上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

AU339 网络智能优化

课程设计报告

姓名：	丁瑞琪、卢一帆、岳博
学校：	上海交通大学
学院：	电子信息与电气工程学院

目录

1. 邮递员问题

1.1. 问题分析

问题一中的邮递员问题，本质上类似于 TSP(Travel Salesman Problem)问题，也即是注重于能够访问到所有的顶点的同时保持代价尽量小。这和传统意义上的“中国邮递员问题”有很大的差异。

通常情况下，TSP 问题的模型是一个完全图，城市是图的顶点，道路是图的边，但在这个问题中的邮递员问题又和传统的 TSP 问题稍有差别，也即是本问题中的模型是一个非完全图，任意两个顶点间可能没有直接相连接的边。为了将这个问题转化为传统的比较方便求解的 TSP 问题，可以求出非完全图中不相邻的顶点之间的最短路径之和，作为两顶点间的距离，由此可以将非完全拓展为完全图。

1.2. 算法调研

TSP 问题已经被证明是组合优化中的一个 NP 难问题，在运筹学和理论计算机科学中非常重要。这个问题在 1930 年首次被形式化，并且是在最优化中研究最深入的问题之一。许多优化方法都用它作为一个基准。尽管问题在计算上很困难，但已经有了大量的启发式和精确方法，因此可以完全求解城市数量上万的实例，并且甚至能在误差 1%范围内估计上百万个城市的问题[1]。启发式算法中包含有遗传算法、模拟退火算法、蚁群算法等，都表现出极好的效果。精确方法有回溯法、分枝定界法、线性规划法和动态规划法等，这些算法能够在规模较小的时候能够精确地给出最优解，但是相比于启发式算法存在计算复杂度太高的，难以解决大规模问题。在算法层面上，王若愚等人在《基于强化学习的旅行商问题解构造方法》中基于迭代局部搜索(ILS)的启发式算法，尝试利用搜索过程中所积累的历史数据，通过强化学习挖掘有用信息，用于引导解的构造过程[2]。黄泽斌等人提出了一种基于 K-近邻分区的蚁群算法在 TSP 问题中的应用方法[3]。黄海松等人在《改进狼群算法求解旅行商问题》一文中同时在迭代过程中引入二次搜索来提高算法求解速度与精度,以实现在达到最大迭代次数前出现最优解[4]。硬件层面上，江建文等人首次尝试使用基于 GPU 的并行遗传算法来求解 TSP 问题[5]，大大提高了求解速度。应用层面上，原丕业等人在《基于蚁群算法的送餐最短路径问题求解研究》一文中，基于蚁群算法原理,求解送餐最短路径问题[6]。吴皓华等人研究了基于球面距离的旅行商问题及其在航海中的应用[7]。单钻头的运动可以看成是典型的 TSP 问题。TSP 可以用整数线性规划来形式化[8][9][10]。

1.3. 所用算法及仿真结果

经过对现有算法的调研，我们小组最后将问题的求解分为两步：先将题目中的给出的非完全图拓展为完全图，再在完全图的基础上进行启发式算法。拓展图的过程中我们使用 Dijkstra 算法求解出不相邻的节点间的最短路径距离之和来替代该两点间的距离。在启发式算法部分，我们同时尝试了遗传算法和模拟退火算法的仿真，并且同时进行了最终返回 1 号点或者不要求返回 1 号点的仿真。

最终得到的如果不返回 1 号点，可以优化出的最短路径长度为：13232，路径为：
1-4-12-18-19-13-5-6-20-21-14-7-2-3-11-17-16-10-9-8-15-22-23-28-33-38-33-34-29-

24-25-42-41-37-32-31-36-35-40-39-30-26-27

迭代过程中路径总长度变化情况及最终路径如下：

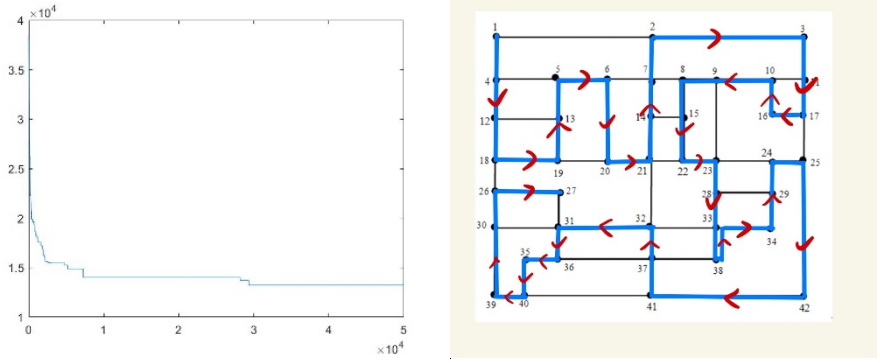


图 1-1：不返回起点情形下迭代曲线（左）与最终路径（右）

如果返回 1 号点，可以优化出的最短路径长度为：14080，路径为：
1-4-5-13-19-20-21-14-7-6-7-2-3-11-17-16-10-9-8-15-22-23-28-33-38-33-34-29-24-25-42-41-37-32-31-36-35-40-39-30-26-27-26-18-12-4-1

迭代过程中路径总长度变化情况如下：

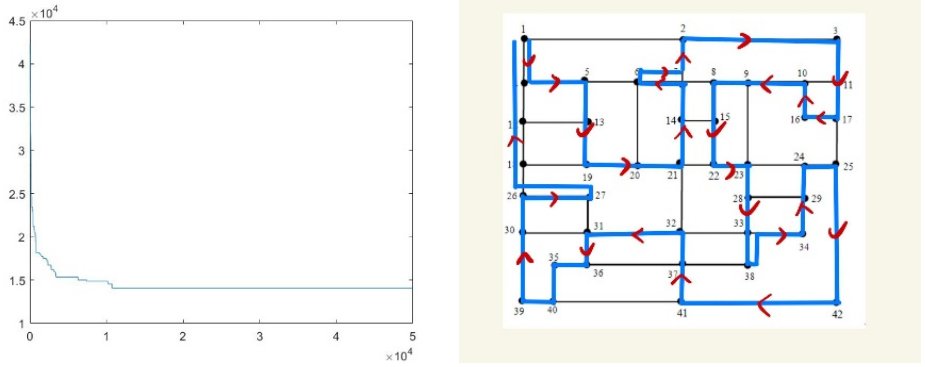


图 1-2：返回起点情形下迭代曲线（左）与最终路径（右）

实验代码见附录。

2. 最大流问题

2.1. 背景描述

定义：最大流问题指在一个单源点、单汇点的网络流中找到一条最大的流。最大流问题的实质是线性规划，数学定义如下。这里 s，v 分别为源点和汇点；E 是边的集合；V 是节点集合；c(u,v)为相邻节点 u 和 v 之间的边容量；f(u,v)为相邻节点 u 和 v 之间的实际流量。

$$\begin{aligned} &\text{maximize} && \sum_{v:(s,v)\in E} f(s,v) \\ &\text{subject to} \\ &&& \sum_{u:(u,v)\in E} f(u,v) = \sum_{w:(v,w)\in E} f(v,w) && \forall v \in V - \{s,t\} \\ &&& f(u,v) \le c(u,v) && \forall (u,v) \in E \\ &&& f(u,v) \ge 0 && \forall (u,v) \in E \end{aligned} \tag{2-1}$$

应用：最大流问题是网络的一个基本问题。许多系统都包含了流量问题。例如交通系统有车流量，金融系统有现金流，控制系统有信息流等。这些流问题主要是确定这类系统网络所能承受的最大流量以及如何达到这个最大流量。

关键概念：

1）增广路：在最大流问题中，我们每次找的一条从源点出发且可以增加汇点流量的路径，即在网络中找出一条源点到汇点的道路，并且求出这条路径所有边剩余容量的最小值 d，并在该路径上所有边的流量都加上这个 d，这个过程是增广。而增广路就是这条可以给汇点带来更多流量的路径。

