上海交通大学

SHANGHAI JIAO TONG UNIVERSITY

上海交通大学自动化系

网络智能优化课程项目报告

小组成员：陈聪、何正保、张定邦

授课老师：王易因, 何建平, 李贤伟

2020 年 12 月 17 日

目录

1 邮递员问题

1.1 问题描述

邮递员要找到最短的路径遍历所有的住宅区域，42 个住宅区域如下图 1 所示，每个城镇间由有权重的无向边相连。

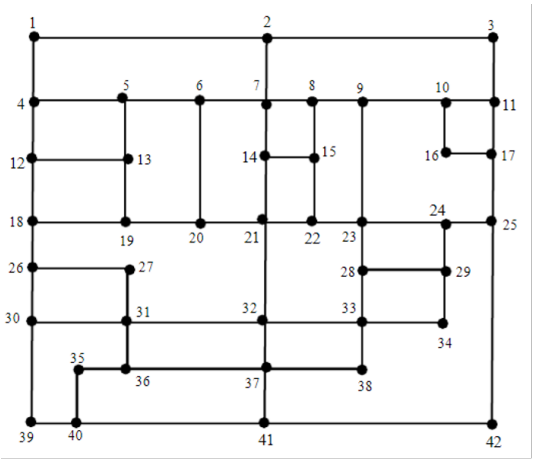

图 1: 住宅区域图

该问题可以抽象为一个旅行商（Traveling Salesman Problem, TSP）问题 [1]，即需要找到可以遍历全图所有节点的最短路径。

在后面的章节中，我们将针对问题一提出多个针对时间复杂度的优化思路。

1.2 问题初探——Dijkstra + 模拟退火

考虑到该问题是一个 TSP 问题，且典型的 TSP 问题为 NP 难问题，也即无法在多项式时间内找到最优解，因此我们便尝试从传统的 TSP 问题的解决方法中寻找思路。在最经典的旅行商问题中，问题常常被如下定义：

给定 N 个节点 $Node_*$，以及两两节点间的距离函数

$$Dist(Node_i, Node_j) = Distance(Node_i, Node_j) \tag{1}$$

优化目标为找到最优的路径 $path$，使得

$$path = \arg\min_{set(i)} \left(\sum_i Dist\left(Node_i, Node_{i+1}\right) \right) \ s.t. \ |set(i)| = N \tag{2}$$

而针对上述定义的 TSP 问题，常用的模拟退火算法如下算法 1 所示：

然而，直接对我们当前问题套用该框架并不容易。存在问题是，因为我们当前的住宅区图并不是全连接的，任意节点间的距离并不好得到。如下图 2 左边的连接情况，A、D 并不直接相连，此时想要得到 A、D 之间的距离会存在困难。

针对这一问题，我们的解决方法是，先对每个点两两之间通过 Dijkstra 算法得到两点间最短路径，该最短路径即视为该两点的最短距离，通过该方法，可以近似地将该图转化为一个全连接图，任意两点间的距离也可以比较容易地得到。

Algorithm 1 针对 TSP 的模拟退火算法

Input: 给定节点集 $Nodes$, 节点间距离函数 $Dis(Nodes_i, Nodes_j)$

Output: 最短遍历路径

```
初始化路径 path = {i}
for 预设循环次数 do
    随机交换当前记录 path 中的几个途经点
    计算当前的 cost 函数 ∑_i Dist(Node_i, Node_{i+1})
    if 当前的 cost 函数小于已经存储的 cost 最小值 then
        记录当前的 path
        记录当前的 cost 作为 cost 最小值
    else
        if 随机数符合模拟退火条件 then
            记录当前的 path
            记录当前的 cost 作为 cost 最小值
        else
            将 path 复原为途经点交换前的 path
return cost 最小路径 path
```

图 2: Dijkstra 算法将原图转化全连接图

算法训练结果如下图 3。图中横轴为迭代的轮数，每轮为 10000 次迭代。纵轴的 loss 即为最小化的目标。虽然该方法找到了不错的解，但存在两个显然的问题：

- 我们的问题中，任意两两城区间的距离函数 $Dist$ 没有良好的定义，因为城区之间并不两两任意可达，也即想要从一个城区到达另一个城区，可能会途径其他城区。这会导致距离函数所计算的并不是直线距离，直接套用会导致“明明经过了某个城镇，但却没有被记录，之后仍会遍历到该城镇”的问题。
- 现在的解决方案并没有考虑到问题中的送快递这一背景，只是单纯地套用了现有的基于图的做法，解决方案的可行性、易用性可结实性较差。

在后面的章节中，将针对这两个问题对现有的算法进行优化，以期得到更好的结果。

1.3 解法优化——Dijkstra 额外记录路径节点 + 模拟退火

针对前述的“明明经过了某个城镇，但却没有被记录，之后仍会遍历到该城镇”问题，我们对原算法进行了修改，在原本算法的基础上，在计算 cost 函数时，如果当前所走的路线已经走过了某些待遍历的节点，这些节点将会被标记并不再主动遍历，如算法 2。

