frac{\partial E(\omega,b)}{\partial \omega_{ij}} = \omega_{ij} - \eta_1 \cdot \delta_{ij}\cdot x_i \tag{3.39}$$

$$b_j = b_j - \eta_2 \cdot \frac{\partial E(\omega,b)}{\partial b_j} = b_j - \eta_2 \cdot \delta_{ij} \tag{3.40}$$

同上，输入层与隐含层之间的权值和阈值修改为：

$$\omega_{ki} = \omega_{ki} - \eta_1 \cdot \frac{\partial E(\omega, b)}{\partial \omega_{ki}} = \omega_{ki} - \eta_1 \cdot \delta_{ki} \cdot x_k \tag{3.41}$$

$$b_i = b_i - \eta_2 \cdot \frac{\partial E(\omega, b)}{\partial b_i} = b_i - \eta_2 \cdot \delta_{ki} \tag{3.42}$$

二、 神经网络的基本特点

(一) M-P

1. 每个神经元都是一个多输入单输出的信息处理单元；
2. 神经元输入分兴奋性输入和抑制性输入两种类型；
3. 神经元具有空间整合特性和阈值特性；
4. 神经元输入与输出间有固定的时滞，主要取决于突触延搁；
5. 忽略时间整合作用和不应期；
6. 神经元本身是非时变的，即其突触时延和突触强度均为常数。

(二) 感知机

1. 首个神经网络用来做模式识别。
2. 二分类线性判别的模型。
3. 梯度下降法求解经验函数的最小值来更新参数。
4. 无法解决简单的“异或”问题。

(三) Hopfield

1. 分为连续，离散两种模型。
2. 全互联神经网络，可对 TSP 问题进行求解。
3. 有利于工程实践，利用运算放大器，电容等组成模拟电路来模型神经网络。
4. 实现联想记忆功能。指的是即使输入的数据被污染网络仍然会通过联想记忆功能来求出完整结果。

(四) BM

1. 以此首次提出了“隐单元”的概念。

2. 随即神经元全连接神经网络。

3. 输出由概率统计规则规定。

4. 强大的无监督学习能力。

5. 难以求出 BM 表示的分布。

6. 双层网络。

(五) RBM

1. 层内不连，层间全相联。

2. 通过 Gibbs 采样得到 RBM 的分布的随机样本。

3. 双层网络。

4. 无法拟合复杂函数。

(六) BP

1. 为多层神经网络训练提供了可靠建议。

2. 三层拓扑结构，输入输出层，隐含层。

3. 反向传播使得网络误差降低。

4. 随着反向传播层数的增加其误差修正会逐层衰减。

5. 复杂的网络优化问题时间效率不高。

由于本项目的实现主要是基于 Hopfield，所以下一节我们将详细介绍 Hopfield 网络模型。

三、 Hopfield 网络

从上节我们可以知道前馈神经网络自身会产生一些缺点，他的学习主要采用误差修正法，导致前馈神经网络收敛速度较慢，计算过程较慢。而反馈型神经网络主要采用 Hebb 学习，所以收敛速度较快。而本节主要介绍的是一种典型的反馈神经网络，Hopfield 神经网络，该网络在 1982 年由美国物理学家 Hopfield 创立，并由 1984 年实现了该网络的硬件电路设计，Hopfield 的论文中描述了该网络可以应用于组合优化问题，比如组合优化的经典问题 TSP 问题，所以 Hopfield 与 Tank 使用 Hopfield 网络求解了 30 个城市的 TSP 问题，1987 年贝尔实验室在 Hopfield 的基础上研制出了神经网络的芯片，Hopfield 首次在神经网络领域引入了“能量函数”的概念，并且证明了网络的稳定性，Hopfield 网络是神经网络领

域一个里程碑式的发现。

(一) Hopfield 网络模型

Hopfield 模型被划分为连续性（CHNN）与离散型（DHNN），本项目用的是连续性神经网络，其中连续型可以去解决组合优化问题，离散型常用于处理联想记忆，如文字识别等。

(1) 离散型:

离散 Hopfield（Discrete Hopfield Network, DHNN）的结构是单层的反馈网络。Hopfield 在论文中最早提出的是二值神经网络，即神经元的输出只有-1，1 两个值，所以又被称为离散型 Hopfield 网络，在输出的 0，1 中 1 表示激活，-1 表示抑制。在图 (3-4) 中，神经元一共在 0 层和 1 层中，其中 0 层没有计算的功能即他没有

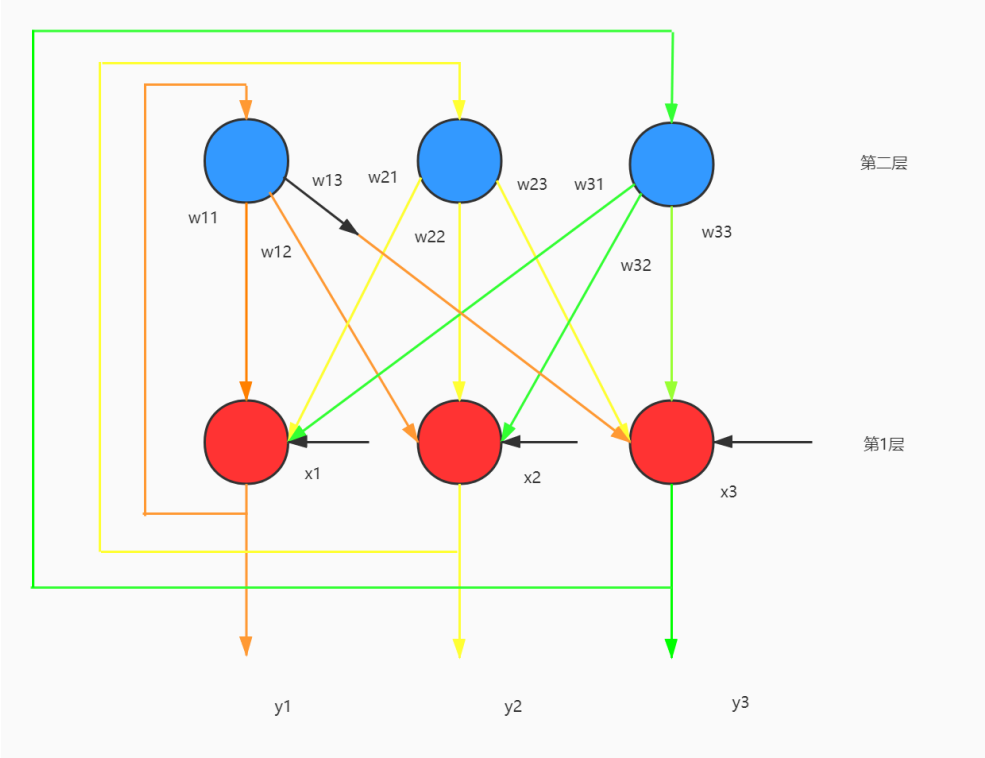

图 3-4　离散型 Hopfield 神经网络

履行实际神经元的职能，所以它仅仅作为网络的输入看待；而第一层的神经元有计算功能，具体计算步骤为每项处理后累加，其中每项为某个输入信息与其对应的系数，该累加和在带入非线性函数 f 计算后作为最终结果输出，其中 f 为一个简单的判断函数，该函数存在一个阈值 θ, 如果累加和大于该阈值则神经元作为

激发态输出，输出结果为 1，如果累加和小于阈值则神经元作为抑制输出，输出结果为-1。下面列出神经元的计算公式。

$$u_i = \sum_i \omega_{ij} y_i + x_i \tag{3.43}$$

其中上式中的 x_i 为输入。其中 y_i 为：

$$y_i = \begin{cases} +1, & u_j \geq \theta_j \\ -1, & u_j < \theta_j \end{cases} \tag{3.44}$$

输出层是一个 n 维向量，假设该向量在 t 时刻的定义为：

$$Y(t) = [y_1(t), y_2(t), \cdots, y_n(t)]^T \tag{3.45}$$

所以输出层向量的状态一共有 2^n 中，所以我们可以认为网络的状态也有 2^n 种，而输出状态之间的迭代公式如下所示：

$$y_j(t+1) = f[u_j(t)] = \begin{cases} +1, & u_j(t) \geq 0 \\ -1, & u_j(t) < 0 \end{cases} \tag{3.46}$$

$$u_j(t) = \sum_{i=1}^{n} \omega_{ij} y_i(t) + x_j - \theta_j \tag{3.47}$$

其中 $y_i(t)$ 表示输出层第 i 个节点在第 t 个时刻的状态，从上式我们可以看出如果 $i = j$ 时 $\omega_{ij} = 0$ 则新状态的输入不会由旧状态的输出影响，我们称这种网络为无自反馈网络; 反之如果在 $i = j$ 时 $\omega_{ij} \neq 0$，则旧状态的输出会影响新状态的输入，所以我们称这种网络为有自反馈的网络。

(2) 连续性：

连续性 Hopfield（Continuous Hopfield Neural Network, CHNN），CHNN 神经网络与 DHNN 神经网络的网络结构相似，二者的不同点在于它传递的函数不是存在间断的函数，而是连续函数。在硬件上是由简单的线路连接来实现，其神经

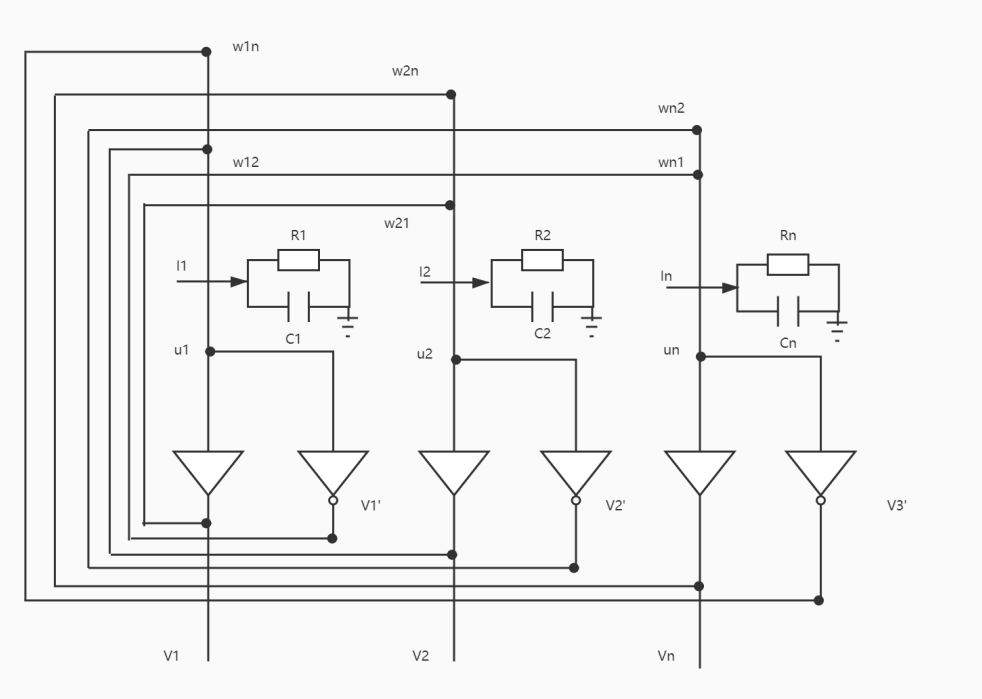

图 3-5　连续性 Hopfield 神经网络

元的输出时随着时间变化而变化的。为了模拟神经元 S 型的函数关系，则采用饱和非线性运算放大器来模拟如下图 (3-6) 所示，且该函数关系为：

$$v_i = f_i(u_i) \tag{3.48}$$

对于 N 个节点的 CHNN 神经元的迭代变化规则使用微分方程来规定即：

$$\begin{cases} C_i \dfrac{du_i}{dt} = \displaystyle\sum_{j=1}^{N} T_{ij} v_j - \frac{u_i}{R_i} + I_i & (3.49) \\ v_i = f_i(u_i) & (3.50) \\ i = 1, 2, 3, \cdots, N & (3.51) \end{cases}$$

能量函数的定义为：

$$E = -\frac{1}{2} \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} T_{ij} v_i v_j - \sum_{i=1}^{N} v_i I_i + \sum_{i=1}^{N} \frac{1}{R_i} \int_0^{v_i} f^{-1}(v) dv \tag{3.52}$$

虽然 CHNN 的能量函数在表达形式上与物流意义上的能量函数一致，但是它的

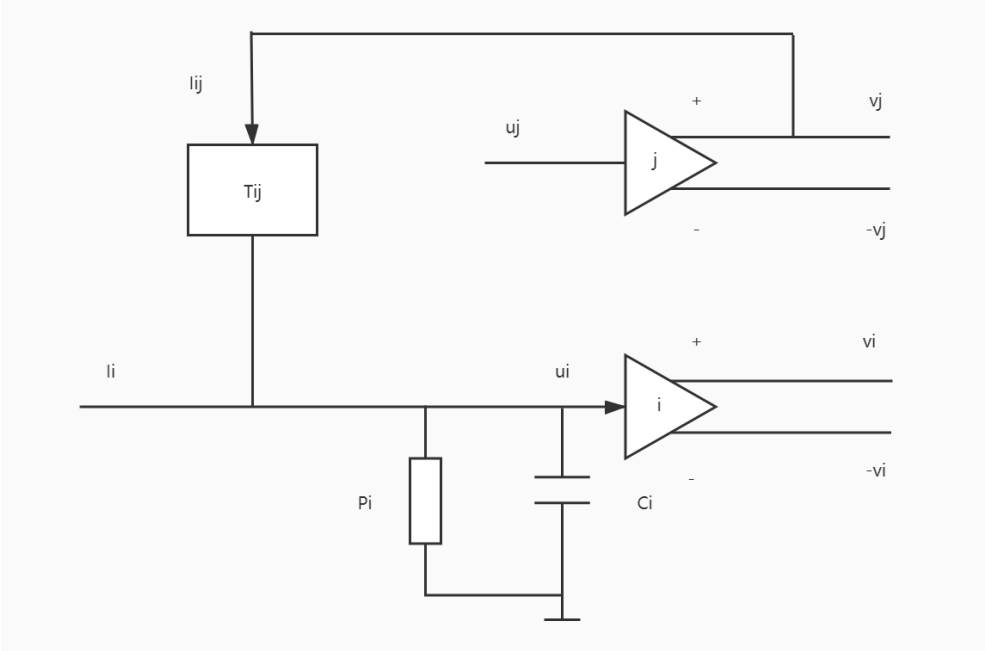

图 3-6　饱和非线性运算放大器

实际意义时为了表示网络的状态的变化。

(二)　Hopfield 网络的学习算法

Hopfield 的工作方式分为异步与同步，其中异步工作方式也叫串行工作方式，即在任意时刻只有一个神经元进行状态更新，其状态更新规则遵循 (3.46) 与 (3.47) 二式。而同步工作方式又被称为并行工作方式，即在某一时刻全部或部分神经元的状态进行更新。下面说明离散型与连续性 Hopfield 网络的学习规则。