2）增广路定理：当且仅当网络中不存在源点到汇点的增广路时，此时的流是从 源点到汇点的最大流。

3）残余网络：我们将已经消费的流量从每条边的流量中减去后得到的网络叫做残余网络。

4）最小割最大流定理：一个网络的最大流等于其最小割容量。

5）反向边：对每条有向边做一条反向边，反向边容量等于消耗（实际）流量的相反数。

2.2. 现有算法与研究概况

现有的最大流算法有 Ford-Fulkerson 方法（FF 方法）、Edmond-Karp 算法（EK 算法）和 Dinic 算法等。

FF 方法：基本流程为：1）DFS 搜索出一条增广路；2）在这条路径中所有的边容量减去这条增广路的流量，并为增广路增加流量相反数的反向边；3）返回第一步，如果没有增广路则得到最大流。FF 方法的时间复杂度为 O(F·E)，F 为最大流的值，E 是边的数量。

EK 算法：由于 FF 方法采用 DFS 搜索增广路，因此当存在噪声边时，且改边位置靠前时，会造成严重的效率低下。为此，EK 算法采用 BFS 搜索增广路，具体流程如下：1）使用 BFS 找到一条增广路；2）计算这条路的最小容量边，为汇点增加流量，并建立反向边，其容量为增加的流量；3）重复第一步，如果不能找到一条增广路则得到最大流；EK 算法的时间复杂度为 O(V·E^2)，V 是节点数量，E 是边的数量。根据 FF 方法和 EK 算法的时间复杂度，我们可以总结出这么一个结论: 1）当流网络中边少的时候，或者说是一个稀疏图（E<VlogV）时，我们可以选用 EK 算法进行最大流求解；2）当流网络中边多的时候，或者说

是一个稠密图（E > VlogV）时，可以选用 FF 方法求解[11]。

Dinic 算法：在 Dinic 算法中，我们使用 BFS 来检查是否有可能有更多的流量，并构造 level graph。在 level graph 中，我们为所有节点分配 level，一个节点的 level 是该节点到源的最短距离（以边的数量表示）。一旦构建了 level 图，我们就可以使用该 level 图发送多个流。Dinic 算法的时间复杂度为 O(E·V^2)这就是它比 EK 更好的原因：在 EK 方法中，我们仅发送通过 BFS 所找路径发送的流

表 1 -1：现有算法对比表

	时间复杂度	区别	适用场景
FF 方法	$O(F \cdot E)$	DFS 搜索增广路	稠密图
EK 算法	$O(V \cdot E^2)$	BFS 搜索增广路	稀疏图
Dinic 算法	$O(E \cdot V^2)$	Level Graph 搜索增广路	均可

我们小组调研了最大流问题的研究现状。在算法改进上，赵礼峰等人提出改进的最短增广链算法，改善了经典最短增广链算法中计算过程中为寻找最短增广链,需要根据原网络循环地构建剩余网络和剩余分层网络,步骤非常繁杂的问题[12]；邵丽萍等人提出了基于宽度优先的网络最大流求解算法，这种方法使用宽度优先搜索原理搜索增广链，避免了 Ford-Fulkerson 方法的标号过程,减少了反复重新寻找增广链的次数,为在大规模网络中快速获取最大流的求解提供了方便并提高了求解网络最大流的执行效率[13]；在具体应用方面，吴海燕等人提出了对城市道路网的路网容量理论和模型进行了研究,在国内外现有较成熟的路网容量模型研究的基础上,分析了各种模型的假设条件、模型特点以及模型应用范围;并分析了国内外现有的最大流算法及其缺陷,提出了改进的算法,并在 Matlab 中得到实现;最后,在中关村实例中进行了应用,效果良好[14]。

2.3. 所用算法

我们小组采用 Edmond-Karp 算法，算法的流程如下：

①使用 BFS 找到一条增广路；

②计算这条路的最小容量边，为汇点增加流量，并建立反向边，其容量为增加的流量；

③重复第一步，如果不能找到一条增广路则得到最大流。

2.4. 程序流程

见附录。

2.5. 仿真结果

通过以上算法及程序实现，最终求得该图从源点 1 到汇点 6 的最大流为 23，各边通过的实际流量如下图所示：

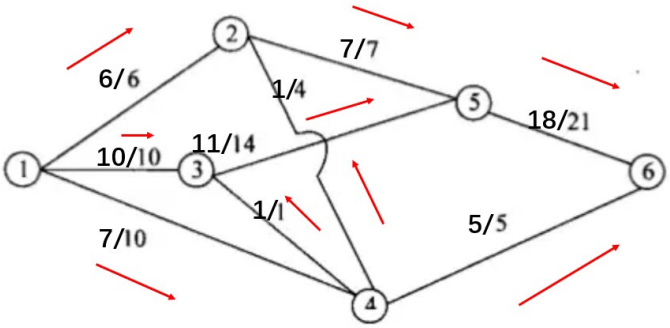


图 2-1：最大流问题仿真结果

3. 无线传感器网络的覆盖问题

3.1. 问题概述

无线传感器网络（Wireless Sensor Network，WSN）因其广泛应用业已成为网络问题中的研究热点。节点的覆盖范围和检测能力决定了数据采集与传输的可能性，节点的能耗与使用时长也影响着无线传感器网络的目标覆盖质量。本课题假设在给定的 400m*400m 的感知区域内，随机分布 100 个目标和 100 个感知节点，假定每个传感器的检测范围为 50m，每个传感器最多可检测到 5 个目标，每个目标在同一时间段内必须被 3 个传感器检测到。本题是典型的覆盖问题即在无线传感器网络中，在传感器节点个数和能力有限的情况下，设计智能优化算法。

我们对一些关键概念进行了进一步的明确，如下：

覆盖范围：以传感器为圆心，半径为 50m 的圆，圆内为覆盖范围。

检测能力：每个传感器能检测目标的个数。传感器检测目标数可以被设置为 0-5 之间的某一个整数，且传感器根据距离顺序，优先检测较近的目标。值得注意的是，设置检测目标数与传感器实际检测到的目标数并不一定相等，下文提到“检测目标数”均指前者。

调度策略：对每个传感器检测目标数的设置。

调度周期：一个调度周期内包括若干持续时长相同且固定的时间段（状态），每个状态均对应一个调度策略，即若干调度策略以一个调度周期为单位进行循环。

节点能耗：和传感器检测的目标个数正相关，这里我们认为检测的目标数为主要矛盾，暂不考虑检测的目标距离。

能耗均衡：我们认为一个周期内各传感器的能耗应尽可能均衡，以免出现短时间内，个别传感器因达到使用时长上限，而带来更换检修等费用。

环境：本课题中，传感器和目标的位置一旦给定，不再随时间变化。

本问题中由于目标的随机分布，可能会出现传感器资源冗余的情形，因此我们设计了一种基于遗传算法的策略优化算法，在给定传感器和目标分布下，通过求得的优化策略对各传感器的工作状态（要求检测的目标数）进行调度，在一个调度周期内，保证较多的目标被成功检测（被至少 3 个传感器检测）到的前提下，减少整体的能耗指标与使用时长指标，即在检测成功率和节点能耗及使用时长指标之间求得一个平衡点。此外，我们还设计了一种基于聚类方法的传感器放置策略，进一步优化网络的性能。