对这一点进行优化后，模拟退火所进行的所有节点交换都将是有效交换，因此所需要探索的状态数量也会有一定程度的下降。

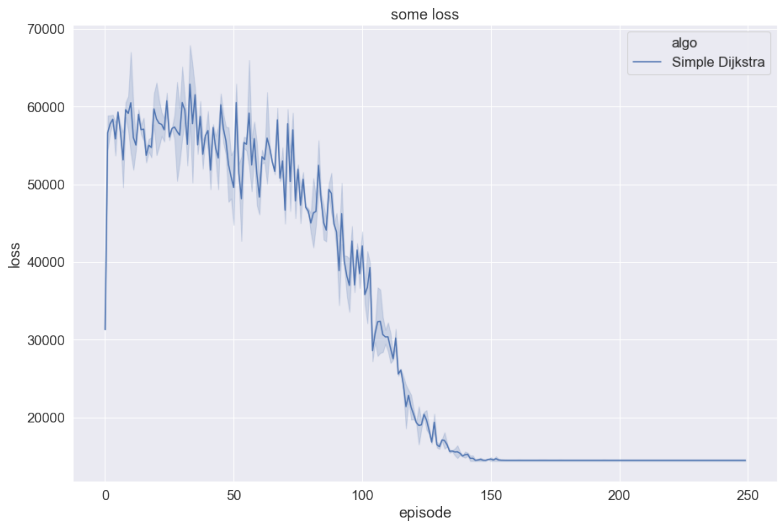

图 3: Dijkstra + 模拟退火实验结果

Algorithm 2 针对 Dijkstra 优化的模拟退火算法

Input: 给定节点集 $Nodes$, 节点间距离函数 $Dis(Nodes_i, Nodes_j)$

Output: 最短遍历路径

初始化路径 $Path = \{i\}$
for 预设循环次数 **do**
  随机交换当前记录 path 中的几个途经点
  for path 中每个途径边 **do**
    计算该途径边的 cost $Dist\,(Node_i, Node_{i+1})$
    从待遍历的节点中删去当前途径边途径的所有节点，并验证交换的有效性 ← 更改部分
  if 当前的 cost 函数小于已经存储的 cost 最小值 **then**
    记录当前的 path
    记录当前的 cost 作为 cost 最小值
  else
    if 随机数符合模拟退火条件 **then**
      记录当前的 path
      记录当前的 cost 作为 cost 最小值
    else
      将 path 复原为途经点交换前的 path
return cost 最小路径 path

进行该优化后的算法与原本简单套用 Dijkstra 算法的比较见下图 4。可以看到，在进行了优化后，算法的表现有了极为明显的提升，因为避免了很多额外的搜索，所以收敛也快了一些，在最后搜索到的解也相较而言更好了一些。

总地来说，优化对算法表现的提升是显著的，在效率、表现这两个启发式算法比较关注的维度上，都取得了更好的效果。

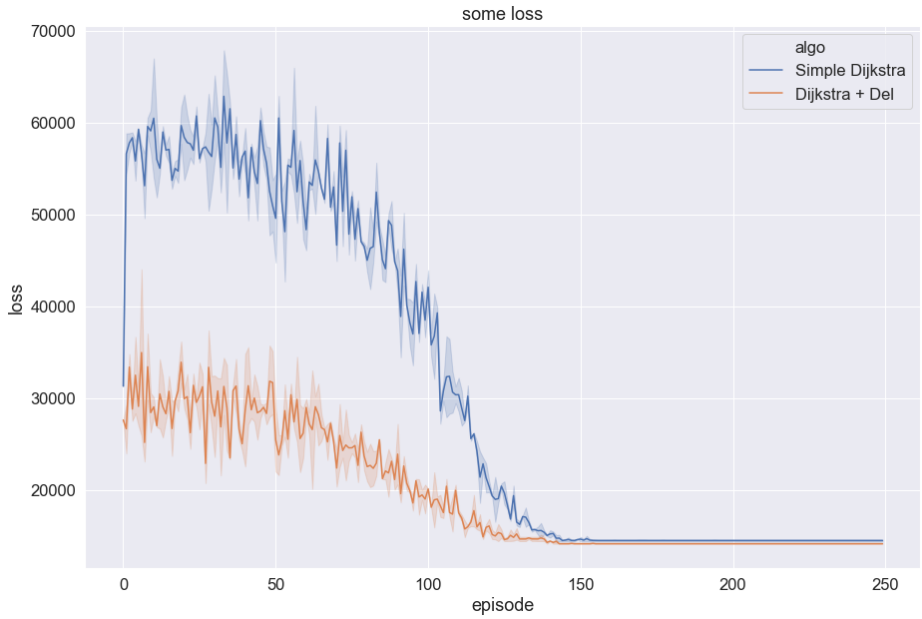

图 4: Dijkstra + 优化后模拟退火实验结果

1.4 结合实际问题的优化方法——自适应最短路径搜索算法

在之前的方法中，我们都没有考虑到当前所要解决的问题是一个配送任务，而对于人类而言，如果打算配送，肯定会分区块地进行配送，而不是对全区域进行随机探索配送。这样的策略有如下几个优势：

- 可以极大地降低搜索的空间大小，提升算法效率。
- 策略更符合人的直觉，而不是进行非直觉的随机搜索。

为了进行“分区域配送”，首先需要解决的就是区域划分问题，在这个问题上，我们有两个方案可以选择：根据当前拿到的图进行手动划分，或者提出一个自适应的算法，进而更具有普适性地解决这个问题。经过思考我们最终选择使用经典的图社团划分算法 Louvain [2] 算法来进行图的分割。接下来先对 Louvain 算法做简单的介绍。

1.4.1 Louvain 算法

Louvain 算法是一种基于图数据的社区发现算法，算法的优化目标为最大化整个数据的模块度，模块度的计算如下：

$$Q = \frac{1}{2m} * \sum_{ij} \left[A_{i,j} - \frac{k_i * k_j}{2m} \right] * \delta\left(C_i, C_j\right) \tag{3}$$

其中 m 为图中边的总数量，k_i 表示所有指向节点 i 的连边权重之和，k_j 同理。$A_{i,j}$ 表示节点 i，j 之间的连边权重。模块度这一指标通常用于衡量当前对于一个图的社团划分是否合理，社团划分越合理，模块度越高。从直觉上来说，一个社团内部的节点应该具有更好的连通性，而这也符合我们对于“属于同一个配送区域”的直觉理解，因为连通性较好的区域理应被统一配送。在下图中给出了通过 Louvain 算法进行社团划分的一个例子，可以看到，连通性更好的点集会被分到同一社区，而如果按照这样的社区划分进行配送，显然是非常符合人类直觉的。