（1）离散型：

1. 外积法：

假设其样本数据为 $\{t^1, t^2, t^3, \cdots, t^N\}$ 其中任意元素的状态为 1，-1，学习规则如下：

$$W = \sum_{k=1}^{N} [t^k (t^k)^T - I] \tag{3.53}$$

该规则被称为“外积规则”。下面我们简单描述其建立 Hopfield 神经网络的步骤。

Step 1: 将样本数据根据 (3.53) 计算权系数矩阵。

Step 2: 初始化网络输出值，并规定网络的迭代次数。

$Step$ 3: 迭代公式如下所示：

$$y_i(k+1) = f(\sum_{j=1}^{N} \omega_{ij} y_j) \tag{3.54}$$

$Step$ 4: 迭代中止条件定为网络状态不变或网络迭代到达最大次数，如果不满足终止条件则返回 Step 3。

2. 正交化法：

算法步骤如下：

$Step$ 1： 初始化 N 个输入设为 $t = \{t^2, t^2, \cdots, t^(N-1), t^N\}$, 参数 τ, h。

$Step$ 2： 计算 $= \{t^2 - t^N, t^2 - t^N, \cdots, t^(N-1) - t^N\}$。

$Step$ 3： 奇异值分解 $A = USV^T$, 并求 $K = rank(A)$。

$Step$ 4： 由 $U^p = \{U^1, U^2, U3, \cdots, U^k\}$ 与 $u^m = \{u^(K+1), u^(K+2), \cdots, u^N\}$ 求出 $T^p = \sum_{i=1}^{k} u^i (u^i)^T, T^m = \sum_{i=K+1}^{N} u^i (u^i)^T$。

$Step$ 5： 求 $W^t = T^p - \tau \times T^m, b^t = t^N - W^t \times t^N$。

$Step$ 6： 求 $W = exp(h \times W^t)$。

$Step$ 7： 求 $b = U \times \begin{bmatrix} C_1 \times I(K) & 0(K, N-K) \\ 0(N-K, K) & C_2 \times I(N-K) \end{bmatrix} \times U^T \times b^t$ 其中 $C_1 = exp(h) - 1, C_2 = -[exp(-\tau \times h) - 1]/\tau$。

（2）连续型：连续型 Hopfield 神经网络解决优化问题主要分为以下步骤：

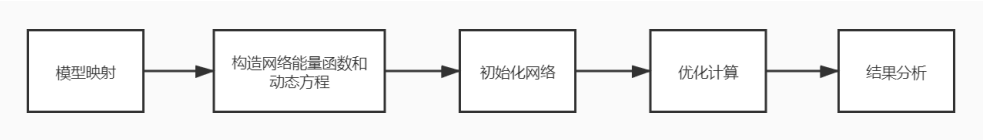

图 3-7 Hopfield 解决优化计算的步骤

(三) Hopfield 网络的稳定性

能量函数设为 $E(t)$，它的定义如下：

$$E(t) = -\frac{1}{2} \sum_{j=1}^{n} \sum_{i=1}^{n} \omega_{ij} V_i(t) V_j(t) - \sum_{j=1}^{n} V_j(t) I_j + \sum_{j=1}^{n} \frac{1}{R_j} \int_0^{V_j(t)} g^{-1}(V) dV \tag{3.55}$$

其中，g^{-1} 代表 $V_j(t) = g_j(U_j(t))$ 的反函数。下面对 $E(t)$ 求导：

$$\frac{\mathrm{d}E(t)}{\mathrm{d}t} = -\frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n}\left[w_{ij}\frac{\mathrm{d}V_i(t)}{\mathrm{d}t}V_i(t) + w_{ij}V_i(t)\frac{\mathrm{d}V_j(t)}{\mathrm{d}t}\right] - \sum_{j=1}^{\infty} I_j\frac{\mathrm{d}V_j(t)}{\mathrm{d}t} + \sum_{j=1}^{n}\frac{U_j(t)}{R_j}\times\frac{\mathrm{d}V_j(t)}{\mathrm{d}t} \tag{3.56}$$

若 $\omega_{ij} = \omega_{ji}$ 则上式可变为：

$$\frac{\mathrm{d}E(t)}{\mathrm{d}t} = -\sum_{j=1}^{n}\sum_{i=1}^{n} w_{ij}V_i(t)\frac{\mathrm{d}V_j(t)}{\mathrm{d}t} - \sum_{j=1}^{n} I_j\frac{\mathrm{d}V_j(t)}{\mathrm{d}t} + \sum_{j=1}^{n}\frac{U_j(t)}{R_j}\times\frac{\mathrm{d}V_j(t)}{\mathrm{d}t} \tag{3.57}$$

$$= -\sum_{j=1}^{n}\frac{\mathrm{d}V_j(t)}{\mathrm{d}t}\left[\sum_{j=1}^{n} w_{ij}V_i(t) + I_j - \frac{U_j(t)}{R_j}\right] \tag{3.58}$$

将 (3.51) 带入上式则：

$$\frac{\mathrm{d}E(t)}{\mathrm{d}t} = -\sum_{j=1}^{n}\frac{\mathrm{d}V_j(t)}{\mathrm{d}t}\times C_j\frac{\mathrm{d}U_j(t)}{\mathrm{d}t} \tag{3.59}$$

因为 $V_j(t) = g_j(U_j(t))$，所以 $U_j(t) = g_j^{-1}(V_j(t))$，上式可以改成：

$$\frac{\mathrm{d}E(t)}{\mathrm{d}t} = -\sum_{j=1}^{n}\frac{\mathrm{d}V_j(t)}{\mathrm{d}t}\times C_j\frac{\mathrm{d}\left[g_j^{-1}\left(V_j(t)\right)\right]}{\mathrm{d}t} \tag{3.60}$$

$$= -\sum_{j=1}^{n}\frac{\mathrm{d}V_j(t)}{\mathrm{d}t}\times C_j\frac{\mathrm{d}\left[g_j^{-1}\left(V_j(t)\right)\right]}{\mathrm{d}V_j(t)}\times\frac{\mathrm{d}V_j(t)}{\mathrm{d}t} \tag{3.61}$$

$$= -\sum_{j=1}^{n}\left[\frac{\mathrm{d}V_j(t)}{\mathrm{d}t}\right]^2\times C_j\times\left[g_j^{-1}\left(V_j(t)\right)\right]' \tag{3.62}$$

若 $g(u)$ 连续有界且单调递增，则 $g^{-1}(u)$ 也单调递增，所以 $[g_j^{-1}(V_j(t))]' > 0$，且 $C_j > 0, (\frac{dV_j(t)}{dt}) \geq 0$，所以 $\frac{dE(t)}{dt} \leq 0$，且当 $\frac{DV_j(t)}{dt}$ 时 $\frac{dE(t)}{dt} = 0$

四、 网络的应用设计

(一) 问题描述

组合优化（combinatorial optimization）是从所有解中找到符合约束条件的最优解，设解空间 $\Omega = \{s_1, s_2, \cdots, s_n\}$，$C(s_i)$ 则为目标函数值，其最优解的定义为：

对于所有 $s_i \in \Omega$，存在 $C(s^*) = min(C(s_i))$，典型的组合优化问题为 TSP（Traveling Salesman Problem）问题，即遍历 N 个城市 1 次后回到原点，且所需路程最小。对于该组合问题他的可行解个数为 $\frac{1}{2}(N-1)!$，当 N 的值越来越大时计算量随着指数增加。下面我们将以 TSP 问题为例，构造 $Hopfield$ 网络模型。

(二) 构造能量函数与动态方程

网络能量方程分为目标函数项与约束项，即定义为:

$$E = \frac{A}{2}\sum_{x=1}^{N}\left(\sum_{i=1}^{N}V_{xi}-1\right)^2 + \frac{A}{2}\sum_{i=1}^{N}\left(\sum_{x=1}^{N}V_{xi}-1\right)^2 + \frac{D}{2}\sum_{x=1}^{N}\sum_{y=1}^{N}\sum_{i=1}^{N}V_{ij}d_{xy}V_{y,t+1} \tag{3.63}$$

其中，前两项为约束项，第三项为目标函数项，其中第三项时待优化的。
其中从 (3.59) 我们可以得到网络的方程为：

$$\frac{\mathrm{d}U_{ni}}{\mathrm{d}t} = -\frac{\partial E}{\partial V_{xi}} = -A\left(\sum_{i=1}^{N}V_{xi}-1\right) - A\left(\sum_{y=1}^{N}V_{yi}-1\right) - D\sum_{y=1}^{N}d_{xy}V_{y,i+1} \tag{3.64}$$

(三) 初始化网络

初始化网络的输入：

$$U_{xi}(t) = U_0\ln(N-1) + \delta_{xi} \quad (x,i = 1,2,\cdots,N;t=0) \tag{3.65}$$

其中 $U_0 = 0,1N$ 代表要遍历的地点总个数，δ_{xi} 为范围为 $(-1,+1)$ 的随机数。

(四) 优化计算

$Step$ 1：计算各城市之间的距离矩阵。

$Step$ 2：初始化网络参数。

$Step$ 3：根据 (3.64) 求 $\frac{dU_{xi}}{dt}$，使用一阶欧拉法求解，$U_{xi}(t+1) = U_{xi}(t) + \frac{dU_{xi}}{dt}\triangle T$；

$Step$ 4：根据 $V_{xi}(t) = g\left(U_{xi}(t)\right) = \frac{1}{2}\left[1+\mathrm{tansig}\left(\frac{U_x(t)}{U_0}\right)\right]$ 求解 $V_{xi}(t)$；

$Step$ 5：根据 (3.63) 求解 E；

$Step$ 6：判断是否达到迭代次数，达到程序终止，否则迭代次数加一，返

回 *Step* 3： 。

第四章　解决路径规划问题的其他优化算法

除了使用 Hopfield 神经网络解决路劲规划问题外，我们下面介绍另外几种常用于解决路径规划的算法。

一、　暴力穷举搜索

穷举法是解决该问题的最简单的一种算法，它为了达到求解的目的，利用计算机强大的算力来寻找与比较各个可行解，但是，穷举算法在城市数量较多时效率不高。下面是对待路径规划问题上的穷举算法流程图：

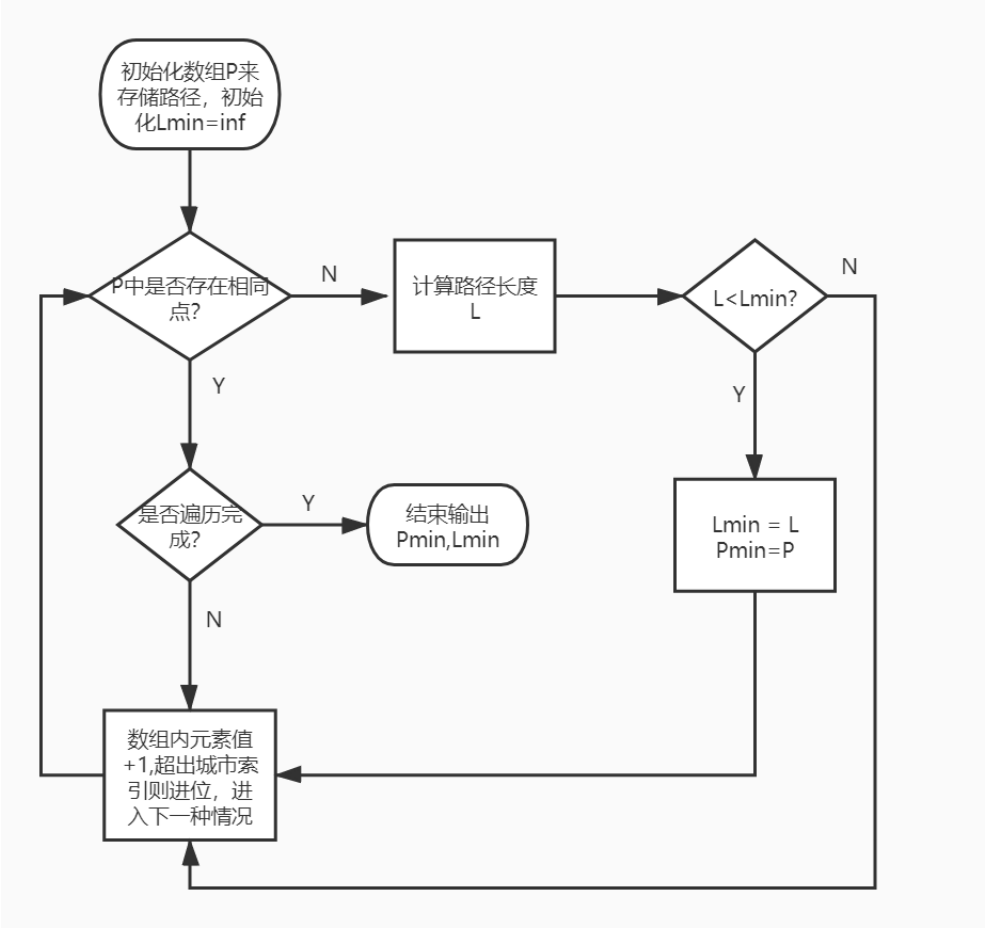

图 4-1　暴力穷举搜索流程图

其中 $P = [p_1, p_2, \cdots, p_n]$ 为存储路径的数组，城市的标号为 $\{1, 2, 3, \cdots, n\}$，P

遍历为下一种路径的方法为，$p_n + 1$，若 $p_n = n$ 则 $p_{n-1} + 1, p_n = 1$，以此类推，遍历得到解空间所有路径集合。

二、 剪枝法

剪枝法又称为分支限界法，我们在解决路径规划问题时分为两部分来介绍此算法。

(一) 剪枝法初步

1. 求解目标：找到一个解，该解既满足约束条件，又是所有可行解中最优的。

2. 搜索方式：搜多解空间树的方式为最小耗费优先的原则或以广度优先的原则进行遍历搜索，其中是以最小耗费优先还是以最大收益优先式问题而定。

3. 基本思想：在分支限界法里，每一个节点有且只有一次机会称为扩展结点，如果一个活结点成为了扩展接待你，则产生该节点的所有儿子节点，在这些儿子节点种，要将不可行解或可行解中的非最优解舍弃，其余的儿子节点加入活结点表中，然后在活结点表中去下一节点成为当前扩展节点，并重复节点的扩展过程，这个步骤一直重复，直到找到所需的解或活结点表为空则该步骤结束。