3.2. 研究现状

我们调研了无线传感器部署和节约能耗方面的相关研究。传统的 LEACH 算法，即低功耗自适应集簇分层型协议，以循环的方式随机选择簇头节点，将整个网络的能量负载平均分配到每个传感器节点中，从而达到降低网络能源消耗、提高网络整体生存时间的目的[15]；J. N. Al-Karaki 和 A. Gawanmeh 提出了一种动态规划方法，研究了具有覆盖度的广义无线传感器网络的连通性及其对路由的影响。对所提方案的性能评估表明，节点的可用性、传感器节点覆盖率和连通性都得到了充分的增强，从而使网络寿命最大化[16]；Christopher Zygowski 和 Arunita Jaekel 提出了三个混合整数线性规划（MILP）公式来指导移动节点的最优方式。目标是以能够在指定时间内实现所需覆盖级别的方式来路由移动节点。第一个公式最大化了区域覆盖率，而第二个和第三个公式将所需时间最小化。利用不同的网络规模、节点感知和移动能力对 MILPs 的性能进行了仿真评估。与文献中的启发式方法和现有方法的

比较清楚地表明，所提出的公式可以在合理的时间范围内获得优越的结果[17]。

3.3. 问题建模

3.3.1 调度策略

对于调度周期内的某个状态 j，传感器在该状态下的调度策略可以视为一串长为 100 的编码 X^j：

$$\begin{aligned} &X^j = \{x_1^j, x_2^j, ..., x_{100}^j\} \\ &x_i \in \{0,1,2,3,4,5\}, i \in \{1,2,...,100\} \end{aligned} \tag{3-1}$$

考虑一个具有 s 个状态的调度周期，整个周期的调度策略可视为一个长为 s*100 的编码 X：

$$X = \{X^1, X^2, ..., X^s\} \tag{3-2}$$

那么整个无线传感器网络覆盖问题可以视为一个整数规划问题：

$$\begin{aligned} &\max_X \ F(X \mid T, S) \\ &X = \{x_1^1, ..., x_{100}^1, x_1^2 ..., x_{100}^s\}, x_i^j \in \{0,1,2,3,4,5\} \end{aligned} \tag{3-3}$$

其中 F 为目标函数，T 为目标分布，S 为传感器分布。

3.3.2 性能指标与优化目标函数

优化目标函数的设置需要兼顾目标覆盖率，传感器能耗，以及能耗均衡。

考虑一个调度周期内的状态 j，被目标的覆盖情况可以用一个长为 100 的 0-1 编码表示：

$$D^j = \{d_1^j, d_2^j, ..., d_{100}^j\}, j \in \{1,2,...,s\} \tag{3-4}$$

其中：

$$d_i^j = \begin{cases} 1 & \text{状态 j 下，目标 i 被三个以上传感器检测到} \\ 0 & \text{其他} \end{cases}$$

则周期内平均目标覆盖率为：

$$d = \frac{1}{100s} \sum_{j=1}^{s} \sum_{i=1}^{100} d_1^j \tag{3-5}$$

根据 3.1 节所述，分析传感器能耗时，只考虑传感器检测目标的数量，对于周期调度策略 X，将周期平均能耗率定义为所有传感器在一个周期内平均检测数除以 5，计算式如下：

$$c = \frac{1}{500s} \sum_{x_i^j \in X} x_i^j \tag{3-6}$$

若设置所有传感器在各状态下均检测 5 个目标，则能耗率为 1。

考虑各传感器的能耗差异，我们采用周期内传感器检测目标总数的方差来反映这一指标，计算式如下，其中 i 代表传感器编号，j 代表所属状态：

$$v = \operatorname*{var}_i \left(\sum_{j=1}^{s} x_i^j \right) \tag{3-7}$$

当一个周期内所有传感器检测目标总数相同时，v=0。

基于以上三项指标，我们启发性地设置（3-3）中的优化目标函数为：

$$F_{l,m,n}=\frac{d^{l}}{c^{m}(1+v)^{n}+\xi} \tag{3-8}$$

其中，d 为平均目标覆盖率；c 为平均能耗率；v 为检测总数方差；ξ 为一个大于 0 的极小整数，防止分母出现 0 情况，实验中取 0.0001；l,m,n 为正数，反映对上述三个指标的偏好情况，例如取 l=1，m=n=0.5，目标函数可写为：

$$F_{1,0.5,0.5}=\frac{d}{\sqrt{c(1+v)}+\xi} \tag{3-9}$$

表示对平均目标覆盖率具有更强的偏好。

除了上述三项指标，我们还定义一个指标——能效比，作为后续策略评价的标准之一。能耗比被定义为平均目标覆盖率和平均能耗率之比，即：

$$q=\frac{d}{c} \tag{3-10}$$

能效比用于反映使用单位传感器资源得到的目标覆盖情况，是网络性能的体现。

3.4. 基于遗传算法的策略优化

3.4.1 优化算法设计

整数规划问题（3-3）是一个 NP 难问题[8]，只能使用启发性算法迭代求解。如 3.3.1 中所述，我们需要得到策略具有天然的编码属性，因此我们采用遗传算法对策略进行优化。

遗传算法（GA）最早在 20 世纪 70 年代提出，该算法根据大自然进化规律而设计提出，模拟达尔文生物进化论的自然选择过程，以及基因交叉变异等遗传学机理，搜索最优解。

在本课题中我们将周期策略 X 视为一个基因序列；使用（3-9）作为适应度函数；采用轮盘赌方法选取适应度较高的父本基因；交叉操作中，在每一个状态对应的基因段都选取一个交叉位点；变异操作中，同样对每个状态对应的基因段都选取一个变异位点。算法伪代码如下：

Algorithm 1 基于遗传算法的策略优化

Input: 迭代轮数N，种群大小P

```
初始化目标传感器分布;
初始化遗传算法参数设置;
随机初始化种群，最优基因;
最优适应度函数值=-∞ ;
for i in 1,2,...N do
    计算适应度函数;
    for i in 1,2,...P do;
        轮盘赌选择父本;
        父本交叉;
        子代变异;
    end for
    更新种群;
    更新最优基因;
end for
```