因为 Louvain 算法本身并不复杂，在此就不做算法的细节介绍了。但值得一提的是，很容易证明的是，Louvain 算法的复杂度为 N^2，而对图进行划分后再进行搜索，复杂度的降低是指数级的。因此先进行社团划

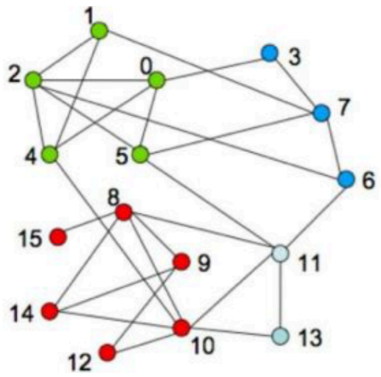

图 5: Louvain 算法社团划分结果例图

分再进行搜索，相当于用一个多项式时间的算法对非多项式算法进行优化。但同时也需要注意的是，复杂度的降低不是无代价的。事先进行社团划分必然会导致会存在一些可能的最优情况不能被探索到，但对于一个 NP 难问题，我们认为这种 trade-off 是可以接受的，因为对于此类问题，人类想要的往往只是“较优解”。

1.4.2 算法与实验结果

加入 Louvain 算法的优化后，算法如算法 3 所示。

Algorithm 3 自适应最短路径搜索算法 (Adaptive Shortest Path Searching Algorithm)

Input: 给定节点集 $Nodes$, 节点间距离函数 $Dis(Nodes_i, Nodes_j)$

Output: 最短遍历路径

```
使用 Louvain 算法对图进行社团划分
初始化路径 path = 所有社团依次按字母大小拼接
for 预设循环次数 do
    if 随机数小于 5% then
        随机交换两个社团的所有途经点
    else
        随机交换当前记录 path 中某个社团内部的几个途经点
    for path 中每个途径边 do
        计算该途径边的 cost = Dist(Node_i, Node_{i+1})
        从待遍历的节点中删去当前途径边途径的所有节点，并验证交换的有效性
    if 当前的 cost 函数小于已经存储的 cost 最小值 then
        记录当前的 path
        记录当前的 cost 作为 cost 最小值
    else
        以一衰减的概率接受次优解
return cost 最小路径 path
```

因为 Louvain 算法可以对有权、无权（只考虑度）进行社团划分，因此我们对该问题的两种算法分别进行了尝试，尝试得到的曲线如下图 6 所示。从图中可以看出，进行社团划分后的算法的 loss 相较其他方法，可以始终保持在一个比较低的位置，这是符合我们的期望和预期的。此外，实验可以发现，无权 Louvain 算法

进行划分在该任务中表现更好一些。

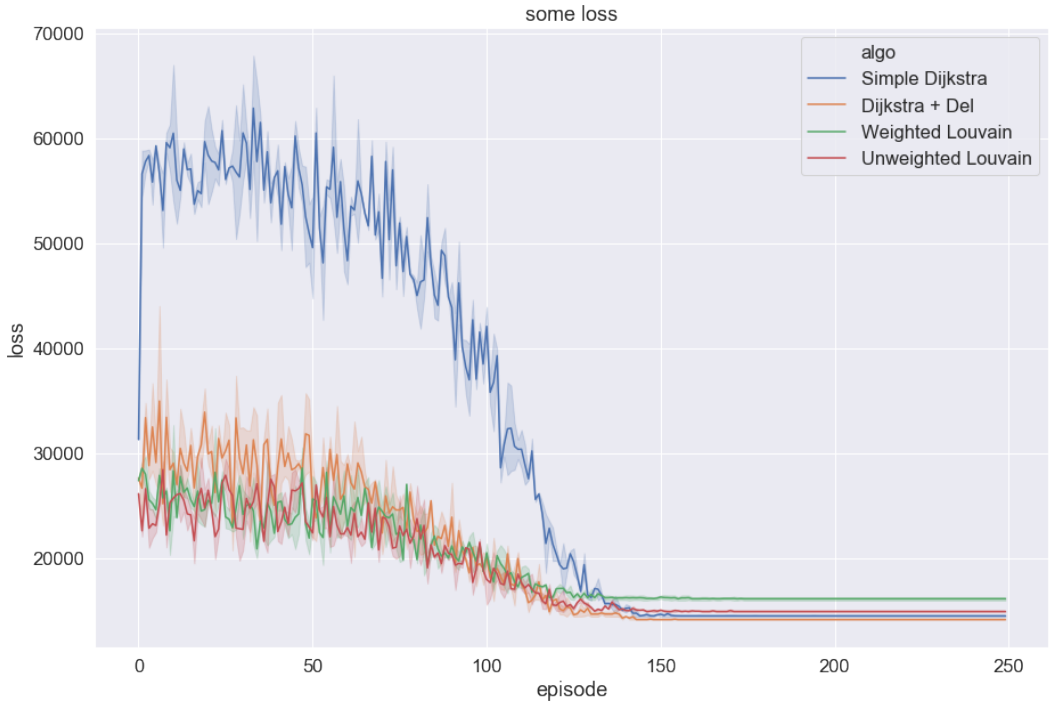

图 6: 全部算法实验结果

每种算法给出的最优解如下表。需要说明的是，我们项目主要想优化的是“在几乎不损失最优解的情况下该 NP 难问题的复杂度 °。这个想法的出发点是，NP 难问题在问题规模足够大的情况下，出于复杂度的考量往往想要的只是一个“比较好的解”，而最优解往往与随机种子强相关。所以与其不停尝试不同随机种子去找更好的解，我们认为对算法效率本身的优化、对短时间内能找到的解的优势程度进行优化才是更好的想法。