4. 具体实现：

以 TSP 为例，首先我们用贪心法求出该问题的近似解，即为该问题的上界，把无向图的邻接矩阵每一行最小的元素相加可以得到一个简单的下界，但是我们可以考虑得到一个信息量更大的下界，我们考虑 TSP 问题的可行解中，路径上的每个城市都有两个邻接边，一个进入城市的一个是离开城市的，如果将矩阵每一行最小两个元素相加在除以 2，再去取上整，则我们得到了一个相对合理的下界。注意该下界并不是一个合法的选择，它仅仅为参考下界。所以我们得到了 TSP 问题解的区间 $[\sum_{i=1}^{n-1} min(Distance(d_i, j)) + Distance(j_n, 1), \sum_{i=1}^{n} \frac{min(Distance(i))+secondmin(Distance(i))}{2}]$。其中 d_i 代表可行路径中第 i 个城市的索引。其中起点为第一个城市。$min(Distance(i))$ 即为求解邻接矩阵第 i 行的最小元素值，$secondmin(Distance(i))$ 即为求解邻接矩阵第 i 行第二小的元素值。以下是分支界定法算法：经典剪枝法，经典思想是搜索可行解，发现解空间树的值不如最优解则中止搜索，但是其效率不高，下面我们对剪枝法进行改进。

Algorithm 1 TSP 分支定界法

Input: input 邻接矩阵，活结点表 RT；
Output: output 最短路，最短路程；

根据限界函数计算目标函数的下界 down；采用贪心法得到上界 up；
将 RT 表初始化为空：
for $i = 1; i \leq; i++$ **do**
    $x[i] = 0;$
$k = 1; x[1] = 1;$
while $x[i] \leq n$ **do**
    if 路径上顶点不重复 **then**
        计算路程花费, 记为 lb;
        if $lb \leq up$ **then** 将路径与花费存在 PT 表中;
        $x[i] = x[i] + 1;$
    if $i == n$&& 叶子节点的 lb 在 RT 表中最小 **then**
        叶子节点对应的解输出。
    else
        if $i == n$ **then**
            在 RT 中取值最小节点 lb，令 $up = lb$，将表中超过 up 的节点删除。
    k 为 RT 中最小路径上点的个数。
return 最短路，最短路程；

(二) 启发式算法的剪枝法

我们对经典剪枝法进行改进，利用启发式算法来估计解空间树某一结点的剩余部分的所代表的路径代价，首先我们先介绍最近邻点算法（nearest neighbor heuristic），该算法的具体步骤如下所示：

$Step\ \ 1:$ 任选点 $r \in V$ 起点，令 $C = r, h = r$。

$Step\ \ 2:$ 找到下一个相邻点，该点满足 $k \in V \backslash C$ 使得 $C_{hk} = min\{c_{hj}, c_{jh}\}$，将 k 加入 C 中。

$Step\ \ 3:$ 当 $|C| = |V|$ 算法终止，否则 $h = k$ 则返回 $Step\ \ 2$。

我们将最近邻点算法来估计路径剩余部分的花费。所以，剪枝法的算法流程图修改如下：

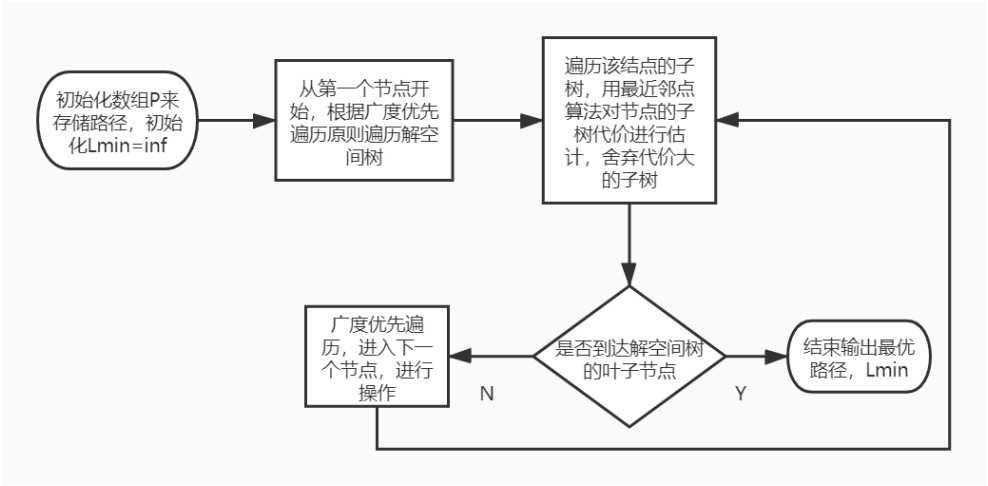

图 4-2　利用启发式算法的剪枝法解决 TSP 问题算法流程图

三、　智能算法简介

(一)　粒子群算法

粒子群算法（particle swarm optimization, PSO）是一种群体智能的优化算法，和蚁群算法，鱼群算法一样。该算法思想来源于鸟类捕猎的行为，即鸟群捕食时最有效的是搜索离目标最近的鸟的区域。算法首先随机一群粒子，根据每个粒子对应的适应度函数来决定该粒子的移动方向与距离。

假设粒子 x_i 在空间的位置为 $x_i = \{x_1, x_2, \cdots, x_n\}$ 初始速度为 $v_i = \{v_1, v_2, \cdots, v_n\}$，每个粒子已知自己经过的最好位置与现在位置，还知道群体的最好位置。则速度与位置的更新规则如下：

$$v_i = v_i + c_1 \times rand() \times (pbest_i - x_i) + c_2 \times rand() \times (gbest_i - x_i) \tag{4.1}$$

$$x_i = x_i + v_i \tag{4.2}$$

其中上式的 v_i 代表粒子的速度，$rand()$ 代表 $[0, 1]$ 的随机数，x_i 代表粒子的位置，$c1, c2$ 表示学习因子。（4.2）也被称为 PSO 标准公式。

以此为基础，为了调节群体全局与局部寻优的权重，上式进行了改进。

$$v_i = \omega \times v_i + c_1 \times rand() \times (pbest_i - x_i) + c_2 \times rand() \times (gbest_i - x_i) \tag{4.3}$$

其中上式 ω 被称为惯性因子，其值较大时，全局寻优能力强，但局部寻优能力弱，当此值较小时，全局寻优能力弱，局部寻优能力强，但其值大于零。在算法实现时动态的 ω 可能会有更好的效果。目前采用的时线性递减权值策略（Linearly Decreasing Weight, LDW）：

$$\omega^{(t)} = \frac{(\omega_{ini} - \omega_{end})(G_k - g)}{G_k} + \omega_{end} \tag{4.4}$$

上式中 G_k 代表最大迭代次数，ω_{ini} 代表初始惯性权值，ω_{end} 表示迭代到最大进化次数时的惯性权值。

下面给出 PSO 算法流程图:

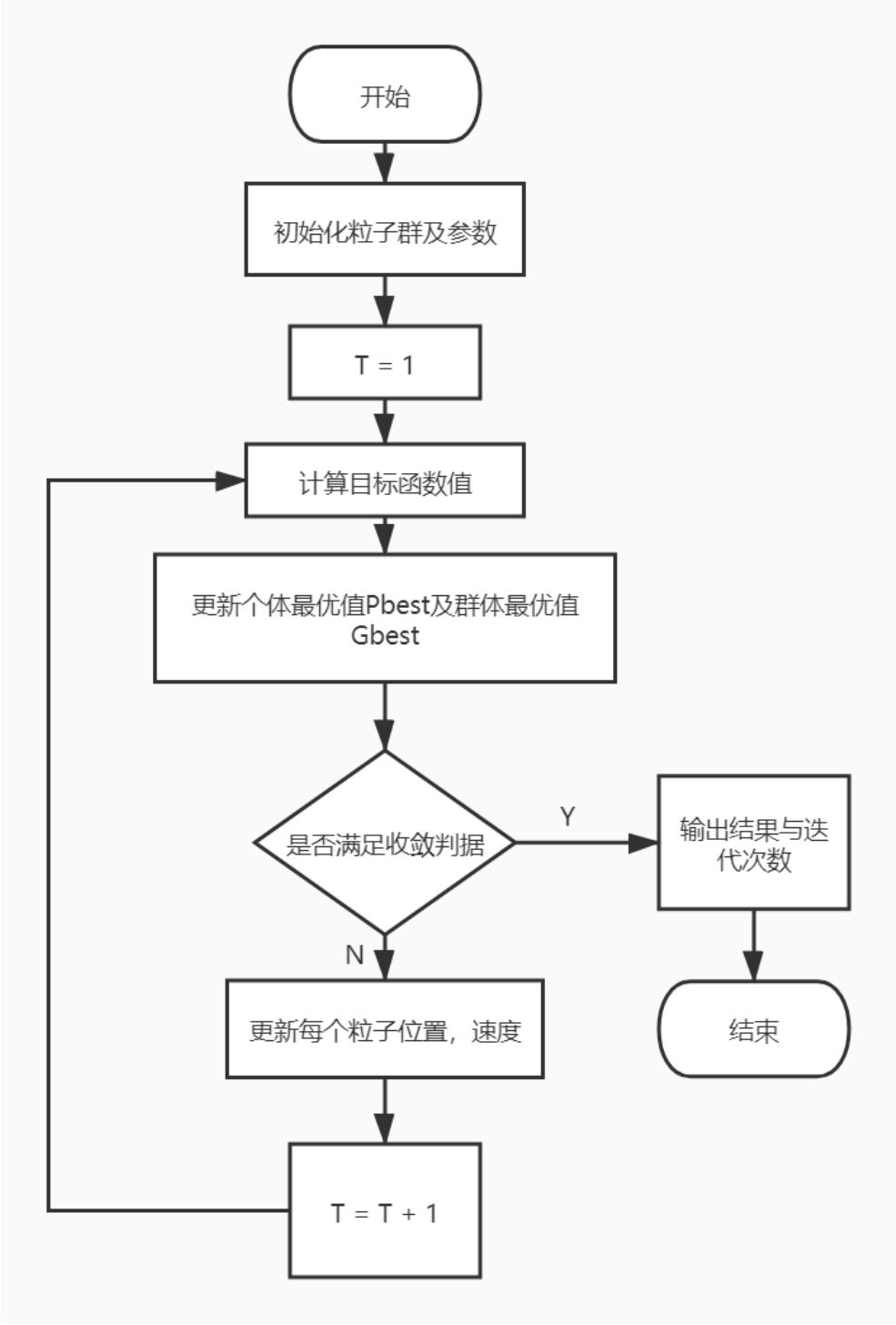

图 4-3　粒子群算法流程图

(二)　模拟退火算法

统计力学证明了物质的粒子不同结构对应于粒子的不同能量水平，高温下粒子可以自由活动，低温下粒子的活动渐渐减弱，所以从高温开始降温，粒子可

以在每个温度达到某种平衡，我们称为粒子热平衡，我们称这种行为为退火。系统降温到被冷却时，材料成为了低能状态下的晶体。根据此现象，研究人员发明创立了模拟退火算法。

Metropllis 算法用数学模型描述了材料退火的过程。$E(i)$ 为 i 状态下的能量，即材料的状态转换遵循如下规律：

（1）若 $E(j) \leq E(i)$，状态转换。

（2）若 $E(j) > E(i)$，状态以一定概率改变，即 $\frac{E(i)-E(j)}{KT}$。其中 K 为玻尔兹曼常数常应用于物理学领域，T 为材料温度。

当材料达到热平衡时，其状态的概率满足玻尔兹曼分布：

$$P_T(x=i) = \frac{e^{-\frac{E(i)}{KT}}}{\sum_{j\in S} e^{-\frac{E(j)}{KT}}} \tag{4.5}$$

所以：

$$\lim_{T\to\infty} \frac{e^{-\frac{E(i)}{KT}}}{\sum_{j\in S} e^{-\frac{E(j)}{KT}}} = \frac{1}{|S|} \tag{4.6}$$

其中 S 为状态集合。$|S|$ 表示集合状态数量，当温度下降时：

$$\lim_{T\to 0} \frac{e^{-\frac{E(i)-E_{\text{ziit}}}{KT}}}{\sum_{j\in S} e^{-\frac{E(j)-E_{\min}}{KT}}} = \lim_{T\to 0} \frac{e^{-\frac{E(i)-E_{\text{mie}}}{KT}}}{\sum_{j\in S_{\text{mia}}} e^{-\frac{E(j)-E_{\min}}{KT}} + \sum_{j\mathbb{E}S_{\min}} e^{-\frac{E(j)-E_{\min}}{KT}}} \tag{4.7}$$

$$= \lim_{T\to 0} \frac{e^{-\frac{E(i)-E_{\min}}{KT}}}{\sum_{i\in S} e^{-\frac{E(j)-E_{\text{mia}}}{KT}}} = \begin{cases} \frac{1}{|S_{\min}|} & \text{若} i \in S_{\min} \\ 0 & \text{其它} \end{cases} \tag{4.8}$$

其中 $E_min = \min_{j\in S} E(j), S_min = \{i|E(i) = E_min\}$。

从 (4.8) 我们看出材料会进入最小能量状态时，大概率材料温度很低。下面我们将退火思想移植到优化问题上来。以组合优化问题为例，设目标函数为 $f: x \to R_+, x \in S, R_+ = \{y|y \in R, y > 0\}$，$N(x) \subseteq S$ 代表 S 代表定义域。