图 3-1 遗传算法伪代码

3.4.2 仿真设置

在目标和传感器均为随机分布的情况下，我们对上述算法进行仿真测试，设定周期内包含状态数为 3，遗传算法相关参数设置如下表：

表 3-1：遗传算法参数设置

参数	值
迭代轮数	2000
种群大小	1000
交叉概率	0.7
变异概率	0.1
基因长度	300

仿真软硬件环境为 i5-7300 CPU 2.50GHz、8GB 内存笔记本电脑及 MATLAB R2020a 软件。

仿真在不同传感器和目标随机分布情形下重复 10 次，在各次实验中，我们还将最高能耗策略（所有传感器在三个状态中均保持检测 5 个目标）作为对照。

仿真代码见附录。

3.4.3 仿真结果与分析

下面是其中一次的实验结果。该次实验中传感器和目标分布情况如下，图中三角形代表目标，空心圆圈代表传感器。

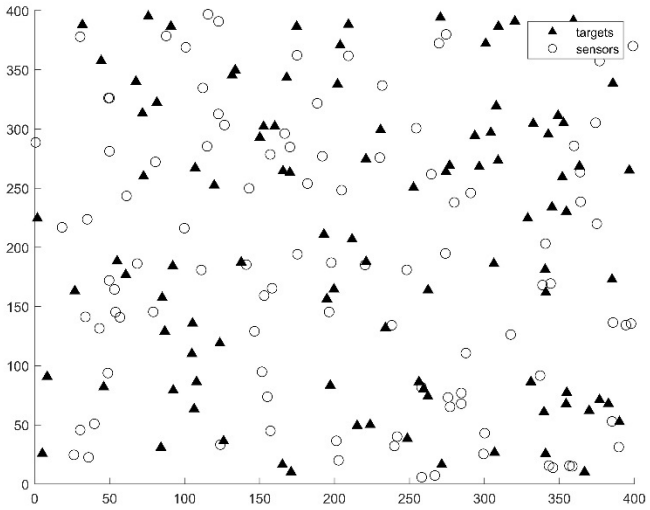


图 3-2：传感器和目标

该分布下算得最高能耗策略的平均覆盖率为 0.71，平均能耗率为 1，能耗比为 0.71。

遗传算法迭代曲线如下：

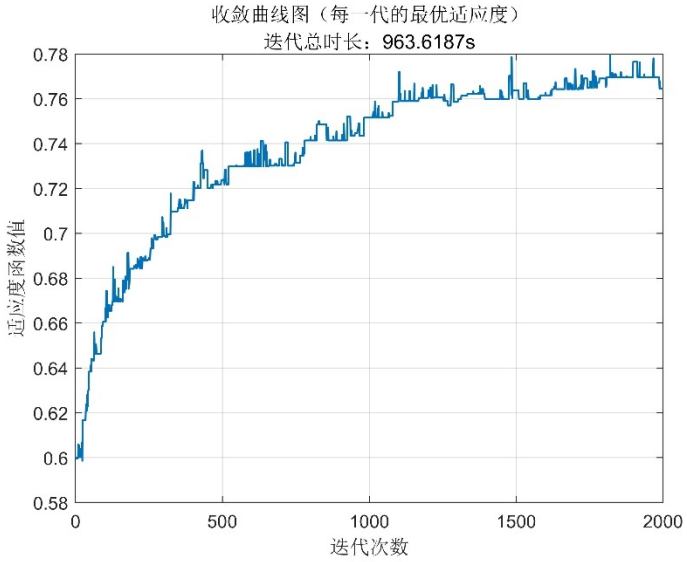
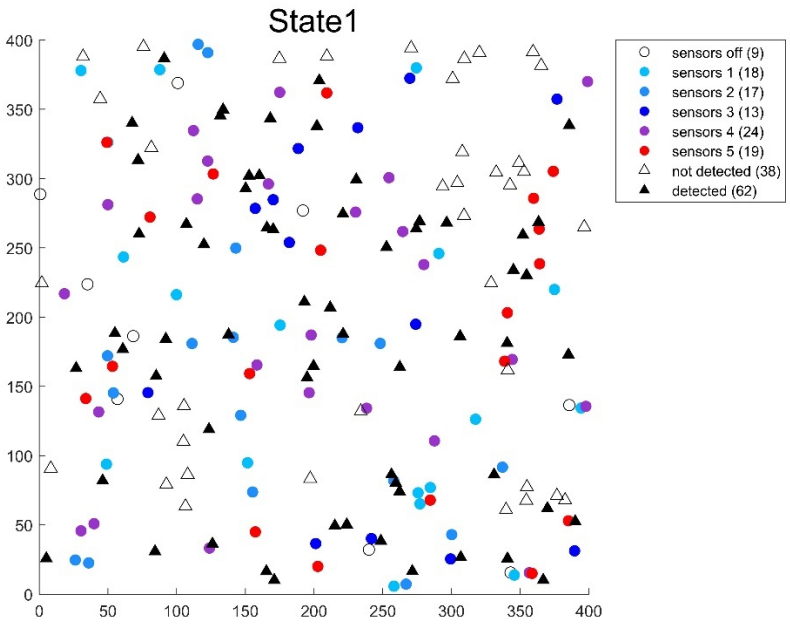


图 3-3：遗传算法迭代曲线

最终得到优化策略的 3 个状态如下图所示，各状态中不同检测目标数的传感器在图中以不同颜色的圆点显示；图中实心三角代表目标被检测到，空心三角代表目标未被检测到。



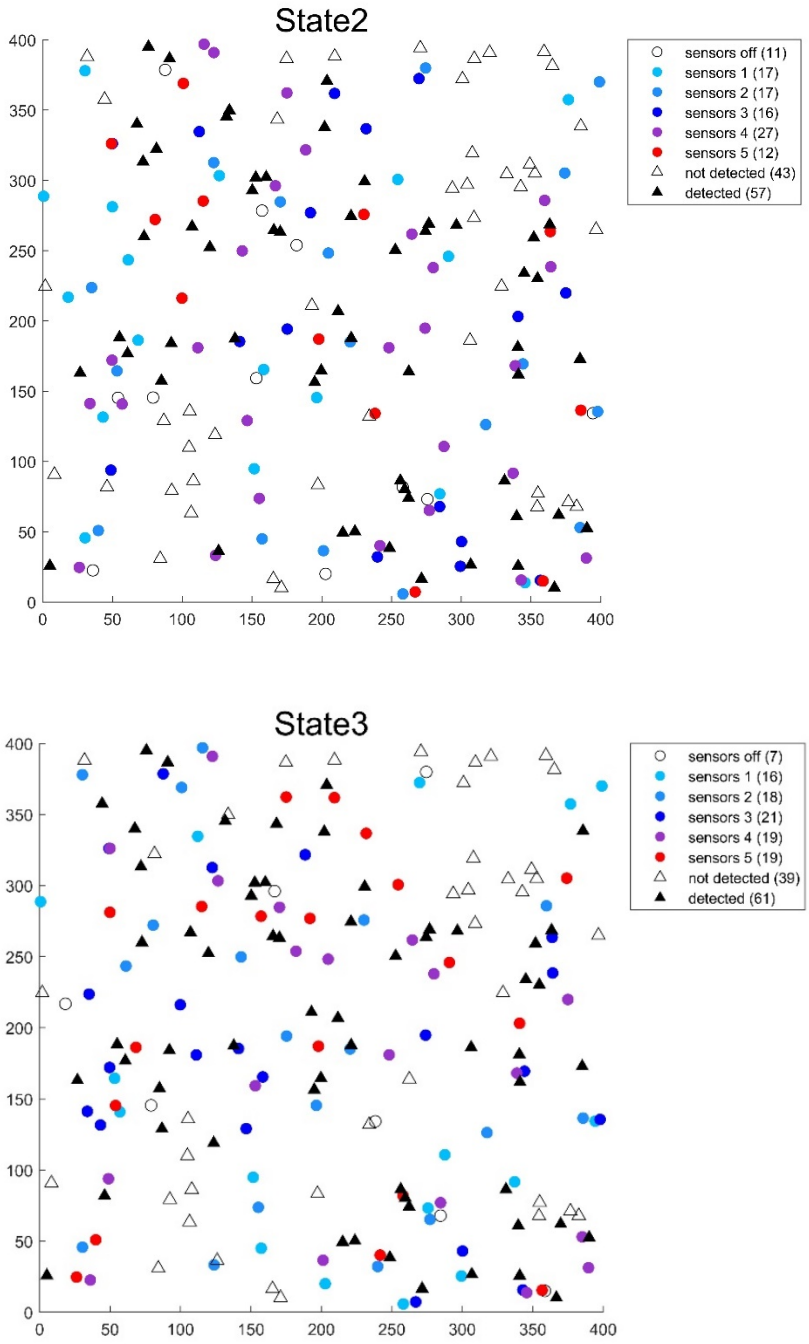


图 3-4：优化策略示意图

其中优化策略的平均感知率为 0.6，平均能耗率为 0.557，能效比为 1.078。

下表为 10 次仿真结果指标平均值（各次实验具体数据见附录 1）：

表 3-2：10 次仿真指标平均值对比

平均检测成功率		平均能耗率		平均能效比		优化耗时（秒）	
最高能耗	优化	最高能耗	优化	最高能耗	优化	最高能耗	优化
0.7060	0.5847	1.0000	**0.5649**	0.7060	**1.0349**	-	970.64

可以发现，经遗传算法得到的优化策略在使用约 56.5%的传感器资源的情况下，能达到最高能耗覆盖水平的 83%；相比最高能耗策略，优化策略在能效比指标上表现突出。使用最高能耗策略，会导致严重的传感器资源浪费；使用优化策略则能充分利用传感器资源冗余，在覆盖性能损失较小的情况下大大降低能耗。