表 1: 全部算法最优解结果

	最优解
传统模拟退火	14400
优化后模拟退火	14240
自适应最短路径搜索算法（有权）	14530
自适应最短路径搜索算法（无权）	14400

1.5 邮递员问题总结

通过前述实验、分析，我们总共尝试了三大类、四种算法。对于启发式而言最重要的时间复杂度经过我们不断地优化，已经降到了完全可以接受的程度，接下来我们将对我们最终提出的自适应算法进行更为全面的评价。**在算法的优势方面**

- **复杂度相较传统算法大幅降低**：算法所需要探索的复杂度相较传统算法大幅降低。通过理论分析可以得知，该方法指数级地降低了算法的复杂度，使得算法需要探索的状态数量大幅度下降。

- **loss 始终较低**：在搜索过程中可以始终保持较低的 loss，如果对解的最优性要求不高，甚至较少的探索即可得到期望的结果。
- **自适应性**：自适应性保证了算法并不只针对某种特殊的环境，而是可以任意地移植到任何类似的环境中，在保证了较好的性能的同时提供了更好的泛化性。

而我们当前的算法也存在一定的**问题**

- **算法所测试的环境较少**：现在我们只是在作业题目给定的环境中进行了算法的测试，而对于一些极端情况——图是全连接、或本身连通性较差的情况，我们没有充分地测试，所以对于算法是否可以稳定地产出更好的结果还存在疑问，这也可以作为之后尝试的方向。
- **社团划分算法可以进一步优化**：现在所使用的 Louvain 算法兼顾高效与易实现，但作为一个已经存在了几十年的算法，它存在很多的优化空间，社团如何更好地自适应划分也是我们之后探索的方向。

2 最大流问题

2.1 问题描述

考虑如图 7 所示的交通网络，有 1 到 6 共 6 个结点，两个结点间的数字代表它们之间的最大运输能力，求结点 1 到结点 6 的最大运输能力（流量）。

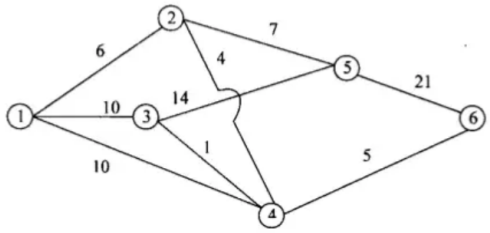

图 7: 交通网络示例

该问题是一个典型的最大流问题（Max Flow Problem）问题，我们希望找到一个流经方式使得从源点到汇点的流量最大。

在本章节中，我们调研了已有的算法，表 2 展示了我们的调研结果 [3]。我们对其中的多种方法做了尝试，实验将在下文依次展开。

算法	复杂度
Ford–Fulkerson 算法	O($Emax\|f\|$)
Edmonds–Karp 算法	O(VE^2)
Dinic 阻塞流算法	O(V^2E)
MPM 算法	O(V^3)
Dinic 算法	O($VElog(V)$)
push-relabel maximum flow 算法	O(V^2E)

表 2: 不同最大流算法调研

2.2 方法 1 ——Edmonds–Karp 算法

Edmonds–Karp 算法通过实现 Ford–Fulkerson 算法来计算网络中的最大流，其时间复杂度为 $O(VE^2)$。该算法由 Yefim (Chaim) Dinic 在 1970 年最先提出并由 Jack Edmonds 和理察 · 卡普在 1972 年独立发表。

该算法的伪代码如算法 4 所示。

Algorithm 4 Edmonds–Karp 算法

Input: 给定节点集 $Nodes$, 节点间最大流量 $Flow(Nodes_i, Nodes_j)$
Output: 最大流量 F

初始化残差网络，最大流量 F = 0
while 存在残差网络 **do**
  使用 BFS 在残量网络中寻找增广路
  if 存在增广路 **then**
    用增流更新残量网络，将其计入总流量 F
  else
    输出总流量 F，即为最大流 s
return 最大流量 F

2.3 方法 2 ——Dinic 算法

Dinic 算法（又称 Dinitz 算法）是一个在网络流中计算最大流的强多项式复杂度的算法，设想由以色列（前苏联）的计算机科学家 Yefim (Chaim) A. Dinitz 在 1970 年提出。算法的时间复杂度为 $O(V^2E)$ 。Dinic 算法与 Edmonds–Karp 算法的不同之处在于它每轮算法都选择最短的可行路径进行增广。Dinic 算法中采用高度标号（level graph）以及阻塞流（blocking flow）实现其性能 [4]。

该算法的伪代码如算法 5 所示。

Algorithm 5 Dinic 算法

Input: 给定节点集 $Nodes$, 节点间最大流量 $Flow(Nodes_i, Nodes_j)$
Output: 最大流量 F

初始化残差网络，最大流量 F = 0, 用 BFS 将残量网络分层，其中源点为第 0 层
if 汇点不可达 **then** 最大流量 F = 0
else 用 DFS 在分层图中寻找增广路，确保路径上的点的层次递增
  while 存在增广路 **do** 在 DFS 回溯时用增流更新残量网络, 重新寻找增广路
return 最大流量 F

2.4 方法 3 ——Push-Relable 算法

压入和重标记算法 (push-relable) 与 Ford-Fulkerson 方法不同，压入和重标记算法不是检查整个残留网络来找出增广路径，而是每次仅对一个顶点进行操作，并且仅检查残留网络中该顶点的相邻顶点。Push-relabel 算法被认为是最有效的最大流量算法之一。通用算法具有很强的多项式 $O(V^2E)$ 的复杂度 [5]。