设该问题的初始温度为 T_0，初始解为 $x(0)$，以及由初始解生成的下一个解

$x' \in N(x(0))$:

$$P(x(0) \to x') = \begin{cases} 1 & 若 f(x') < f(x(0)) \\ e^{-\frac{f(x')-f(x(0))}{T_0}} & 其它 \end{cases} \tag{4.9}$$

其中 $P(x(0) \to x')$ 代表 x' 是否为新解的概率。上式可以这么解释，即若 x' 的函数值比上一个解小，则 x' 必定为新解，否则以一定概率作为新解。

将此式推广则可以得到：

$$P(x(0) \to x') = \begin{cases} 1 & 若 f(x') < f(x(k)) \\ e^{-\frac{f(x')-f(x(k))}{T_k}} & 其它 \end{cases} \tag{4.10}$$

若重复多次该过程即为不断求得新解且系统逐渐降温的过程。在每个 T_i 下每个新状态 $x(k+1)$ 依赖于前一个状态，且于前 $k-1$ 个状态无关，这是一个马尔可夫过程，所以我们进一步可以得到以下结论：$x(k)$ 生成 x' 的概率在 $N(x(k))$ 中服从均匀分布，且概率满足（4.10），所以平衡态下的解分布如下所示：

$$P_i(T_i) = \frac{e^{-\frac{f(x_i)}{T}}}{\sum_{j \in S} e^{-\frac{f(x_i)}{T_i}}} \tag{4.11}$$

温度 T 为 0 时，解的分布为：

$$P_i^* = \begin{cases} \frac{1}{|S_{\min}|} & 若 x_i \in S_{\min} \\ 0 & 其它 \end{cases} \tag{4.12}$$

其中 $\sum_{x_i \in S_{\text{niin}}} P_i^* = 1$，所以我们推断，经过足够次降温，且每个温度有多次状态转移，则全局最优解一定会被找到。

模拟退火算法在计算机实现时可能会出现如下问题：1. 降温过慢，找到优质解的概率较大，但时间较长，降温过快，找到优质解的概率较小，但时间较短，所以在实现过程中要兼顾速度与性能。2. 若连续多次转换都没有使得状态改变则改变该温度，程序结束条件可以为，连续几个温度下状态都没有发生转变。3. 初值的选取会影响程序的性能与解的质量。

(三) 遗传算法

遗传算法（Genetic Algorithms,GA）是一种搜索算法，他的思想来源于自然选择，与遗传进化。模拟自然界的选择机制，对优化目标进行寻优求解，算法与自然界的遗传选择之间的对应关系为：适者生存具体表现为每次算法迭代完成一次后都会最大可能的留住最优解淘汰劣质解，算法中的每个个体都代表一个解，解在算法中被编码对应了染色体，其基因代表，解的每个分量，个体适应度，具体体现在目标函数上，交配根据交换优质解的特征来生成其他解，变异，改变解的某一分量以扩大搜索范围，它在算法实现方面主要分为 6 部分，下面进行详细介绍：

1. 种群初始化

随机构造可行解，并将解编码化，常见的编码方式有，实数编码，多级参数编码，有位串编码，*Grey* 编码。一般将解编码为一个实数向量。

2. 构造适应度函数：

适应度函数建立的目的是为了衡量种群中每个个体的优劣，以此来进行自然选择，适应度函数一般是有目标函数简单变换得到，若求最小值，则可以将适应度函数定义为，目标函数的倒数即为：

$$F[f(x)] = \frac{1}{f(x)} \tag{4.13}$$

在这种情况下，目标函数越大，适应度函数越小，反之，目标函数越小，适应度函数越大。

3. 选择操作：

生成新种群的一种方式，主要操作为，从原种群中挑取优质个体，淘汰劣质个体，个体被选择的概率与适应度函数成正相关，即适应度函数越大，该个体被选择的几率越大，在遗传算法中，选择的常见规则有，轮盘赌法，锦标赛法等方法，但本案例使用轮盘赌法，每个个体根据适应度的不同选中的几率不同，具体规则如下：

$$p_i = \frac{F_i}{\sum_{j=1}^{N} F_j} \tag{4.14}$$

其中，第 i 个个体的适应度为 F_i，个体个数为 N。

4. 交叉操作具体操作为，从种群选取两个个体，将两个个体的染色体的随机两端进行组合交换，衍生出来的子串会携带父串的优秀基因，若个体采用实数编码，则交换规则为：

$$a_{kj} = a_{ij}(1-b) + a_{ij}b \tag{4.15}$$

$$a_{lj} = a_{lj}(1-b) + a_{kj}b \tag{4.16}$$

其中 b 为 [0, 1] 区间内的随机数，a_{ij} 代表第 i 个染色体的第 j 位。

5. 变异操作从自然选择的方面来说变异为了维持种族基因多样性，从优化算法来说，变异增加了搜索的范围，其具体操作为，随机抽取一个个体。选择个体基因的一个位点进行变异，使其个体成为更优秀的个体，使种群基因更加丰富。其变异规则如下所示：

$$a_{ij} = \begin{cases} a_{ij} + (a_{ij} - a_{max}) * f(g), & r \geq 0.5 \quad (4.17) \\ a_{ij} + (a_{min} - a_{ij}) * f(g), & r < 0.5 \quad (4.18) \end{cases}$$

其中 a_{ij} 的上界为 a_{max}，a_{ij} 的下界是 a_{min}，$f(g) = r_2(1 - \frac{g}{G_{max}})$，其中 r_2 是个随机数，g 是迭代次数，G_{max} 代表进化数的最大值，r 代表 [0, 1] 的随机数。6. 寻优进化到一定程度后，将程序的结果为初始值对函数初始值附近的点进行寻优，寻优后在作为新的个体继续进行优化。

所以遗传算法解决优化问题的算法流程如下，以非线性优化为例：

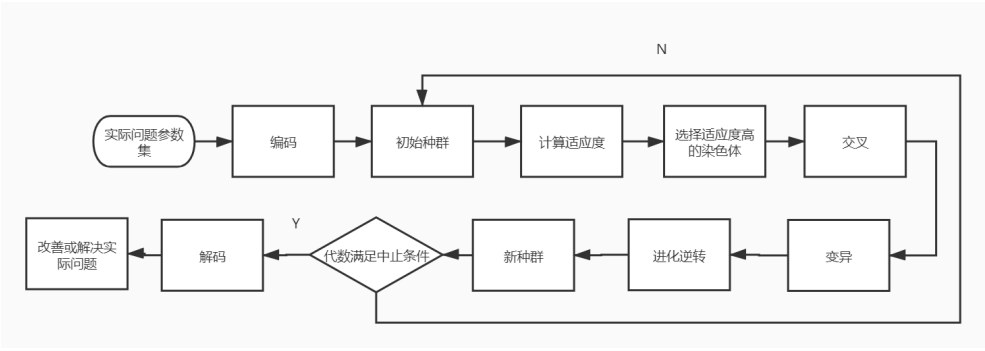

图 4-4　遗传算法流程图

(四) 蚁群算法

1991 年来自意大利的学者 M. Dorigo 首先提出了蚁群算法，人么并以此为基础逐渐发展完善了蚁群算法，科学家们发现蚂蚁会在路上释放挥发性的信息素，蚂蚁个体之间就是通过这种信息素进行交流，蚂蚁总是趋向于走更多信息素的路径，假设最短路径的蚂蚁回到了巢穴因为其路径最短时间最优，所以他的路径下的信息素浓度最高，所以走最短路径的蚂蚁会越来越多，以此形成正反馈，所有的蚂蚁都会趋于这条路径。以解决 TSP 问题为例，n 表示城市数量，m 表示蚁群的蚂蚁数量，d_{ij} 表示 i, j 城市之间的距离，$b_i(t)$ 表示在 t 时刻位于城市 i 的蚂蚁的个数。$m = \sum_{i=1}^{n} b_i(t)$；$\tau_{ij}(t)$ 为 t 时刻 i 到 j 的信息素浓度，其初值为常数，$\eta_{ij}(t)$ 表示 t 时刻 i 到 j 城市的转移期望。但人工蚁群中植入了记忆功能，随着时间推移信息素浓度会减少，所以人工蚁群模型为：

（1）在 t 时刻人工蚁的转移概率为:

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{\tau_{ij}^{\alpha}(t)\eta_{ij}^{\beta}(t)}{\sum_{v \in S} \tau_{iv}^{\alpha}(t)\eta_{iv}^{\beta}(t)}, & j \in S \\ 0, & j \notin S \end{cases} \tag{4.19}$$

其中为了调节信息素浓度与能见度，使用 α 来表示信息素浓度的重要性，β 表示可见度的重要性，两个参数都是非负的。S 表示可行点集合，及合法的点集，蚂蚁下一步可以去的城市集合。

（2）找到可行解后，信息素浓度更新方程为：

$$\begin{aligned} &\tau_{ij}(t+1) = \rho\tau_{ij}(t) + \Delta\tau_{ij}, \quad \rho \in (0,1) \\ &\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \end{aligned} \tag{4.20}$$

$\Delta\tau_{ij}^{k}$ 表示第 k 个蚂蚁在 i 到 j 城市的路上留下的信息素浓度，$\Delta\tau_{ij}$ 为信息素增量，ρ 来控制信息素挥发的速度。

所以蚁群算法步骤可以被总结为：

$Step$ 1： 初始化迭代次数，个参数初始化，蚁群位置初始化

$Step$ 2： 对于每个蚂蚁威威按概率 p_{ij}^{k} 进行更新。

$Step$ 3： 计算蚁群中每个个体的目标函数值，记录最优解。

$Step$ 4： 按照 (4.20) 更新信息素浓度。

Step 5： 迭代次数加一，若达到预定迭代次数则退出，否则转 *Step* 2。为了该模型能够适应大规模求解大规模 TSP 问题的能力，将 (4.20) 改为：

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + (1-\rho)\Delta\tau_{ij}, \quad \rho \in (0,1) \tag{4.21}$$

对应的：

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{1}{d_{ij}}, & (i,j) \in BE \\ 0, & \text{其它} \end{cases} \tag{4.22}$$

上式的 BE 代表最优线路的边的集合。

四、 动态规划

动态规划（dynamic programming）通过将待求解的问题划分为子问题的方法来求解。与将问题划分为互不相交的子问题然后递归的求解子问题的分治算法不同，动态规划不同它应用于子问题重叠的情况，子问题递归求解，再将其划分为更小的问题，我们将其称为子子问题，子子问题在动态规划中只会被求解一次而在分治算法中会被重复求解，因为动态规划会将解存到一个表格中，避免了重复运算。我们将动态规划具体分为四个步骤：

Step 1： 构造并寻找解的结构。

Step 2： 递归的定义解。

Step 3： 自底向上的求解问题的解。

Step 4： 利用已知信息得到最优解。下面我们来推导 TSP 问题的动态规划方程：

我们将问题划分为以下几种情况：

(1) 若 V' 为空是，则 $d(i,V')$ 表示点 i 直接回到了原点。

(2) 若 V' 不为空，则 $d(i,V')$ 代表了对子问题的最优解，程序中会在 V' 的集

合中逐一尝试最终得到最优解：所以 TSP 问题的动态规划方程如下：

$$d(i,V)=\begin{cases} c_{is} & V=\phi, i\neq s \\ \min_{hcr}\{c_{ik}+d(k,V-\{k\})\} & V\neq\phi \end{cases} \tag{4.23}$$

第五章　数据预处理

为了构造程序所需要的数据结构，我们根据参数来爬取指定经纬度的 osm 地图数据，然后转化为 shp 地图数据，读取 shp 地图数据来构造图的数据结构—邻接矩阵。

一、　数据爬取

(一)　爬取 osm 数据

OPen Street Map 简称 OSM，是一个由网络大众共同打造的免费开源服务，在上面你可以下载指定经纬度的地图信息，该组织由史蒂夫·克斯特与 2004 年 7 月创立，2006 年 4 月 OSM 基金会成立，他鼓励社区的用户自由的发布地理数据，并与社区所有人分享并使用地理数据，为了搜集本项目的数据，我们在 osm 中采用爬虫技术爬取 osm 文件。下面是分析过程：