3.5. 基于聚类的传感器放置策略

3.5.1 算法设计

考虑到随机放置传感器会造成传感器资源有较多冗余，以及部分目标不在任何传感器感知范围内的问题，我们设计了一种基于聚类的传感器放置策略，进一步优化传感器网络的性能。

对于随机分布的目标，我们采用以欧氏距离为度量的 k 均值聚类方法，将所有目标划分为 33 个类别，传感器则被放置在各类中心；各中心点传感器数量则根据该类包含目标数量，按照下表从上到下的优先级顺序设置：

表 3-3：传感器数量设置策略

目标数量 n	传感器数
3	3
>3	n-1
2	3
1	剩余平均分配

相比随机放置传感器，基于聚类的放置策略能够将传感器的位置分布特性纳入考虑，从而更加合理的地在空间上分配传感器资源，且能保留一定冗余。

3.5.2 仿真设置

本算法的仿真在仅目标随机分布的条件下进行，以基于目标聚类得到的传感器空间分布代替随机分布，随后使用 3.4 中所述的优化算法得到优化策略。仿真仍设定周期内包含状态数为 3，其他参数设置，软硬件环境等均与 3.4.1 相同。实验重复 10 次，同样以相同分布下的最高能耗策略作为对照。

仿真代码见附录。

3.5.3 仿真结果与分析

一次实验中聚类结果如下，图中不同颜色空心三角代表属于不同类的目标，相应颜色的实心圆点代表该类中心点，即传感器位置。

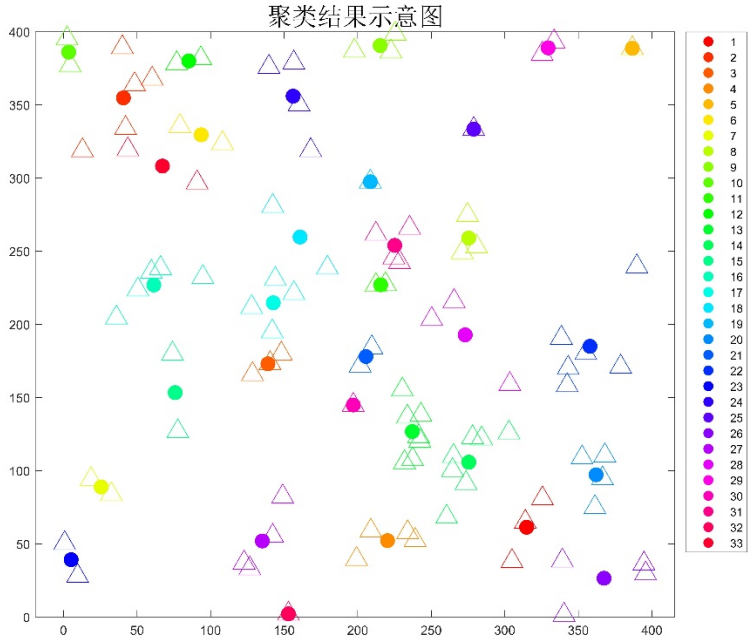


图 3-5：聚类结果

下表为 10 次仿真结果指标平均值（各次实验具体数据见附录 2）：

表 3-4：10 次仿真指标平均值对比

平均检测成功率		平均能耗率		平均能效比		优化耗时（秒）	
最高能耗	优化	最高能耗	优化	最高能耗	优化	最高能耗	优化
0.8290	0.6520	1.0000	**0.5770**	0.8290	**1.1300**	-	796.75

可以发现，基于聚类算法的传感器分布设置下，优化策略相比最高能耗策略仍有较大提升，且相比传感器随机分布时的相同算法有较大性能提升。此外，其他条件相同时，本算法中遗传算法优化耗时少于传感器随机分布的情形。

3.6. 小结

基于遗传算法的策略优化利用了传感器资源的冗余，可以视作在时间维度上对传感器资源进行调度；基于聚类算法的传感器放置策略则根据目标分布特点，可以视作在空间维度上对传感器资源进行调度。上文所述 4 种算法的性能对比情况如下图所示：

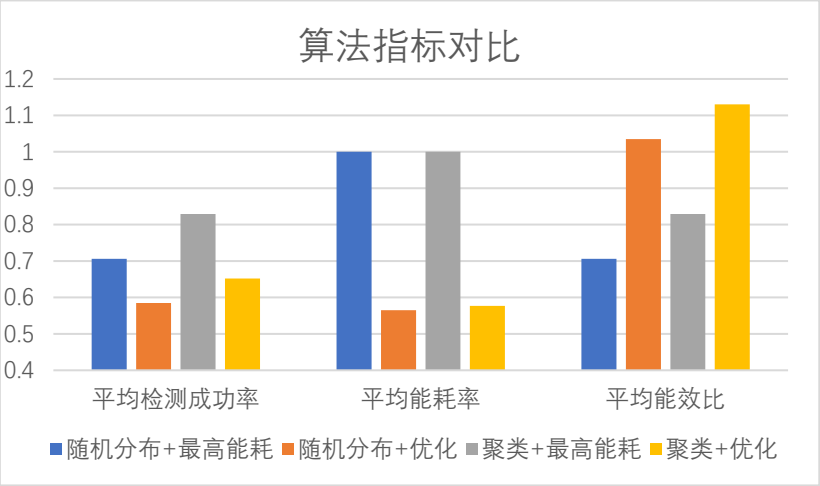


图 3-6：算法指标对比

可以发现无论是在时间层面上的调度策略还是空间层面上的调度策略，都能够提高网络的能效比；二者同时使用能带来进一步的提高。

参考文献

[1]: Applegate, D. L.; Bixby, R. M.; Chvátal, V.; Cook, W. J., The Traveling Salesman Problem, 2006, ISBN 0-691-12993-2.
[2]王若愚,陈勇全.基于强化学习的旅行商问题解构造方法[J/OL].计算机工程:1-11
[3]黄泽斌,黄钢忠,杨志鹄,姜春涛,黄颖欣.一种基于 K-近邻分区的蚁群算法在 TSP 问题中的应用研究[J].网络安全技术与应用,2019(11):50-52.
[4] 黄海松,任竹鹏,魏建安.改进狼群算法求解旅行商问题[J/OL].计算机应用研究,2019(12)
[5] 江建文. 基于 GPU 的并行遗传算法求解 TSP 问题[C].
[6]原丕业,张明,王岐昌,刘晓伟.基于蚁群算法的送餐最短路径问题求解研究[J].中国储运,2019(11):127-129.
[7] 吴皓华,曹茜.基于球面距离的旅行商问题及其应用[J].物流科技,2019,42(01):10-13.
[8]Beardwood, J.; Halton, J.H.; Hammersley, J.M., The Shortest Path Through Many Points, Proceedings of the Cambridge Philosophical Society, 1959, 55: 299–327, doi:10.1017/s0305004100034095.
[9]Bellman, R., Combinatorial Processes and Dynamic Programming, (编) Bellman, R., Hall, M., Jr. (eds.), Combinatorial Analysis, Proceedings of Symposia in Applied Mathematics 10,, American Mathematical Society: 217–249, 1960.
[10]Bellman, R., Dynamic Programming Treatment of the Travelling Salesman Problem, J. Assoc. Comput. Mach., 1962, 9: 61–63, doi:10.1145/321105.321111.
[11]一瓜.一瓜算法小册[EB／OL].https://www.desgard.com/algo,2020-07
[12]赵礼峰,纪亚宝.最大流问题的改进最短增广链算法[J].计算机技术与发展,Computer Technology and Development,编辑部邮箱,2016 年 08 期
[13]邵丽萍,赵礼峰.基于宽度优先的网络最大流求解算法[J].计算机技术与发展,2019,29(06):62-65.
[14]吴海燕,高进博,冷传才.路网容量最大流的一种改进算法[J].交通运输系统工程与信息,Journal of Transportation Systems Engineering and Information Technology,编辑部邮箱,2006 年 02 期
[15]Heinzelma , A. Chandrakasan and H. Balakrishnan, "Energy-Efficient Communication Protocol for Wireless Microsensor Networks, Proc. 33rd Hawaii Int'l. Conf. Sys. Sci., Jan. 2000.
[16]J. N. Al-Karaki and A. Gawanmeh, "The Optimal Deployment, Coverage, and Connectivity Problems in Wireless Sensor Networks: Revisited," in IEEE Access, vol. 5, pp.18051-18065, 2017, doi: 10.1109/ACCESS.2017.2740382.
[17]Christopher Zygowski and Arunita Jaekel. Optimal path planning strategies for monitoring coverage holes in Wireless Sensor Networks,Ad Hoc Networks,Volume 96,2020,101990,ISSN 1570-8705.
[18]王倩. 基于遗传算法的无线传感器网络覆盖控制研究[D].