该算法的伪代码如算法 6 所示。

Algorithm 6 Push-Relable 算法

Input: 给定节点集 $Nodes$, 节点间最大流量 $Flow(Nodes_i, Nodes_j)$

Output: 最大流量 F

将每个顶点的高度和流初始化为 0，初始化源顶点的高度等于合计图中的顶点数，对于与源 s 相邻的所有顶点，流量和多余的流量最初等于容量

while 存在残差网络 **do**

if 顶点有过多的流量且相邻顶点都不处于较低高度 **then**

选择最小的相邻高度（在残差图中，即我们可以向其添加流量的相邻高度）并对其加 1

if 如果顶点有多余的流动，并且在残差图中有一个相邻的高度较小 **then**

将流动从顶点推到相邻的较低高度。通过管道的推动流量等于多余流量和边缘容量的最小值

F= 与源 s 相邻的所有顶点的路径上的流量

return 最大流量 F

我们在一台 24GRAM, i7-7700HQ, 2.8GHz 的笔记本上分别对以上提到的三种算法进行了尝试，最终的结果如表 3 所示。由于原图为无向图，我们通过将每条无项边转化为两条指向相反、容量与无项边相同的有向边，得到了一个新的有向图，以应用上述算法求解最大流问题。

从算法的理论时间复杂度来看，EK 算法的复杂度为 $O(VE^2)$，而另外两个算法的复杂度均为 $O(V^2E)$，本次实验中无向图的节点个数为 6 个，边的数量为 9 个，从结果表明，EK 算法的速度明显低于 Dinic 和 Push Relabel，这和复杂度是吻合的。三个算法的最大流结果均为 23，经过简单验算之后可以确认这已经是最优解。

算法	运行 10000 次用时	结果
Edmonds–Karp	1.36s	23
Dinic	780ms	23
Push-Relabeled	878ms	23

表 3: 不同最大流算法对比

上述算法得到的一个最大流实验结果如图 8 所示，最大流量为 23。

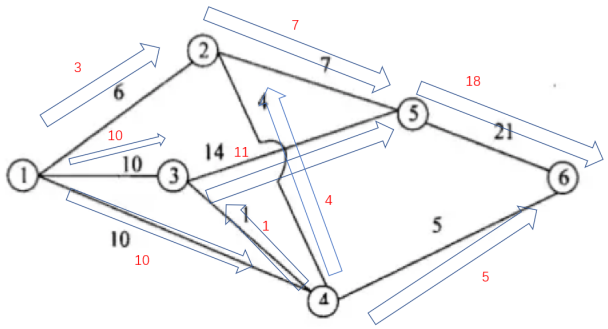

图 8: 最大流实验结果

3 无线传感器网络的覆盖问题

3.1 问题描述

当前环境检测对数据传输精度要求越来越高，无线传感器网络逐渐成为了研究的热点。节点的覆盖范围和监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。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，提高该网络检测多个目标的覆盖率和降低算法时间复杂度。

考虑覆盖问题。本问题假设在给定的 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 个传感器检测到才判定连接成功。我们的目的是，设计网络优化算法，使得成功连接的目标尽可能的多。

我们先假设这是一个全局的组合优化问题，即所有的传感器可以共享数据以决定目标的连接情况。为了方便算法上的叙述，在伪代码中称传感器为 sensor，称被检测目标为 target。我们首先以感知矩阵的形式进行建模。设第 i 个传感器为 $sensor_i$，其检测范围为 A_i，第 j 个被检测目标为 $target_j$，感知矩阵为 $P_{100\times100}$，其中 P 满足：

$$p_{ij}=\begin{cases}1, \text{if } target_j \text{ in } A_i\\ 0, \text{else}\end{cases} \tag{4}$$

，其中 p_{ij} 为感知矩阵 P 第 i 行，第 j 列的元素。我们需要求一个连接矩阵 $C_{100\times100}$，满足：

$$c_{ij}=\begin{cases}1, \text{如果 } p_{ij}=1 \text{ 且传感器 } sensor_i \text{ 与节点 } target_j \text{ 建立连接}\\ 0, \text{else}\end{cases} \tag{5}$$

。算法的目标是求得这样的 C，使得连接数大于等于 3 的被检测节点尽可能的多。表达成数学语言即为：

$$\begin{aligned}&\max_{C} \quad f(a^TC,3)a\\ &s.t. \quad a^Tf(Ca,6)=0\end{aligned} \tag{6}$$

$$\text{where} \quad a_{100\times1}=[1,1,\cdots,1], \quad f(x,y)=\begin{cases}1, \text{if } x\geq y\\ 0, \text{else}\end{cases}$$

3.2 贪心算法

贪心算法是一个非常容易想到的做法，我们直接对每个传感器要求连接最多数量的目标。由于采取不同的衡量标准会得到不同的贪心算法，我们设计了两种贪心算法作为基线。一种是直接贪心算法（Direct-Greedy，DG），对于每一个传感器，按照目标初始化生成时的顺序遍历连接其检测范围内的被检测目标，达到传感器的连接能力限制后，停止遍历，转向下一个传感器。这种算法由于传感器与目标的生成是随机的，其衡量标准也是随机的。另外一种是最近邻贪心算法（Nearest-Neighbor-Greedy, NNG）。其出发点是为了保证连接性能，每个传感器应该连接其检测范围内最近的 5 个目标（如果数量达到 5 个的话）。对于每一个传感器，按照目标初始化生成时的顺序遍历连接其检测范围内的被检测目标，达到传感器的连接能力限制后，停止遍历，转向下一个传感器。