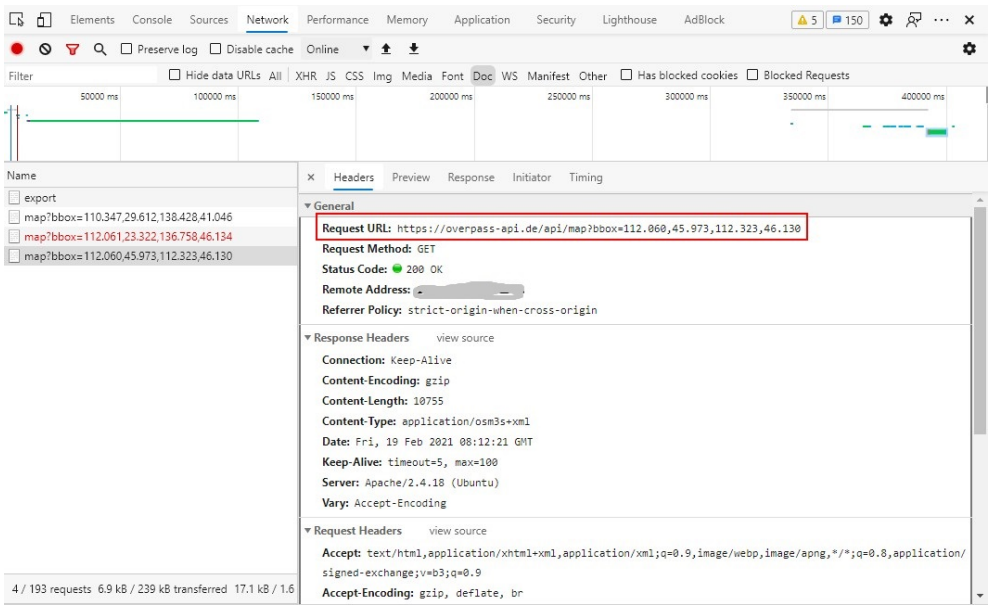

图 5-1　调试信息 General 项

我们从上图可以看出下载的 osm 文件的请求地址，其中请求类型为 GET 表明该数据的请求信息加载在网址中，且成功返回数据，状态码为 200，返回的数

据大小为 10755。然后我们查看该请求的请求头信息：

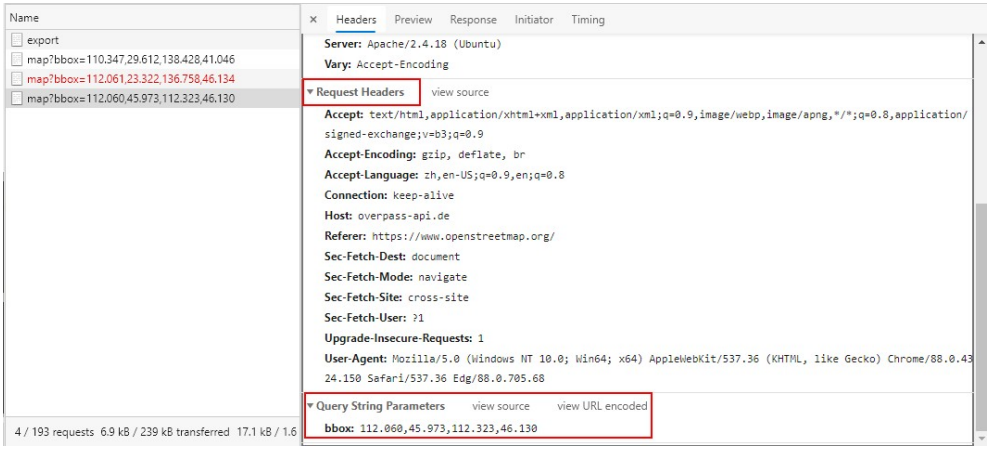

图 5-2　调试信息请求头

根据请求头我们发现该网站在请求头中并不存在验证本机的信息，说明该网址并不存在反爬虫，所需无需手动构造请求头，让程序请求时自动生成即可，所以程序为：

```
import requests
import json
import sys
import  zipfile
for  arg  in  sys.argv:
    bbox = arg # 经纬度信息作为参数传递
osm_path = '/TSP_hopfield/shp_distance/map'
zip_file  = '/TSP_hopfield/shp_distance/map.zip'
def  download_osm():
    url  = 'https://overpass-api.de/api/map?bbox='+ bbox #
        115.3141,38.8425,115.4748,38.9303
    myfile  = session.get(url  ,  allow_redirects=True)
    open ( osm_path, 'wb').write(myfile.content)
session  = requests.session()
# 下载 osm 文件
```

```
15 download_osm()
```

代码 **5.1**　爬取 **osm** 数据代码

为了使得其余程序能够直接调用此文件来下载特定区域内的 osm 数据，我们的程序会读取命令行列表，来动态下载地图数据，下面为了将地图数据转变为计算机好读取的格式我们将其转化为 shp 类型地图文件。

(二)　转化成 shp 数据

首先打开”https://geoconverter.hsr.ch/” 网站，如下所示：

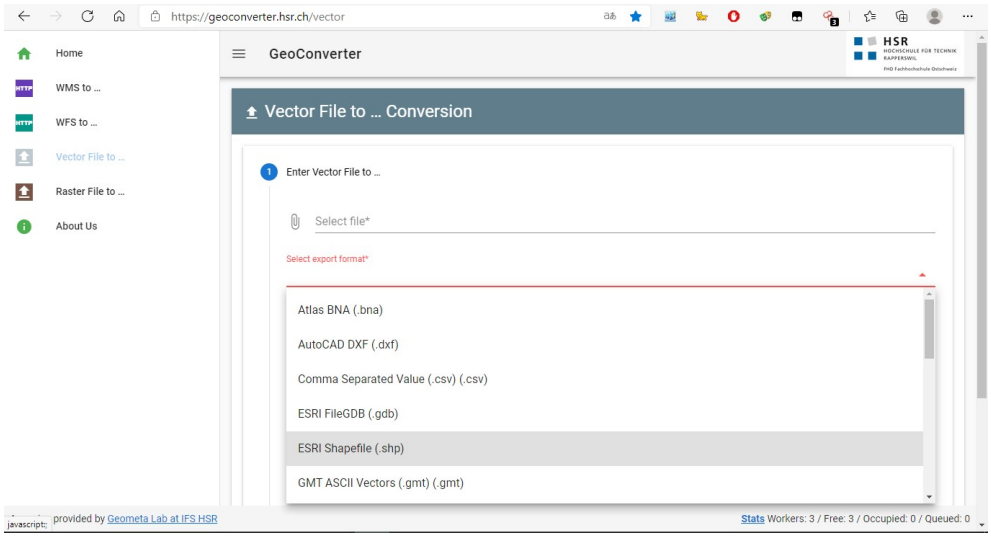

图 5-3　进入 geoconverter 网站

我们上传 osm 文件，并调整转化格式。然后点击转换，即可下载转换后的文件，但是为了能够自动化的进行文件转换，我们采用爬虫技术，来实现文件的上传，转换，下载的步骤，为了模拟该流程，我们首先进行一次数据的提交转换，来观察其过程。

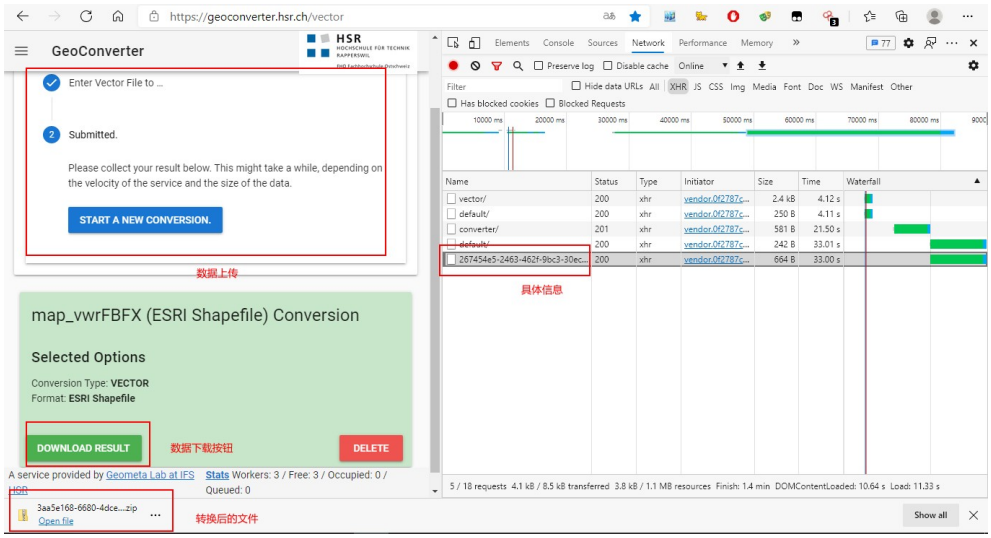

图 5-4　模拟提交转换文件

其中转化后的 shp 的文件在下载的 zip 压缩文件内，他被分为了 3 个部分分别为，路网，水网，建筑网三个文件，我们程序只需提取路网文件即可。再上图中我们在调试 XHR 栏目中可以看到许多请求过程文件，但是值得注意的只有 converter/请求与由长数字字母构成的 267454e5-2463-462f-9bc3-30ecbe41956d/请求，并且查看文件发送时间后发现，converter/请求发送时间较早，而 267454e5-2463-462f-9bc3-30ecbe41956d/请求发送较晚，并且在查看请求内容后我们发现，converter/请求时发送文件的并返回 267454e5-2463-462f-9bc3-30ecbe41956d/请求名称即：

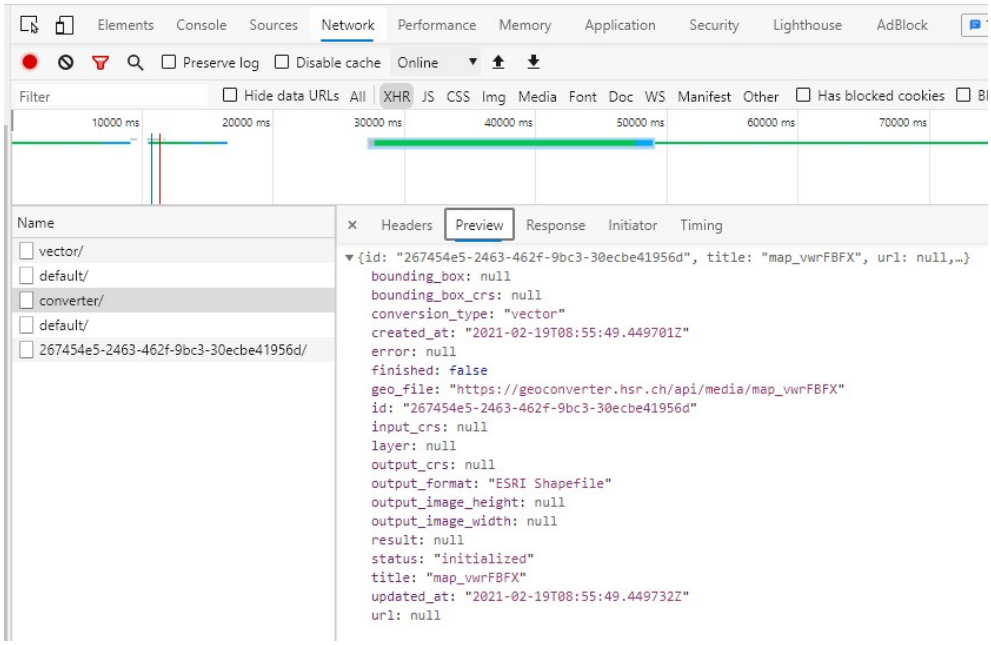

图 5-5　请求文件信息

所以我们的程序会对 multipart/form-data 请求进行模拟，通过 post 请求将文件发送到服务器。并保存服务器响应数据，读取响应数据的 id 值，并以此构造第二次请求的 url，这就是图中的 267454e5-2463-462f-9bc3-30ecbe41956d/请求，请求详细信息如下：

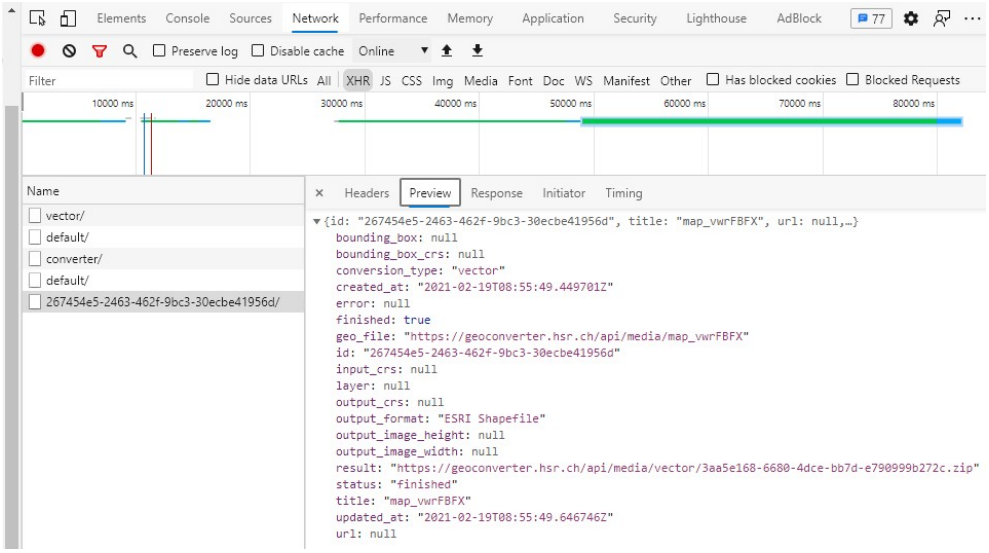

图 5-6　第二次请求信息

从以上的信息中提取到转化后的文件下载地址。在此次模拟提交中是:”https:

//geoconverter.hsr.ch/api/media/vector/3aa5e168-6680- 4dce-bb7d-e790999b272c.zip”，转换结束，然后我们以此来编写代码来代替人工进行自动化请求转化下载步骤，然后解压文件提取 shp 文件，代码如下：

```
    second_headers = {
    'authority': 'geoconverter.hsr.ch',
    'method': 'GET',
    'path': '',
    'scheme': 'https',
    'accept': 'application/json, text/plain, */*',
    'accept-encoding': 'gzip, deflate, br',
    'accept-language': 'zh,en-US;q=0.9,en;q=0.8',
    'referer': 'https://geoconverter.hsr.ch/vector',
    'sec-fetch-dest': 'empty',
    'sec-fetch-mode': 'cors',
    'sec-fetch-site': 'same-origin',
    'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
        AppleWebKit/537.36 (KHTML, like Gecko) Chrome
        /87.0.4280.88 Safari/537.36 Edg/87.0.664.66',
}
# 将 osm 文件转化为 shp 文件

# 构造 multipart/form-data 用来上传 osm 文件
files = {'geo_file': ('file', open(osm_path, 'rb'),'application
    /octet-stream'),
        'conversion_type':( None, 'vector'),
        'output_format':( None, 'ESRI Shapefile')
        }
# 不断的请求上传直到响应状态码正确
while True:
```

```
24      res = session.post(url, files=files)
25      if res.status_code // 100 == 2:
26          break
27  # 获取响应 id
28  id = res.text.split('"')[3]
29  # 构造请求头，准备进行第二次请求，下载转换的 zip 文件
30  second_url = 'https://geoconverter.hsr.ch/api/converter/' + id
        + '/'
31  path = '/api/converter/'+ id +'/'
32  second_headers['path'] = path
33  # 不断的进行第二次请求直到返回正确的 zip 文件下载地址
34  while True:
35      res = session.get(second_url, headers = second_headers)
36      if res.status_code // 100 == 2 and json.loads(res.text)['
            result'] != None:
37          break
38  # 将 zip 下载地址从 text->json 中提取出来
39  res_dict = json.loads(res.text)
40  shp_download_url = res_dict['result']
41  # 压缩文件名称
42  yasuo_file = shp_download_url.split("/")[-1].split('.')[0]
43  # 下载 shp 的压缩文件并保存
44  myfile = session.get(shp_download_url, allow_redirects=True)
45  open(zip_file, 'wb').write(myfile.content)
46  with zipfile.ZipFile(zip_file) as zf:
47      zf.extractall()
48  shp_path = '/TSP_hopfield/shp_distance/' + yasuo_file + '/
        converted.shp'
49  print(shp_path)
```