附录

附录 1 传感器随机分布条件下仿真数据

实验编号	检测成功率		能耗率		能效比		耗时	
	最高能耗	优化	最高能耗	优化	最高能耗	优化	最高能耗	优化
1	0.72	0.5967	1	0.568	0.72	1.0505	-	965.77
2	0.75	0.5867	1	0.5767	0.75	1.0173	-	975.19
3	0.65	0.5767	1	0.5653	0.65	1.0202	-	950.97
4	0.73	0.6333	1	0.6073	0.73	1.0428	-	1016.70
5	0.7	0.5767	1	0.556	0.7	1.0372	-	989.89
6	0.69	0.5633	1	0.534	0.69	1.0549	-	946.22
7	0.73	0.5867	1	0.5633	0.73	1.0415	-	944.50
8	0.71	0.6	1	0.5567	0.71	1.0778	-	963.62
9	0.67	0.5333	1	0.5433	0.67	0.9816	-	983.60
10	0.71	0.5933	1	0.5787	0.71	1.0252	-	969.92
平均	0.706	0.58467	1	0.56493	0.706	1.0349	-	970.64

附录 2 基于聚类算法放置传感器仿真数据

实验编号	检测成功率		能耗率		能效比		耗时	
	最高能耗	优化	最高能耗	优化	最高能耗	优化	最高能耗	优化
1	0.82	0.65	1	0.5987	0.82	1.0857	-	775.00
2	0.84	0.6667	1	0.5827	0.84	1.1442	-	774.60
3	0.87	0.69	1	0.5773	0.87	1.1952	-	807.28
4	0.88	0.6833	1	0.5833	0.88	1.1714	-	848.65
5	0.77	0.5967	1	0.568	0.77	1.0505	-	736.08
6	0.73	0.5633	1	0.52	0.73	1.0833	-	776.82
7	0.85	0.6633	1	0.6153	0.85	1.0780	-	850.28
8	0.76	0.6067	1	0.5673	0.76	1.0695	-	742.58
9	0.86	0.6933	1	0.5713	0.86	1.2135	-	844.05
10	0.91	0.7067	1	0.586	0.91	1.2060	-	812.12
平均	0.829	0.652	1	0.57699	0.829	1.1300	-	796.75

附录 3 邮递员问题 Matlab 代码

①main.m 主程序

```
clear;
clc;
tStart = tic; % 算法计时器

load('new_A.mat');
```

```
G=graph(full_A);
%%%%%%%%%%%自定义参数%%%%%%%%%%%%%
rng(2);
maxGEN = 3000;
popSize = 1000; % 遗传算法种群大小
crossoverProbabilty = 0.7; %交叉概率
mutationProbabilty1 = 0.1; %变异概率
mutationProbabilty2 = 0.5;
mPChange=0.0002;
cPChange=0.0001;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
gbest = Inf;
pop = zeros(popSize, G.numnodes-1);
for i=1:popSize
    pop(i,:) = randperm(G.numnodes-1);
    pop(i,:) = pop(i,:)+1;
end
offspring = zeros(popSize,G.numnodes-1);
%保存每代的最小路径便于画图
minDists = zeros(maxGEN,1);
minPaths = zeros(maxGEN,G.numnodes-1);
% GA 算法
for  gen=1:maxGEN
%
%     % 计算适应度的值，即路径总距离
     [fval, sumDistance, minDist, maxPath] = fitness(full_A, pop);
%
    % 轮盘赌选择
    tournamentSize=4; %设置大小
    for k=1:popSize
        % 选择父代进行交叉
        tourPopDistances=zeros( tournamentSize,1);
        for i=1:tournamentSize
            randomRow = randi(popSize);
            tourPopDistances(i,1) = sumDistance(randomRow,1);
        end

        % 选择最好的，即距离最小的
        parent1  = min(tourPopDistances);
        [parent1X,parent1Y] = find(sumDistance==parent1,1, 'first');
        parent1Path = pop(parent1X(1,1),:);

```

```
%         if gen>4000
%             random = rand();
%             if random<=databaseProbability
%                 parent1Path = geneDatabase(randperm(databaseSize,1),:);

%             end
%         end

        for i=1:tournamentSize
            randomRow = randi(popSize);
            tourPopDistances(i,1) = sumDistance(randomRow,1);
        end
        parent2  = min(tourPopDistances);
        [parent2X,parent2Y] = find(sumDistance==parent2,1, 'first');
        parent2Path = pop(parent2X(1,1),:);

        subPath = crossover(parent1Path, parent2Path, crossoverProbabilty);%交叉

        if mod(gen,1000)<800 && mod(gen,1000)>500
            subPath = mutate(subPath, mutationProbabilty2);%变异
        else
            subPath = mutate(subPath, mutationProbabilty1);
        end

        offspring(k,:) = subPath(1,:);

    end
    minDists(gen,1) = minDist;
    fprintf('代数:%d    最短路径:%.2f \n', gen,minDist);
    % 更新
    pop = offspring;
    % 画出当前状态下的最短路径
    if minDist < gbest
        gbest = minDist;
        [minX,minY] = find(sumDistance==minDist,1, 'first');
        minPath=pop(minX,:);
%         paint(cities, pop, gbest, sumDistance,gen);
    end
    minPaths(gen,:) = minPath;

%     if gen>1000
%         mutationProbabilty=mutationProbabilty+mPChange;
%         crossoverProbabilty=crossoverProbabilty-cPChange;
```

```
%     end

end

figure
plot(minDists, 'MarkerFaceColor', 'red','LineWidth',1);
title('收敛曲线图（每一代的最短路径）');
set(gca,'ytick',10000:5000:50000);
ylabel('路径长度');
xlabel('迭代次数');
grid on
tEnd = toc(tStart);
fprintf('时间:%d 分  %f 秒.\n', floor(tEnd/60), rem(tEnd,60));
```

②fitness.m 计算适应度函数

```
function [ fitnessvar, sumDistances,minPath, maxPath ] = fitness( A, pop )
% 计算整个种群的适应度值
    [popSize, col] = size(pop);
    sumDistances = zeros(popSize,1);
    fitnessvar = zeros(popSize,1);
    for i=1:popSize
       sumDistances(i)= sumDistances(i)+A(1,pop(i,1));
       for j=1:col-1
          sumDistances(i) = sumDistances(i) + A(pop(i,j),pop(i,j+1));
       end
       sumDistances(i)= sumDistances(i)+A(pop(i,col),1);
    end
    minPath = min(sumDistances);
    maxPath = max(sumDistances);
    for i=1:length(sumDistances)
        fitnessvar(i,1)=(maxPath - sumDistances(i,1)+0.000001) / (maxPath-
minPath+0.00000001);
    end
end
```