3.3 遗传算法

遗传算法是一种智能算法，广泛应用于求解各种复杂组合优化问题 [6]。该算法模拟自然界中生物进化的过程，将问题的求解过程转换成类似生物进化中的染色体基因的交叉、变异等过程，通常能够较快地获得较好

的优化结果。

在本问题中，基于前面的假设，由于所有的传感器可以相互通信，我们不再需要对所有的可能连接进行搜索，而是通过一定预先的筛选，大大减少搜索空间。经过简单的分析后，我们得到了一些筛选条件：

- 对于连接数小于 3 的检测目标，所有传感器都不连接它们
- 所有连接数小于等于 5 的传感器不加入优化队列

基于以上条件，我们设计了改进后的遗传算法（Improved Genetic Algorithm，I-GA），算法伪代码如算法 7 所示。

Algorithm 7 改进后的遗传算法（I-GA）

Input: 感知矩阵 P，最大迭代次数 Max_iter，种群规模 $SIZE$，杂交概率 P_1，变异概率 P_2
Output: 连接矩阵 C

```
将 P 中列和小于 3 的列置零
随机初始化大小为 SIZE 的种群池 Pool，P 中行和小于等于 6 的行不加入优化队列
迭代次数 i = 0, 最优个体 I = max(Pool)
while i < Max_iter do
    对种群池 Pool 以 P1 概率杂交
    对种群中的每个子代以 P2 概率变异
    最优个体个体 I = max(Pool, I)
    用 I 替换 Pool 群中表现最差的个体
    i = i+1
return 最优个体 I 对应的连接矩阵 C_I
```

3.4 转化为最大流问题求解

经过文献查阅 [7] 和思考，我们发现，此问题还可以转化为最大流问题求解。为此，我们需要先以二分图 [8] 的形式对上述问题重新定义。

对于一个图 $G = (V, E)$，如果可以将顶点集 V 分为两个互不相交的子集 (A, B)，且图中的每一条边 e_{ij} 所关联的两个顶点 v_i, v_j 分别属于这两个顶点集，则称图 G 为一个二分图。若 G 为有向图，则称其为有向二分图。在本问题中，顶点集 A 即为传感器节点集合，顶点集 B 即为目标节点集合，若一个目标节点在另外一个传感器节点的检测范围内，则存在一条边由该传感器节点指向目标节点，图 9 左图给出了一个示例。

若能找到一个边集 E 的子集 E'，使得顶点集 A 与顶点集 B 中的每一个顶点最多与另外一个集合中的 1 个顶点相连（即一对一连接）时，称 E' 为 G 的一个匹配。G 的所有匹配中，包含边数最多的一组匹配成为 G 的最大匹配。解决最大匹配问题已经有很多优秀的算法，如匈牙利算法 [9] 等。若一个顶点集中的每个顶点最多与另外一个顶点集中的多个顶点相连（即多对一），则称这样的匹配问题为多重匹配问题，使得边最多的匹配则称为多重最大匹配，此类问题可以用最大流算法求解。

在本问题中，传感器节点集中的每个节点可以与目标节点集中的多个顶点相连，目标节点集中的每个节点可以也需要传感器节点集中的多个顶点相连，即为多对多问题。虽然与上述的多重最大匹配问题略有不同，但我们仍然可以用最大流方法近似解决此问题。

首先，在传感器节点集 $\{sensor_*\}$ 与目标节点集 $\{target_\#\}$ 构成的有向图 G 中，设每条边的容量为 1。其次，在图 G 中添加一个超级源点 S 和一个超级汇点 T，S 与每一个传感器节点 $sensor_*$ 之间存在一条由 S

指向 $sensor_*$ 的边，容量为 5；T 与每一个传感器节点 $target_\#$ 之间存在一条由 T 指向 $target_\#$ 的边，容量为 3，如图 9 右图所示。当我们求解由 S 到 T 的最大流量问题时，得到的解即为传感器覆盖问题的一个近似最优解。

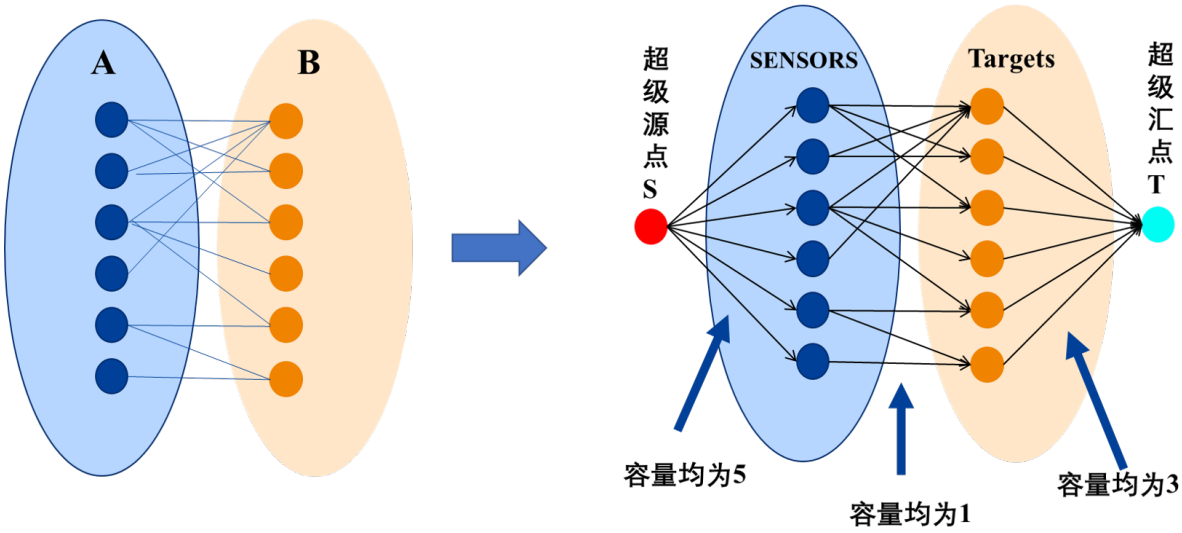

图 9: 二分图示例与传感器覆盖问题的二分图转化

上述转化过程中，传感器连接能力的限制被转化为超级源点流入其节点时的流量限制。而求解最多可连接的被检测目标则被近似的转化为求解由 S 到 T 的最大流量。当我们在求解到 T 的最大容量时，由于每个目标节点 $target_\#$ 到 T 的边容量受到限制，当传感器的数量足够时，为了得到最大流量，达到流量限制的 $target_\#$ 到 T 的边的数量会尽可能的多，这保证了解的优异性。