代码 **5.2** 转化为 **shp** 数据

二、 图数据结构的构建

(一) 读取 line 数据

在 matlab 中读取 shp 文件数据，我们可以得到 $n \times 1$ *struct* 的数据其中 *n* 表示线的组数，其 line 数据包含以下数据： 下面我们根据以上信息初步生成邻接

表 **5-1 line** 字段

字段	含义
Geometry	拓扑结构
BoundingBox	边界框
X	线段各点的 x 坐标
Y	线段各点的 y 坐标
osm_id	路段 id
name	路段名称
highway	道路类别
waterway	水路类别
aerialway	空中路段类别
barrier	障碍物
man_made	人造物
z_order	高度
other_tags	其他标签

矩阵，其基本思想为遍历 *line* 数据结构的每个元素，将每个元素的 X,Y 项分别输入一个存储已知点集的数组中，我们称其为 x_{pos}, y_{pos}。然后对 *line* 中的每个元素 X,Y 相邻的坐标点进行距离计算并存储到邻接矩阵 distance 中。由于各个路段之间村子啊交点，但 line 数据结构并不会主动将这些交点标注出来，所以我们准备用快速排斥与跨立算法对线集中的交点进行求解。

(二) 求交点

在判断平米拿上两条线段是否相交时，我们采用快速排斥与跨立实验的算法，其基本思想为，首先通过快速排斥实验来对不满足条件的情况进行排除，然

后对剩余情况用跨立实验进行具体判断并求解。

快速排斥

假设二维平面上存在两个线段，$P_1, P_2 Q_1, Q_2$，以 P_1, P_2 组成的线段为对角线做矩形 R, 以 Q_1, Q_2 为对角线做矩形 T，当两个矩形不相交时两个线段肯定不相交，但两个线段不相交时，两个矩形可能会相交。

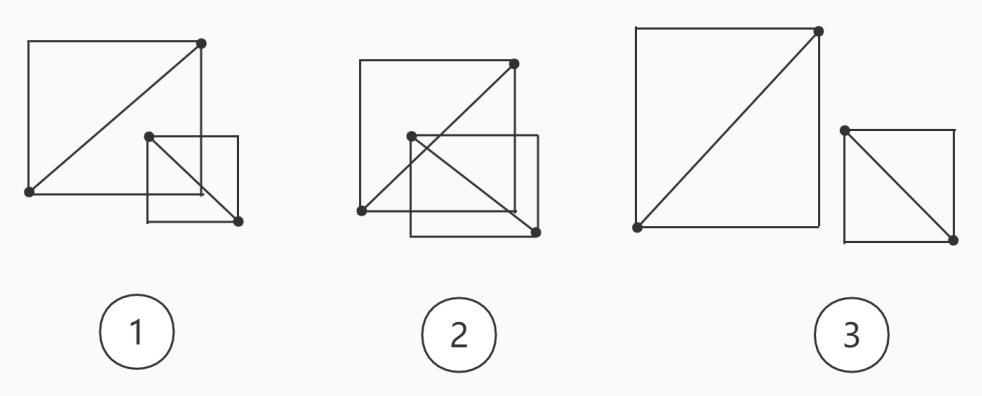

图 5-7　快速排斥与跨立实验

其中第一种情况通过了快速排斥实验但并未通过跨立实验，第二种情况通过了快速排斥实验，并且也通过了跨立实验，第三种情况并未通过快速排斥实验，直接可以得出结论，不存在交点。

在所以在快速排斥中的矩阵相交条件为：

$$min(p_1.x, p_2.x) \le max(q_1.x, q_2.x)\&\& \tag{5.1}$$

$$min(q_1.x, q_2.x) \le max(p_1.x, p_2.x)\&\& \tag{5.2}$$

$$min(p_1.y, p_2.y) \le max(q_1.y, q_2.y)\&\& \tag{5.3}$$

$$min(q_1.y, q_2.y) \le max(p_1.y, p_2.y); \tag{5.4}$$

如果通过了快速排斥实验，也不能直接得出结论二者有交点，而是在通过跨立实验得到结果。

跨立实验

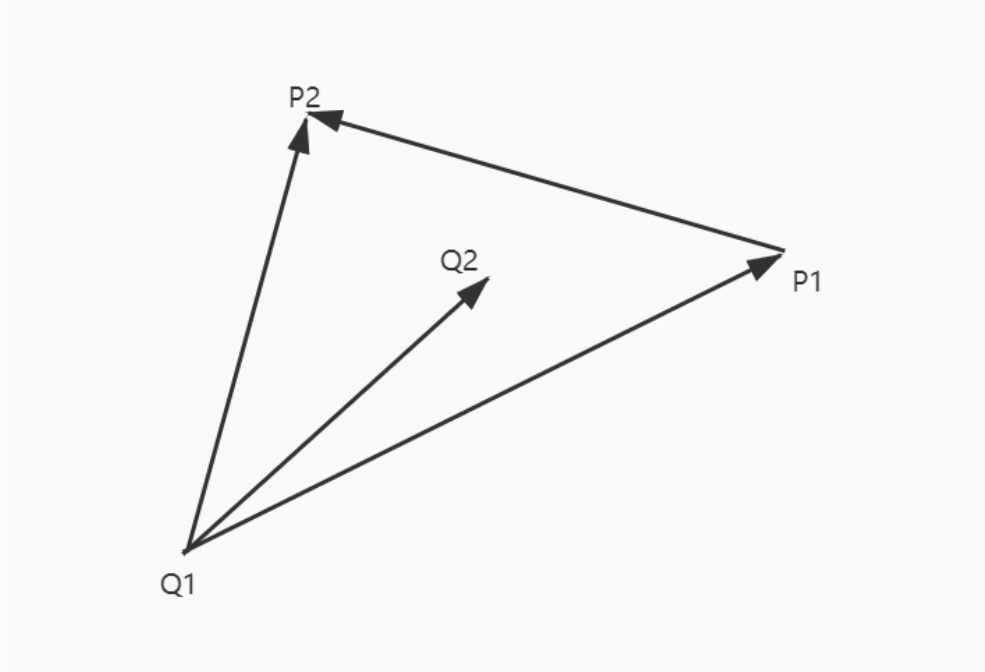

图 5-8　跨立实验

若线段 P_1P_2 跨过了线段 Q_1Q_2，那么 P_1P_2 就在 Q_1Q_2 的两侧，所以将：

$$((P_1 - Q_1) \times (Q_2 - Q_1)) * ((Q_2 - Q_1) \times (P_2 - Q_1)) > 0 \tag{5.5}$$

记为条件 1，同理 Q_1Q_2 分布在 P_1P_2 两侧的条件为：

$$((Q_1 - P_1) \times (P_2 - P_1)) * ((P_2 - P_1) \times (Q_2 - P_1)) > 0 \tag{5.6}$$

记为条件 2，满足以上两个条件两线段才会相交。

下面我们给出求解邻接矩阵算法的完整算法流程图，其中我们设 shp 文件读取后的数据为 *Map*，*Map* 的每个元素记为 *line*，*line* 中的元素构成如（5-1）所示。

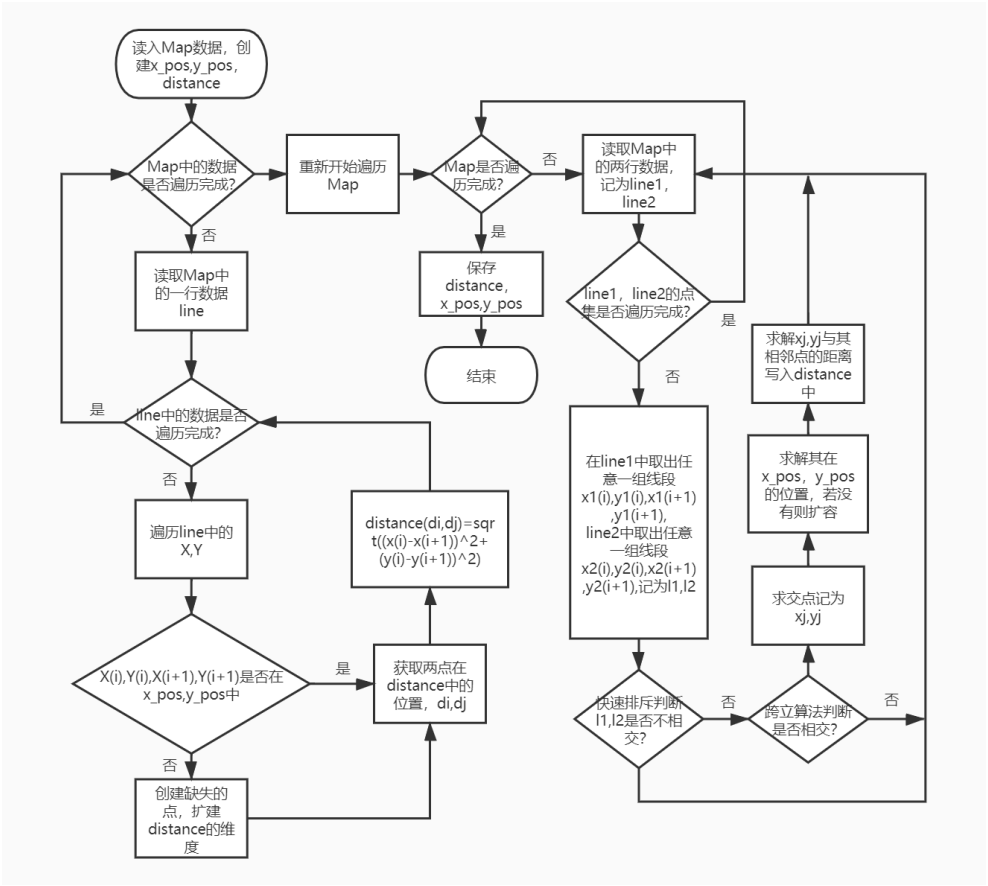

图 5-9　邻接矩阵构造算法

以上为数据的所有准备工作，下一章的项目实现将以此数据为依据对路径进行规划。

第六章　集成 Hopfield 网络与其他优化算法求解运输问题

一、　集成的必要性与可行性

我们前面介绍到 Hopfield 神经网络可以很好的解决 TSP 问题，但是他在解决优化问题时存在着容易求解得到非法解，或者容易收敛到局部最优解，前者随着 TSP 问题城市数量的增多，非法解出现的频率会逐渐增大。后者会导致最优解的质量得不到保证。所以本例采用了遗传算法与退火策略相结合对初始解进行优化在使用 Hopfield 网络进行优化的方法。采用以上方法是由于要克服 Hopfield 容易陷入局部最优解的缺点，所以我们采用遗传算法来对初始解进行优化并且也增加了其算法的全局搜索能力，但遗传算法局部寻优差并且收敛速度较慢，所以我们该算法引入了模拟退火算法的思想，保证了在遗传算法的一次迭代中的搜索空间内进行局部寻优，得到问题的最优解。

二、　集成的方法

(一)　单源最短路

在本例中由于每个地点之间并不直接通达，而是中间经过其他地点后相通，所以为了求解任意两个城市之间的路径我们采用了单源最短路算法-Dijkstra 算法：

该算法设置一个集合 P, 来记录在其求解的图中的最短路顶点，在算法开始时，会将原点放入该集合中，然后算法依次遍历并且将新遍历的点并入集合中, 然后更新最短路径，与其路径权值。在算法运行过程中会创建两个中间变量，*dist*[] 数组与 *path*[] 数组。其中 *dist* 数组记录原点到图中各个顶点的路径长度，*path* 表示从原点到某一顶点最短路径的前驱节点的索引值。算法结束后也以此变量来反推最短路径。下面算法中 V 表示图中的点集，下面给出其算法步骤：

$Step\ \ 1:$　数据初始化，P 集合将原点包含进去，$dist[i] = distance[0][i],\quad i = 123\cdots n-1$

$Step\ \ 2:$　在 $V-P$ 中找到某点 v_k，该点满足 $dist[j] = Min\{dist[i]|v_i \in V-P\}$，令 $S = S \cup k$。

Step 3： 求改从 v_0 出发到集合 $V-P$ 中任意一点 v_k 的距离，其更新公式为：*if* $dist[j]+distance[j][k]<dist[k]$ *then* $dist[k]=dist[j]+distance[j][k]$。

Step 4： 重复 *Step* 2 ~ *Step* 3 直到图中所有顶点都被包含到了 P 集合中。

(二) 主程序

本例的算法属于王银年在 2009 年提出的 3PM 交叉算子的模拟退火算法的变种，称为基于遗传模拟退火策略的 Hopfield 神经网络的优化算法，对适应度函数，遗传子程序到 Hopfield 程序的过渡部分，交叉，变异方式进行了添加与改动。由于本算法结合了多种算法所以为了方便叙述，我们将本算法简称为 GASA-Hopfield 算法, 下面给出算法的设计步骤：

Step 1： 输入初始数据，如邻接矩阵，路径矩阵，数据初始化，如种群，网络参数，迭代步长，交叉，变异概率，初始温度，降温幅度，种植温度，退火迭代链长。

Step 2： 进入遗传算法子程序，见（三），传入种群，邻接矩阵，网络参数，编译交叉概率。

Step 3： 将子程序搜索到的最优路径翻译为置换矩阵，并作为 Hopfield 网络的输入。

Step 4： 对神经网络输入 $U_{xi}(t)$ 进行初始化；即 $U_{xi}(t)=U_0'+\delta_{xi}$，其中 $U_0'=\frac{1}{2}U_0 ln(N-1)$，$\delta$ 为（-1，+1）区间的随机数。

Step 5： 动态方程计算 $\frac{dU_{xi}}{dt}$;

Step 6： 根据一阶欧拉法对 U_{xi} 即:

$$U_{xi}(t+1)=U_{xi}(t)+\frac{dU_{xi}(t)}{dt}\Delta T \tag{6.1}$$

Step 7： 使用 sigmoid 函数计算 $V_{xi}(t)$ 即：

$$V_{xi}(t)=\frac{1}{2}(1+tanh(\frac{U_{xi}(t)}{U_0})) \tag{6.2}$$

Step 8： 计算能量函数 E;