③crossover.m 交叉

```
function [childPath] = crossover(parent1Path, parent2Path, prob)
% 交叉
    random = rand();
    if prob >= random
        [l, length] = size(parent1Path);
        childPath = zeros(l,length);
```

```
        setSize = floor(length/2) -1;
        offset = randi(setSize);
        for i=offset:setSize+offset-1
            childPath(1,i) = parent1Path(1,i);
        end
        iterator = i+1;
        j = iterator;
        while any(childPath == 0)
            if j > length
                j = 1;
            end

            if iterator > length
                iterator = 1;
            end
            if ~any(childPath == parent2Path(1,j))
                childPath(1,iterator) = parent2Path(1,j);
               iterator = iterator + 1;
            end
            j = j + 1;
        end
    else
        childPath = parent1Path;
    end
end
```

④mutate.m 变异

```
function [ mutatedPath ] = mutate( path, prob )
%对指定的路径利用指定的概率进行更新
    random = rand();
    if random <= prob
        [l,length] = size(path);
        index1 = randi(length);
        index2 = randi(length);
        %交换
        temp = path(l,index1);
        path(l,index1) = path(l,index2);
        path(l,index2)=temp;
    end
        mutatedPath = path;
end
```

⑤rebuild.m 根据 dijkstra 建完全图

```
clc;clear;
graph_mat=xlsread('TSP_graph.xls');
s=[graph_mat(:,2);graph_mat(:,6);graph_mat(:,10);graph_mat(:,14)];
t=[graph_mat(:,3);graph_mat(:,7);graph_mat(:,11);graph_mat(:,15)];
weight=[graph_mat(:,4);graph_mat(:,8);graph_mat(:,12);graph_mat(:,16)];
G=graph(s(1:62),t(1:62),weight(1:62));
A=adjacency(G,'weighted');
full_A=full(A);

for i=1:G.numnodes
    for j=i+1:G.numnodes
        [P,d]=shortestpath(G,i,j,'Method','positive');
        full_A(i,j)=d;
        full_A(j,i)=d;
    end
end
save('new_A.mat','full_A');
```

⑥paint.m 可视化

```
function [ output_args ] = paint( cities, pop, minPath, totalDistances,gen)

    gNumber=gen;
    [~, length] = size(cities);
    xDots = cities(1,:);
    yDots = cities(2,:);
    %figure(1);
    title('GA TSP');
    plot(xDots,yDots, 'p', 'MarkerSize', 14, 'MarkerFaceColor', 'blue');
    xlabel('经度');
    ylabel('纬度');
    axis equal
    hold on
    [minPathX,~] = find(totalDistances==minPath,1, 'first');
    bestPopPath = pop(minPathX, :);
    bestX = zeros(1,length);
    bestY = zeros(1,length);
    for j=1:length
       bestX(1,j) = cities(1,bestPopPath(1,j));
       bestY(1,j) = cities(2,bestPopPath(1,j));
    end
    title('GA TSP');
```

```
    plot(bestX(1,:),bestY(1,:), 'red', 'LineWidth', 1.25);
    legend('城市', '路径');
    axis equal
    grid on
    %text(5,0,sprintf('迭代次数: %d 总路径长度: %.2f',gNumber, minPath),'FontSize',10);
    drawnow
    hold off
end
```

附录 4 最大流问题 Matlab 代码

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%
% This program solves a sample max-flow-min-cut
% problem by the Ford-Fulkerson Algorithm
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function fordfulkerson
%% Input Network as Adjacency Matrix
    clc; clear;
    s = 1; t = 6; f = 0;
    cap = [  0  6 10 10  0  0;
             6  0  0  4  7  0;
            10  0  0  1 14  0;
            10  4  1  0  0  5;
             0  7 14  0  0 21;
             0  0  0  5 21  0  ];
    origin = cap;


    len = length(cap);

%% Ford-Fulkerson Algorithm
    while true
        p = findPath(cap);
        if p(1) == 0, break; end
        flow = max(max(cap));
        for j = 2:length(p)
            flow = min(flow,cap(p(j),p(j-1)));
```

```
        end
        for j = 2:length(p)
            a = p(j); b = p(j-1);
            cap(a,b) = cap(a,b) - flow;
            cap(b,a) = cap(b,a) + flow;
        end
        f = f + flow;
        disp(['Intermediate max flow is ' num2str(f)]);
        disp('Intermediate residual graph:');
        disp(cap);
        disp('Intermediate network flow:');
        disp((origin-cap));
    end
    disp(['Max flow is ' num2str(f)]);
    disp('Residual graph:');
    disp(cap);
    disp('Network flow:');
    disp((origin-cap));

%% Find an Augmenting Path
    function F = findPath(A)            % BFS (Breadth-first Search)
        q = zeros(1,len);               % queue
        pred = zeros(1,len);            % predecessor array
        front = 1; back = 2;
        pred(s) = s; q(front) = s;
        while front ~= back
            v = q(front);
            front = front + 1;
            for i = 1:len
                if pred(i) == 0 && A(v,i) > 0
                    q(back) = i;
                    back = back + 1;
                    pred(i) = v;
                end
            end
        end
        path = zeros(1,len);
        if pred(t) ~= 0
            i = t; c = 1;
            while pred(i) ~= i
                path(c) = i;
                c = c + 1;
                i = pred(i);
            end
```

```
            path(c) = s;
            path(c+1:len) = [];
        end
        F = path;
    end
end
```

附录 5 无限传感器网络覆盖问题 Matlab 代码

①main.m 主程序

```
for ind=1:10
clear;
clc;
tStart = tic; % 算法计时器

%%%%%%%%%%%环境设置与参数%%%%%%%%%%%%
%rng('default');
targets=400*rand(100,2);                % 目标
%sensors=400*rand(100,2);                % 传感器
sensors=distribute(targets);
distances=pdist2(sensors,targets);      % 距离矩阵 100*100，(i,j)返回传感器 i,目标 j 距离
stateNum=3;                             % 单位周期状态数
maxGEN = 2000;                          % 迭代轮数
popSize = 1000;                         % 遗传算法种群大小
geneLength = 100*stateNum;              % 序列长度
crossoverProbabilty = 0.7; %交叉概率
mutationProbabilty = 0.1; %变异概率
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

pop = zeros(popSize,geneLength);
for i=1:popSize
    % pop(i,:) = 5*(randi(2,1,geneLength)-1);          % 不考虑目标数，0-5 二值
    pop(i,:) = randi(6,1,geneLength)-1;                % 考虑目标数，0~5 六值
end
offspring = zeros(popSize,geneLength);

%保存每代的最优
optFVals = zeros(maxGEN,1);
optDetailedFVals = zeros(maxGEN,3);
% optGenes = zeros(maxGEN,geneLength);

```

```
%保存总体最优
bestGene = zeros(1,geneLength);
bestFVal = -Inf;
bestDFVal = zeros(1,3);

% GA 算法
for  gen=1:maxGEN

   % 计算适应度的值
    [fitnessValue,detailedFval] = fitness(distances, pop, stateNum);

    % 轮盘赌选择
    tournamentSize=4; %设置大小
    for k=1:popSize
        % 选择父代进行交叉
        parent1Gene=tournament_selection(pop,popSize,fitnessValue,tournamentSize);
        parent2Gene=tournament_selection(pop,popSize,fitnessValue,tournamentSize);

        subGene = crossover(parent1Gene, parent2Gene, crossoverProbabilty, stateNum);%交叉
        subGene = mutate(subGene, mutationProbabilty, stateNum);%变异

        offspring(k,:) = subGene(1,:);
    end

    [optFVal,optIndex]=max(fitnessValue);
    optFVals(gen,1)=optFVal;
    optDetailedFVals(gen,:)=detailedFval(optIndex,:);
    % optGenes(gen,:)=pop(optimIndex,:);
    fprintf('代数:%d \n 最优适应函数:%.2f 感知率:%.2f\n 能耗率:%.2f\n'...
            ,gen, optFVal,detailedFval(optIndex,1),detailedFval(optIndex,2));