不过，求解过程中仍然有可能出现一个被检测节点 $target_j$ 本可以从第 i 个传感器节点 $sensor_i$ 处得到流量而达到流出流量限制（即检测成功），但由于 $sensor_i$ 连接了另一个未达到流出流量限制的被检测节点 $target_{j'}$ ，且 $sensor_i$ 达到了流出流量限制，导致 $target_j$ 检测不成功。这是由于从最大流量的角度看，$target_j$ 流向 T 的流量为 2、$target_{j'}$ 流向 T 的流量为 1 与 $target_j$ 流向 T 的流量为 3 、$target_{j'}$ 流向 T 的流量为 0 得到的最终流量是一样的，但从传感器覆盖问题的角度，前者比后者多出了一个成功检测的节点。为了缓解这种情况，我们在求解完最大流问题后加入了一个询问过程，即每一个满流量的传感器 $sensor_*$ ，“询问”其决定连接的、不满流的目标 $target_{j'}$ 和其能检测到但未连接的、即将满流的目标 $target_j$ 是否存在上述情况。若 $target_j$ 与 $target_{j'}$ 通过 $sensor_*$ 相连，且存在上述情况，则置 $sensor_*$ 到 $target_{j'}$ 的流量为 0，$sensor_*$ 到 $target_j$ 的流量为 1，使得 $target_j$ 被检测。

该算法的伪代码如算法 8 所示。

3.5 结果对比

本章节我们通过实验以对比各个算法的目标检测数量和时间性能。

3.5.1 传感器数量与目标数量均为 100 时的目标检测数量与时间性能

我们进行了 200 次随机试验。由于每次随机试验的环境不同，最多能成功检测的目标也不相同。我们通过排除掉那些连接数小于 3 的检测目标，对成功被检测的目标数量提出了一个上界，在当前设置下（传感器数量与目标数量均为 100），我们的实验证明此上界较为接近最优检测数量。利用此上界减去各个算法成功检

Algorithm 8 利用最大流求解传感器覆盖问题（Max-Flow，MF）

Input: 感知矩阵 P

Output: 连接矩阵 C

根据连接矩阵建立有向二分图 G
在 G 中添加超级源点 S 与超级汇点 T
在 S 到每个传感器节点 $sensor_*$ 之间添加容量为 5 的有向边
在每个目标节点 $target_\#$ 到 T 之间添加容量为 3 的有向边
建立邻接矩阵，求由 S 到 T 的最大流量
每个流出流量不满载的传感器节点 $sensor_*$ 进行“询问”，使得其检测范围内的目标尽可能满载
根据每条边的流量情况，生成连接矩阵 C
return 连接矩阵 C

测的目标数，将此作为纵轴，绘制出箱型图如图 10 所示。由箱型图可以看出，在 200 次实验中，我们的最大流算法不仅均值较高，几乎与我们估计的上界重合。算法的稳定性也较好，与上界距离的方差较其他算法明显减小。

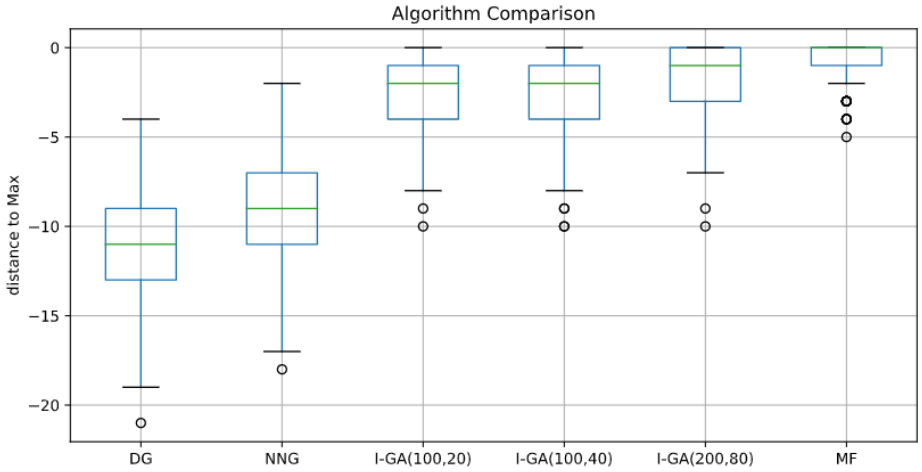

图 10: 传感器节点与目标节点均为 100 时，200 次随机实验下各个算法成功检测目标数量箱型图。圆点表示离群点，由于上界在某些情况下是不准确的，当上界偏差过大时，可能会出现离群点。每个箱中的绿线代表均值，箱的长度则可以近似表示其数据的分散情况。算法名称中，I-GA(m,n) 代表遗传算法采用了最大迭代次数为 m，种群数量为 n