Step 9： 检查路径合法性，判断是否达到规定迭代次数，若没达到返回 *Step* 5。

Step 10： 输出每次迭代的能量函数，最优路径。以及最终的路径图，最短路程。

(三) 遗传算法子程序

Step 1： 对每个种群求解其对应的适应度。

Step 2： 对种群进行选择，交叉，变异操作。

Step 3： 对种族的每个染色体执行模拟退火算法子程序，见（四），传入一个染色体，此温度下链长，邻接矩阵，初始温度，与网络参数。

Step 4： 对模拟退火得到的优解保留到下一代中，即继续执行选择操作。

Step 5： 对此次迭代中得到的新解进行适应度计算。

Step 6： 判断当前温度是否达到终止温度，如果达到则退出子程序，否则对温度进行更新，转 *Step* 3，更新算法为：$T_{i+1} = q \times T_i$，其中 q 代表降温速率，以此来调节算法迭代次数。

(四) 模拟退火算法子程序

Step 1： 初始化各个变量，将每次迭代的结果存储下来以便筛选最优解。

Step 2： 对当前解 S_1 使用交换法进行扰动来产生新解 S_2。

Step 3： 通过 Metropolis 准则来判断是否接受新解，即新解优于旧解则接受，若不优于旧解则根据 $p = exp(\frac{E(S_{new}-E(x_{old}))}{KT})$ 概率接受新解。

Step 4： 在规定的链长下进行迭代，若执行次数达到规定次数则执行退火操作退出子程序。若没有达到规定次数则返回 *Step* 2 继续执行产生新解并筛选的过程。

(五) k-means 算法

对于 Hopfield 网络引入了混合智能算法，虽然提升了算法局部寻优与全局寻优能力，但是解的求解时间也会增加，随着城市数量的增加其求解时间也会出现较长的问题。为了优化算法的求解速率，我们将引入 K-means 算法来优化求解时间，其算法思想采用了“分而治之”的思想，将大规模 TSP 问题分解为小规模多个 TSP 问题，一方面降低了解的规模，减少了求解时间；另一方面避免了

大量无效的劣质解的搜索，优化了解的质量。多个小规模 TSP 问题解决后，采用类间连接将各个类的路径进行相连。下面我们给出 K-means 算法的流程图：

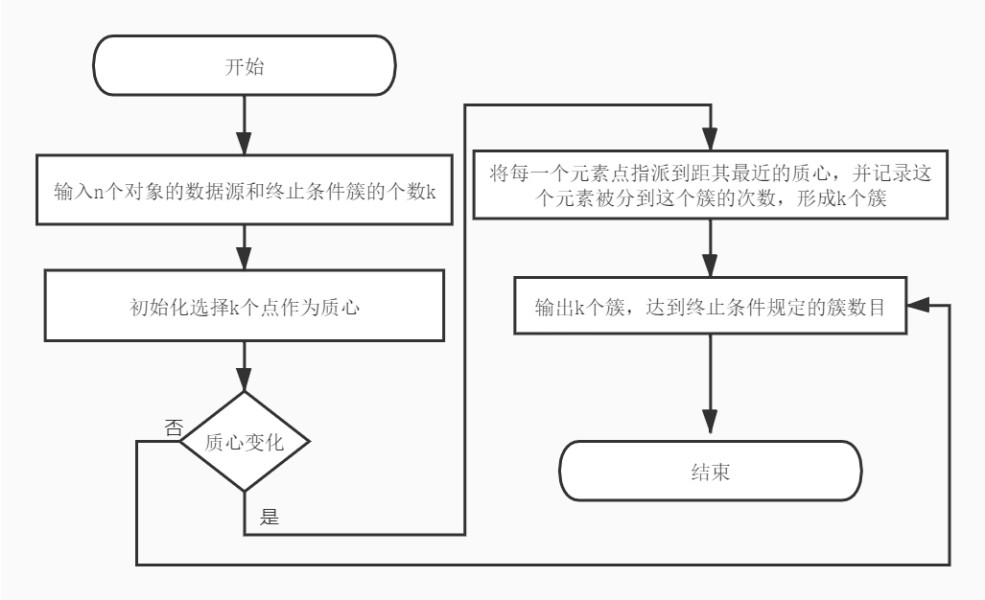

图 6-1　K-means 算法

其中对于最优 k 值的选取则采用轮廓系数的方法来确定：其轮廓系数的求解方法为：

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{6.3}$$

其中某一样本 i 到同簇的其余节点的平均距离被称为 a_i，我们称 a_i 为簇不相似度，即其值越小越属于该簇。一个簇中所有节点的簇不相似度称为该簇的不相似度，也是上式的 a_i。

某一样本 i 到其余某簇的所有样本的平均距离作为该样本与某簇的不相似度，样本 i 到簇 j 的不相似度称为 b_{ij}，样本 i 的簇间不相似度，定义为：$b_i = min\{b_{i1}, b_{i2}, \cdots b_{ik}\}$。其值越小样本越属于该簇。选取好最优分类值后我们开始准备簇间连接。

然后运行主程序对每个类的点进行性求解，然后采用类间连接算法对各个类进行相连，其主要思想为贪心思想，即以第一个簇作为起点簇，依次寻找离此簇最近的点所属于的簇即为簇链的下一个簇元素。

三、 针对运输问题集成 Hopfield 网络与其他优化算法进行求解

(一) 算法实现

在（二）中我们已经给出了算法流程图，接下来我们偏向于程序实现的角度具体的对改程序的步骤进行阐述。

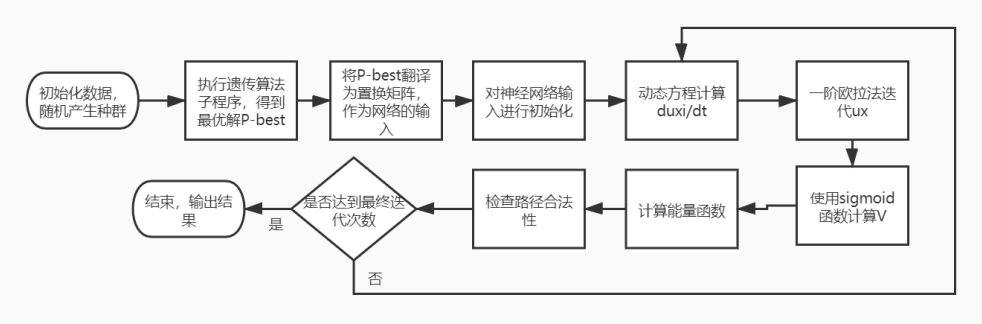

图 6-2 主程序

上图为主程序的实现，其中需要提前求出的变量为，该区域所有点集的邻接矩阵记为 *distance*，进行计算的点集的邻接矩阵 $distance_p$，存储所有点集坐标的 x_{pos}, y_{pos}，以及存储第 i 个点到第 j 个点的路径的 $path-group$，随机生成规定的种群后进入遗传算法子程序，即下图：

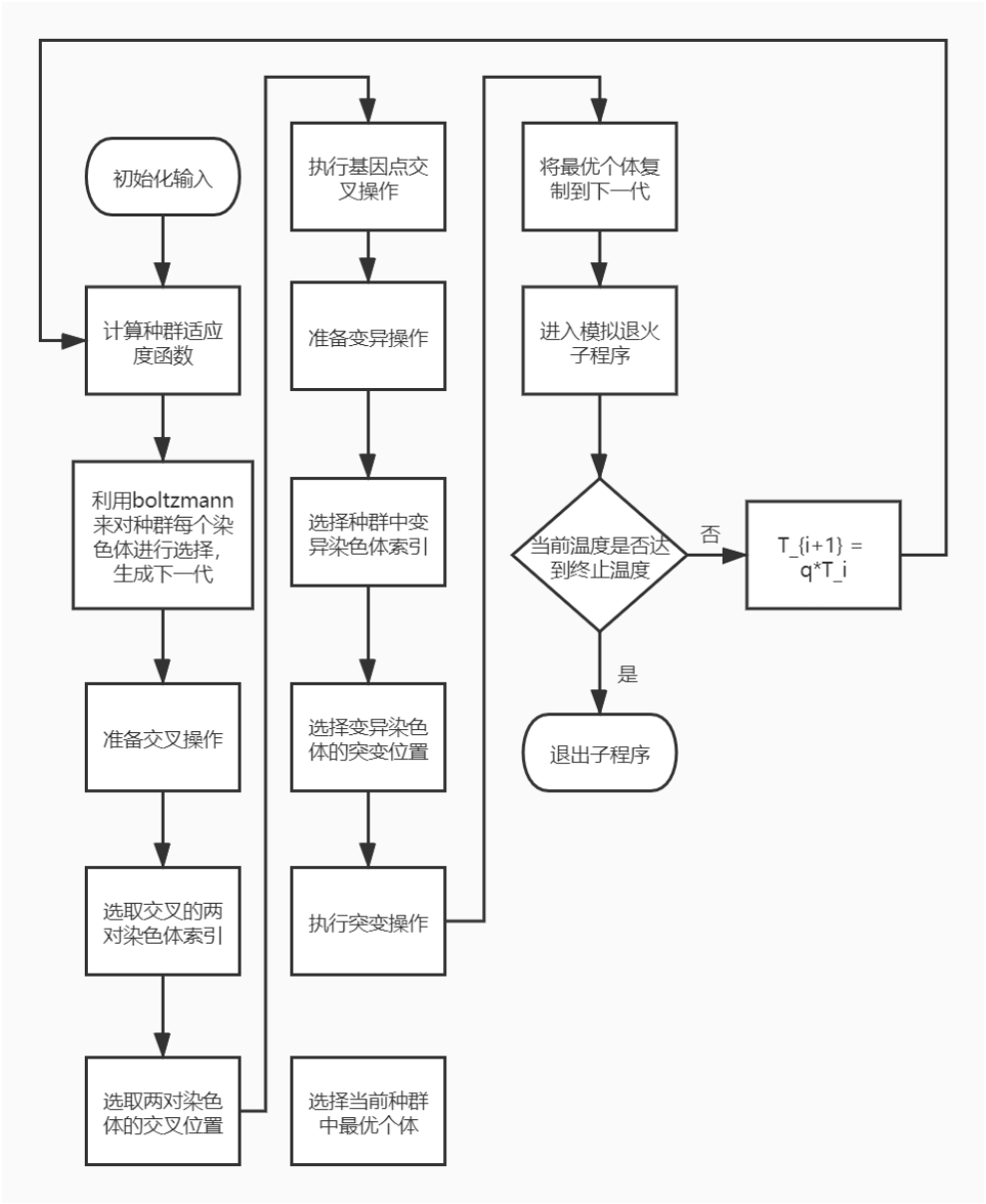

图 6-3　遗传算法子程序

在遗传算法子程序中我们的子程序终止条件为温度降为规定温度，并且子程序将种群，$distance_p$，变异率，交换率，网络指标作为初始输入。其中网络指标是由于需要计算能量函数来求解适应度函数。在每次迭代中程序依次执行选择，交叉，变异操作，然后进入模拟退火子程序，对当前代数的最优解附近的空间进行局部搜索，来弥补遗传算法的局部寻优的短板，经模拟退火子程序返回后，在将返回的最优解复制到下一代。然后根据迭代次数的要求判断是否继续迭代。下面我们介绍模拟退火子程序。

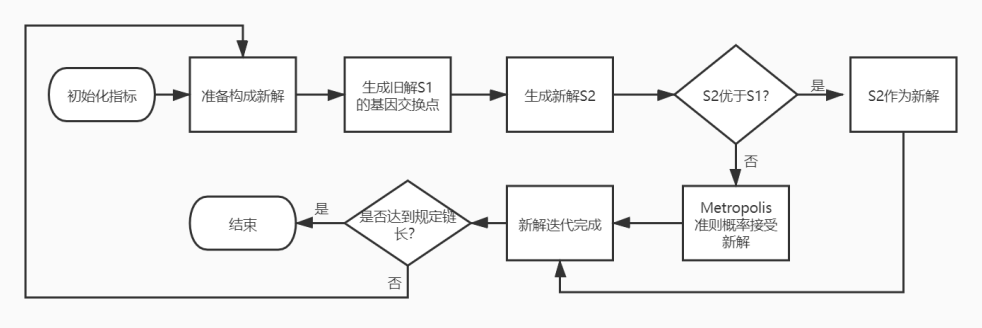

图 6-4　模拟退火子程序

模拟退火子程序的设计目的是为了弥补遗传算法局部寻优的不足，并且遗传算法也可以弥补模拟退火算法的全局寻优的短板。模拟退火程序的结束条件生成了某一温度固定次数的链长。首先对传入的初始解 S_1 进行随机扰动生成新解 S_2，若新解的质量比旧解的质量高则接受新解，否则根据 Metropolis 准则函数对新解进行概率接受。然后记录每次的解，迭代结束后将最优解作为其初始解空间附近寻找到的的最优解返回。

(二)　实例求解

表 **6-1**　程序参数

参数名称	大小
种群数量	1000
网络参数 A	1.5
网络参数 D	1
网络迭代步长	0.02
网络迭代次数	1500
交叉概率	65%
变异概率	50%
初始温度	500
终止温度	1.00E-04
降温速率	0.98
每个温度链长	50

我们根据以上算法对天津西青大学城附近的地区随机选取地点进行实验，以 50 个地点的 TSP 问题为例，我们首先运行 k-means 算法，并且根据轮廓系数选取最优的分类数，轮廓系数图如下：