    % 更新
    if optFVal > bestFVal
        bestFVal = optFVal;
        bestDFVal = optDetailedFVals(gen,:);
        bestGene = pop(optIndex,:);
    end
    pop = offspring;
end
tEnd = toc(tStart);
```

```
%%%%%%%%%%最高能耗%%%%%%%%%%
g_full=5*ones(1,100);
[fullFValue,fullDetailedFval]=fitness(distances,g_full,1);
fprintf('最高能耗 适应函数:%.2f 感知率:%.2f\n 能耗率:%.2f\n'...
            ,fullFValue,fullDetailedFval(1),fullDetailedFval(2));
%%%%%%%%%%最高能耗%%%%%%%%%%

folder=num2str(fix(datevec(now)));
folder=strrep(folder, ' ', '');
folder=['d_results/',folder];
mkdir(folder);
save([folder,'/targets.mat'],'targets')
save([folder,'/sensors.mat'],'sensors')
save([folder,'/optFVals.mat'],'optFVals')
save([folder,'/optDetailedFVals.mat'],'optDetailedFVals')
save([folder,'/bestGene.mat'],'bestGene')
save([folder,'/bestDFVal.mat'],'bestDFVal')
save([folder,'/tEnd.mat'],'tEnd')
save([folder,'/fullDetailedFval.mat'],'fullDetailedFval')

end
% figure
% plot(optimFVals, 'MarkerFaceColor', 'red','LineWidth',1);
% title('收敛曲线图（每一代的最短路径）');
% set(gca,'ytick',5000:5000:50000);
% ylabel('路径长度');
% xlabel('迭代次数');
% grid on
% tEnd = toc(tStart);
% fprintf('时间:%d 分  %f 秒.\n', floor(tEnd/60), rem(tEnd,60));
```

②fitness.m 计算适应度函数

```
function [fitnessValue,detailedFval] = fitness(distances, pop, stateNum)
%%%%%%%%%%%%参数说明%%%%%%%%%%%%%
% distances：传感器与目标之间的距离，100*100
% pop：种群，包含所有序列，种群大小*（100*stateNum）
% stateNum：单位周期传感器状态数
% fitnessValue：各序列适应度值，越大越优，种群大小*1
% detailedFval：各序列分项适应度值，种群大小*3，3 项包括：
%                周期内各状态目标成功检测率的平均值；
%                周期内所有传感器检测目标总数平均值；
%                周期内所有传感器检测目标总数的方差。
%                以上三项组合（加权求和、乘除...）得到 fitnessValue
```

```
% popSize = 1000 遗传算法种群大小
%%%%%%%%%%%%参数说明%%%%%%%%%%%%%
%% 一个周期求三个适应度值
    popSize=size(pop,1);
    detailedFval = zeros(popSize,3);
    avg_successful_rate = zeros(1,stateNum);
    fitnessValue = zeros(popSize,1);

    for i=1:popSize
        for j=1:stateNum
            targetDetected=check_targets(pop(i,(j*100-99):j*100),distances);
            avg_successful_rate(j) = mean2(targetDetected);     % 第 j 状态目标成功检测率;
        end
        detailedFval(i,1) = mean(avg_successful_rate);          % 周期内各状态目标成功检测率的平均值;
        detailedFval(i,2) = mean(pop(i,:))/5;                   % 周期内所有传感器检测目标总数平均值;
        pop_r=reshape(pop(i,:),stateNum,100);
        detailedFval(i,3) = var(sum(pop_r,1))/100;              % 周期内所有传感器检测目标总数的方差
        fitnessValue(i)=detailedFval(i,1)/sqrt(detailedFval(i,2)*(detailedFval(i,3)+1)+0.00001);
        %fitnessValue(i)=detailedFval(i,1);
    end
end
```

③check_targets.m 检查目标是否被 3 个传感器感知

```
function [successDetected]=check_targets(stateGene,distances)
    sensorConnected=zeros(100,1);
    for j = 1:100                                           % 每个传感器
        if stateGene(j)~=0
            [~, n] = sort(distances(j,:));                  % 排序，m 为值，n 为值对应的索引（传感器）
            for i=1:stateGene(j)
                if distances(j,n(i))<=50
                   sensorConnected(n(i))=sensorConnected(n(i))+1;
                end
            end
        end
    end
    successDetected=(sensorConnected>=3);
```

```
14. end
```

④crossover.m 交叉

```
1.  function [subGene]=crossover(parent1Gene, parent2Gene, crossoverProbabilty,s
    tateNum)
2.  % 根据两个父本基因和交叉概率求一个子代
3.      if rand() <= crossoverProbabilty
4.          [~, length] = size(parent1Gene);
5.          for i=1:stateNum
6.              crossPoint1 = randi(100)+(i-1)*100;
7.              subGene = zeros(1,length);
8.              subGene((i-1)*100+1:crossPoint1) = parent1Gene((i-
    1)*100+1:crossPoint1);
9.              subGene(crossPoint1+1:i*100) = parent2Gene(crossPoint1+1:i*100);
10.         end
11.     else
12.         subGene = parent1Gene;
13.     end
14. end
```

⑤mutate.m 变异

```
1.  function [subGene] = mutate(orgGene, mutationProbabilty, stateNum)
2.  % 根据概率变异
3.      if rand()<=mutationProbabilty
4.          subGene = orgGene;
5.          for i=1:stateNum
6.              randIndex1 = randi(100,5,1)+(i-1)*100;
7.              subGene(randIndex1) = randi(6,1,5)-1;
8.          end
9.      else
10.         subGene = orgGene;
11.     end
12. end
```

⑥tournament_selection.m 轮盘赌选择

```
1.  function [parentGene]=tournament_selection(pop,popSize,fitnessValue,tourname
    ntSize)
2.
3.      tourPopFVal=zeros(tournamentSize,1);
4.      tourIndex=zeros(tournamentSize,1);
5.      for i=1:tournamentSize
```

```
            tourIndex(i,1)=randi(popSize);
            tourPopFVal(i,1) = fitnessValue(tourIndex(i,1),1);
        end

        % 选择最好的，即适应度最大的
        [parentFVal,parentIndex]= max(tourPopFVal);
        parentGene = pop(tourIndex(parentIndex,1),:);
    end
```

⑦distirbute.m 基于聚类的传感器分布

```
function [sensorPoints] = distribute(targetPoints)
    % targetPoints 100*2
    [idx, centers] = kmeans(targetPoints,33);

    count = hist(idx,unique(idx));
    [channel, classes] = size(count);
    sensorPoints = [];
    sensorCount = 0;
    targetCount = 0;
    for i=1:classes
        if count(i) == 3
            sensorPoints = [sensorPoints; repmat(centers(i,:),3,1)];
            sensorCount = sensorCount + 3;
        end
        if count(i) >3
            repNum = count(i)-1;
            repCon = centers(i,:);
            sensorPoints = [sensorPoints; repmat(repCon,repNum,1)];
            sensorCount = sensorCount + count(i)-1;
        end
    end

    if isempty(find(count == 2,1)) > 0
        for i = find(count == 2)
            if sensorCount <= 97
                sensorPoints = [sensorPoints; repmat(centers(i,:),3,1)];
                sensorCount = sensorCount + 3;
            end
        end
    end
    index1 = find(count == 1);
    length1 = length(index1);
    if length1 > 0
```

```
        numIndex = 0;
        while sensorCount <100
            sensorPoints = [sensorPoints; centers(index1(mod(numIndex,length(index1))+1),:)];
            sensorCount = sensorCount +1;
            numIndex = numIndex + 1;
        end
    end
end
```