在时间性能上，由于编程原因，如某些算法更多地使用遍历，而某些算法可能更多地使用 Numpy 库并行，表 4 给出的算法时间仅供参考。从表中可以看出，贪心算法的速度最快，且由于 NNG 的遍历操作较少，其算法速度远远大于 DG。遗传算法的运行速度次之，且当种群数量和最大迭代次数增大时，运行时间会显著增加。最大流算法最慢，因为我们在计算最大流时采用的是 Edmonds–Karp 算法，它的时间复杂度是 O(VE^2)，若采用 Dicnic 算法（时间复杂度为 O($VE\ logV$)）或其他运行效率更高的算法，算法的运行时间应该会简短。

3.5.2 目标数量增加时的目标检测数量

我们同样测试了当目标节点增多时，各个算法的表现性能，其结果如图 11 所示。由于当传感器节点数量与目标节点数量相差较大时，我们在前面估计的上界偏差也会严重增大，因此我们直接展示了各个算法的成

算法	DG	NNG	I-GA(100,20)	I-GA(100,40)	I-GA(200,80)	MF
运行时间 (s)	0.062	0.002	0.259	0.4927	1.872	2.100

表 4: 200 次随机实验下，各个算法运行的平均时间。

功被检测的目标数量。

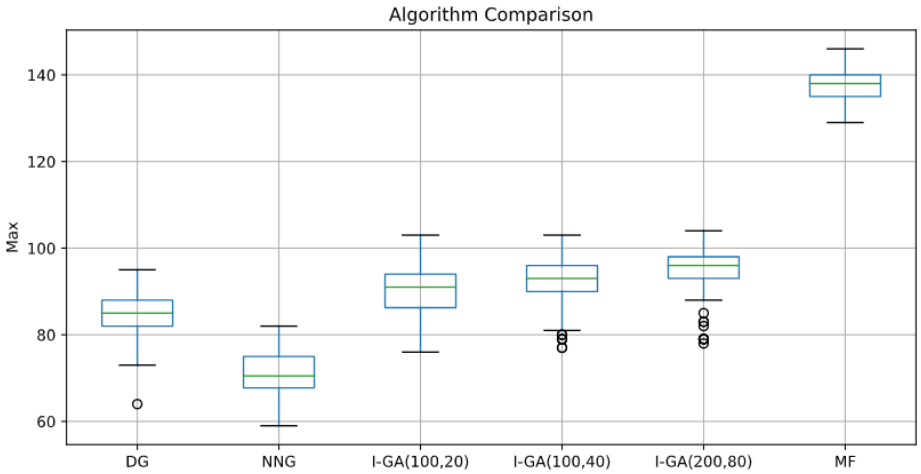

图 11: 传感器节点数量为 100，目标节点数量为 300 时，200 次随机实验下各个算法成功检测目标数量箱型图。

当传感器数量为 100, 目标数量为 300 时, 待检测目标分布的更为密集, 复杂度大大上升。可以看出, 贪婪算法与遗传算法的表现均不好, 而最大流算法仍然能保持较好的表现, 距离传感器的极限检测能力 ($100 \times 5 \div 3 \approx 166$) 也比较接近。

值得提出的是，当传感器数量为 300，目标数量为 100（即传感器数量远多于目标数量时），检测目标的难度会大大降低，因此这里对此种情况不再讨论。

3.6 总结

在本章节中，我们针对传感器覆盖问题比较了贪婪算法、遗传算法、最大流算法的效果与性能。结果显示，最大流算法的效果要好于其他两种算法，尤其是当传感器数量较少时，但其时间性能较其他算法稍差，尤其是当节点数增多时。为了解决这个问题，我们可以采用时间复杂度更小的最大流求解算法，也可以采用前文提到的 Louvain 算法先对图进行分块，再分别应用最大流算法。这也是未来可以优化的方向。

4 附录

4.1 课程感想与建议

经过一学期的学习，我认为这门课还是比较优秀的。课程内容涉及很多凸优化、图论和组合优化的知识，补充了我们这方面知识的欠缺。各个老师负责不同的模块，也显得课程条理清晰，结构明显。考核方式上，我认为这门课作为一门大学生涯后期的课程，采用大作业的考核方式也较为合理，不仅能够考察我们对于知识的应用，也避免了我们浪费时间在“刷题”上。

4.2 关于授课学期调整的看法

我认为这门课放在大三下学期是比较合理的。一方面，电院各个专业在大四上都没有什么课程，应该把这个时间留给同学们自己规划，例如考研、出国等。另一方面，这门课的内容也是比较扎实的，放在大三下能够促使同学们更加用心的听课、学习。

参考文献

[1] E. L. Lawler, "The traveling salesman problem: a guided tour of combinatorial optimization," *Wiley-Interscience Series in Discrete Mathematics*, 1985.

[2] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[3] "Maximum flow problem," Website, 2020, https://en.wikipedia.org/wiki/Maximum_flow_problem.

[4] E. Dinic, "An algorithm for the solution of the max-flow problem with the polynomial estimation," *Doklady Akademii Nauk SSSR*, vol. 194, no. 4, pp. 1277–1280, 1970.

[5] A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum-flow problem," *Journal of the ACM (JACM)*, vol. 35, no. 4, pp. 921–940, 1988.

[6] M. D. Vose, *The simple genetic algorithm: foundations and theory*. MIT press, 1999.

[7] 陈泽亚, 王庆, 郭静, 陈晰, and 王晶华, "基于二分图网络的项目与专家多重匹配策略," 小型微型计算机系统, vol. 37, no. 3, pp. 545–550, 2016.

[8] A. S. Asratian, T. M. Denley, and R. H"aggkvist, *Bipartite graphs and their applications*. Cambridge university press, 1998, vol. 131.

[9] M. J. Golin, "Bipartite matching and the hungarian method," *Course Notes, Hong Kong University of Science and Technology*, 2006.