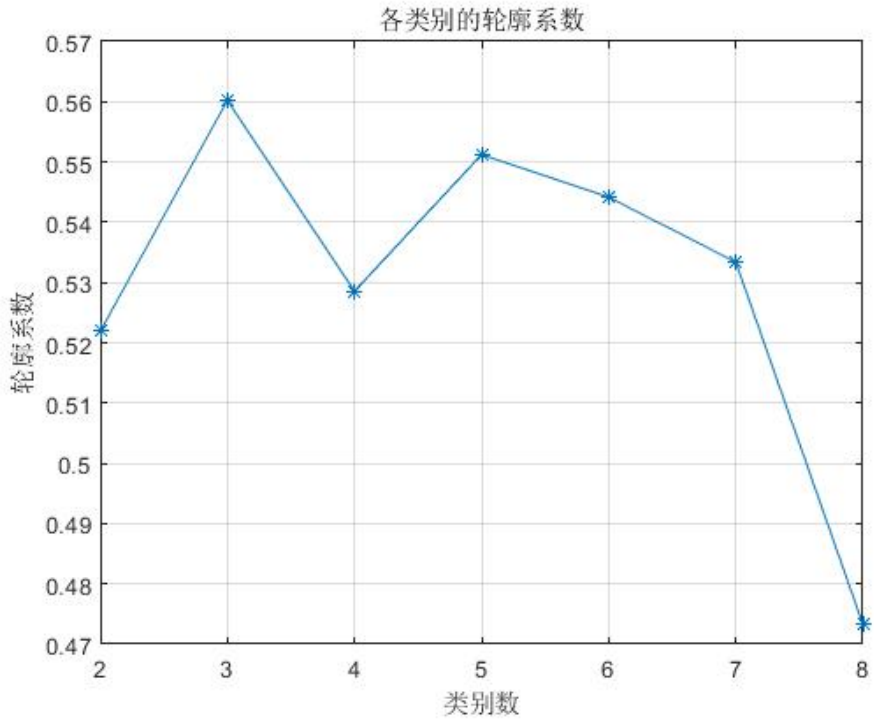

图 6-5　轮廓系数

则最优的分类数为 3，然后运行子程序，得到如下结果：

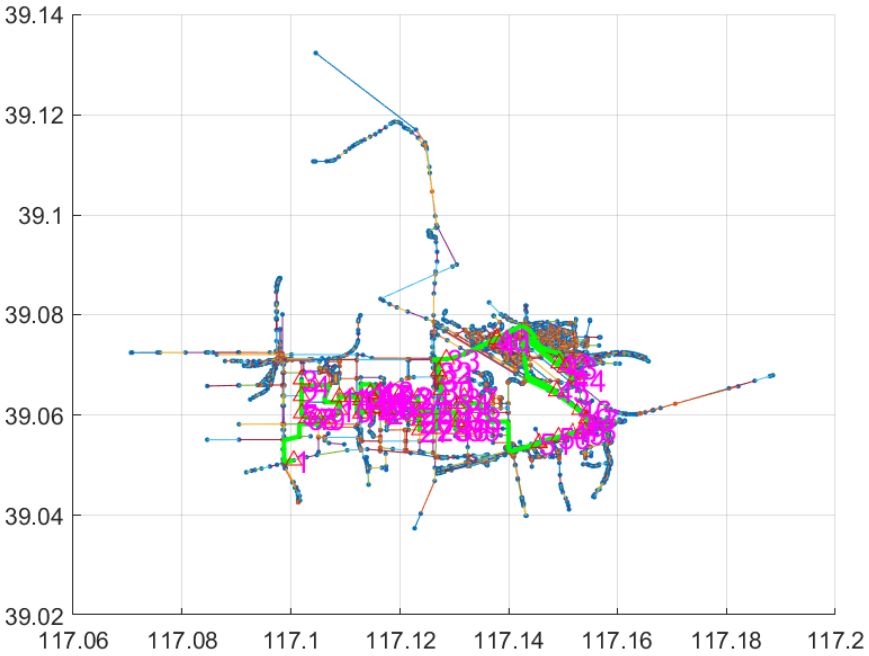

图 6-6　运输路线图

上面为城市路线图，正三角形标记了每一个需要送达的每一个地点，路径通过绿色路线来标注，由于地图点集过多，点集距离跨度较大，导致无法较为清晰的看到其运输路径，所以我们简化了了的路径图像如下：

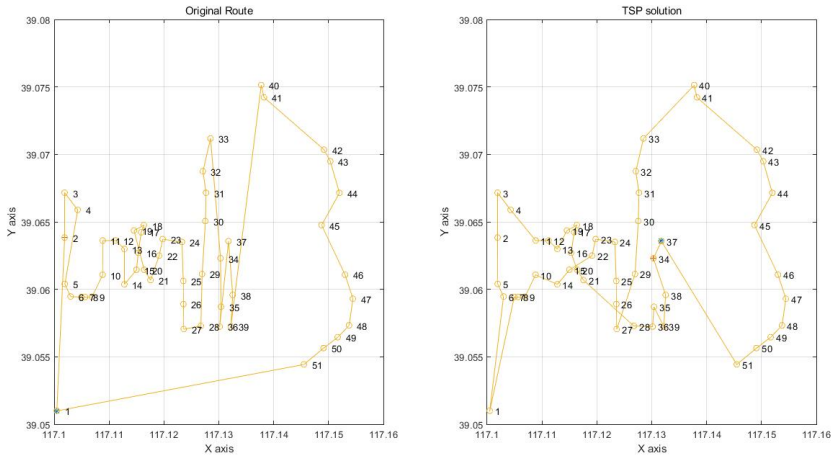

图 6-7　简化运输路线

上图为简化版的运输路线，从上面我们可以看出经过程序优化过的路线图相较于人工绘制的路线图路径更加简单，在配送时间上更加快速程序中 hopfield

的能量函数如下：

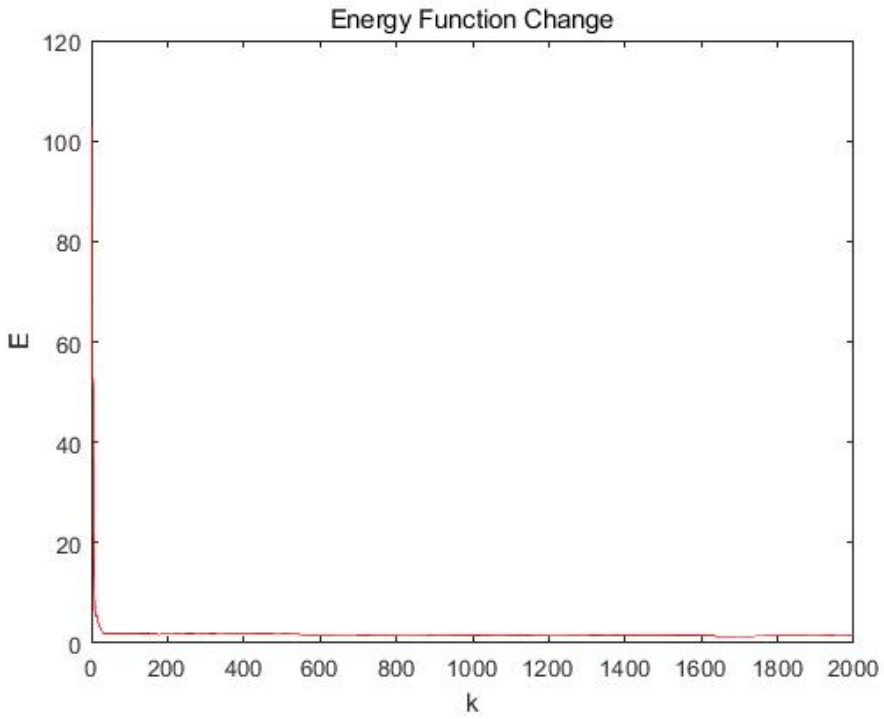

图 6-8　能量函数

此外为了对比此算法的优越性，将 GASA-hopfield 算法与常见的用于解决 TSP 问题的智能算法效率进行比较，结果如下：

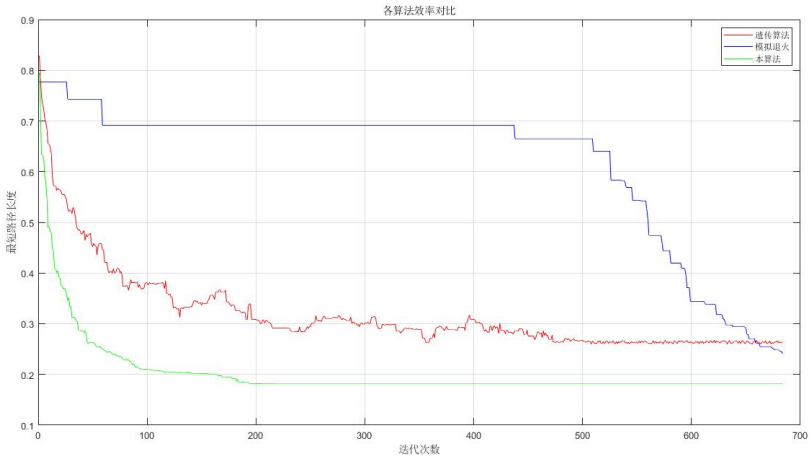

图 6-9　各算法效率对比图

可以看出 GASA-Hopfield 混合优化算法的收敛的比模拟退火与遗传更快，并且也更早的找到了最优解，有趣的是遗传算法由于参数设置的较差所以导致其

陷入了局部最优解，而模拟退火算法则到了 700 此迭代也没有彻底收敛，由此可见本算法的优越性。下面我们将比较的范围扩大到第四章介绍的所有解决 TSP 问题的算法，其对比维度为，求得的最优解，求得的平均解，与求解时间三个维度，运行程序，最后得到如下表格：

表 **6-2**　各算法运行效率

算法	最优解	平均解	运行时间
暴力穷举	0.18121	0.18121	1000s+
剪枝	0.18121	0.18121	1000s+
粒子群	0.25	0.27	119s
模拟退火	0.23	0.25	30s
遗传	0.27	0.28	729s
蚁群	0.1871	0.23	76s
动态规划	0.18121	0.18121	1000s+
GASA-Hopfield	0.18121	0.218	53s

我们可以看出暴力枚举，剪枝等确定性算法可以准确的算出路径最小值 0.18121 但是其时间已经远远超出了可以接受的范围，但智能算法等启发式算法可以在可接受时间内得到质量较高的解，其中蚁群算法在最优解与平均解上仅次于本算法，并且运行时间上也十分优秀，模拟退火算法在所有智能算法中运行时间是最优的但其解的质量没有其他解优秀，其粒子群与遗传，与 GASA-Hopfield 算法虽然运行时间较短，由此可以看出本例使用的 GASA-Hopfield 算法在时间与解的质量上是较为优秀的。

总结

本文根据提出的遗传模拟退火进行改进与 Hopfield 网络进行结合，相对于遗传算法增强了算法的局部搜索能力，避免算法时间过长接质量较差的缺点，相对于模拟退火算法，增强了算法的全局搜索能力，避免了算法陷入局部最优解的状态，相对于 Hopfield 神经网络通过调整神经元初始状态避免了城市数量过大时频繁出现的迭代出非法路径的问题。虽然相对于模拟退火与 Hopfield 神经网络，混合优化算法时间上较长，但是在实际应用中属于可以接受的范围，并且通过改进程序的实现方式还有较大的优化空间。

致谢

感谢国家

参 考 文 献

[1] 仇旺令. 遗传算法和 Hopfield 神经网络集成求解运输优化问题[D]. [出版地不详: 出版者不详], 2005.

[2] 范立南, 吕鹏. 基于改进遗传算法的校园外卖配送路径规划[J]. 物流科技, 2021, 44(01):14-19.

[3] 何黎明. 我国物流业 2020 年发展回顾与 2021 年展望[J]. 中国流通经济, 2021: 1-6.

[4] 胡大伟, 陈海妹, 梁一为, 等. 车辆与无人机混合编队的路径优化问题模型构建[J]. 长安大学学报 (自然科学版), 2021, 41(01):78-89.

[5] 兰兆青. Hopfield 神经网络在 TSP 问题中的应用[D]. [出版地不详: 出版者不详], 2008.

[6] 李博, 颜靖艺. 基于剪枝算法解决非对称 TSP 问题的算法研究[J]. 桂林航天工业学院学报, 2020, 25(04):430-436.

[7] 刘宁钟, 杨静宇. 遗传算法和 Hopfield 模型求解货郎担问题的比较和分析[J]. 计算机工程与应用, 2003(04):95-97.

[8] 马俊, 董良雄, 李军. 一种基于 K-means 改进蚁群算法的船舶航线设计方法[J]. 中国修船, 2020, 33(03):38-41.

[9] 庞燕, 罗华丽, 邢立宁, 等. 车辆路径优化问题及求解方法研究综述[J]. 控制理论与应用, 2019, 36(10):1573-1584.

[10] 帅训波, 马书南. 一种基于遗传 Hopfield 神经网络求解 TSP 问题的算法[J]. 微型机与应用, 2009, 28(21):7-9+15.

[11] 田贵超, 黎明, 韦雪洁. 旅行商问题 (TSP) 的几种求解方法[J]. 计算机仿真, 2006(08):153-157.

[12] 王铁, 胡泓. 基于 K-means 信息挥发速率动态调整的改进蚁群算法[J]. 机械与电子, 2020, 38(02):25-29.

[13] 王银年. 遗传算法的研究与应用[D]. [出版地不详: 出版者不详], 2009.

[14] 王颖. 基于 Hopfield 网络的 TSP 路径优化研究[J]. 赤峰学院学报 (自然科学版), 2015, 31(12):31-33.

[15] 吴高航. Hopfield 神经网络解 TSP 问题及能量函数参数分析[J]. 现代计算机(专业版), 2016(09):9-12.

[16] 闫玉莲. 一种改进的 Hopfield 神经网络对 TSP 问题的求解方法[J]. 闽南师范大学学报 (自然科学版), 2014, 27(03):37-43.

[17] 于兆敏. 基于遗传模拟退火策略的霍普菲尔德神经网络求解 TSP 问题[J]. 中国水运 (下半月), 2019, 19(04):89-91+94.

[18] 于兆敏. 基于遗传模拟退火策略的霍普菲尔德神经网络求解 TSP 问题[J]. 中国水运 (下半月), 2019, 19(04):89-91+94.

[19] 余一娇. 用 Hopfield 神经网络与遗传算法求解 TSP 问题的实验比较与分析[J]. 华中师范大学学报 (自然科学版), 2001(02):157-161.

[20] 张广林, 胡小梅, 柴剑飞, 等. 路径规划算法及其应用综述[J]. 现代机械, 2011(05):85-90.

[21] 张露. 基于改进遗传算法求解带时间窗车辆路径规划问题[J]. 中国物流与采购, 2020(14):66-69.

[22] 朱献文, 张敬. 基于遗传算法的 Hopfield 神经网络应用[J]. 信息与电脑 (理论版), 2011(20):166-167.

附录 A　常见问题

附录 B　联系我们