车辆路径问题：整数线性规划与搜索方法

白晨旭　褚宸源　檀容韬

chenxu.bai@stu.pku.edu.cn

2200012860@stu.pku.edu.cn

trt01012345@stu.pku.edu.cn

https://github.com/a1henu/Vehicle-Routing-Problem-with-ILP-and-SA

2024 年 5 月 27 日

1　问题介绍

现在，我们有一定数量的客户，各自有一定的货物需求，我们有一个配送中心用于分发货物。配送中心用一个车队分送货物，我们需要对各车辆分配适当的行车路线，使得客户的需求得到满足，并在一定的约束下，达到诸如路程尽可能短、成本尽可能小、耗费时间尽可能少等目的。这就是车辆路径问题（Vehicle Routing Problem,VRP）。

车辆路径问题是一类常见的组合优化问题，同时也是 NP-hard 问题。在本研究中，我们使用北太天元与 MATLAB 软件，辅以 python 辅助编程尝试解决如下两个问题：

- 基本车辆路径问题（VRP）：设某配送中心有 m 辆车，需要对 n 个客户进行配送，客户 i 与客户 j 的距离为 c_{ij}。每辆车都需要从配送中心出发给若干客户送货，最终回到配送中心。求车辆行驶路程最短的配送方案。

- 容量限制车辆路径问题（Capacitated Vehicle Routing Problem, CVRP）：设某配送中心有 m 辆车，每辆配送车的最大载重量为 Q_i，配送中心需要对 n 个客户进行配送，客户点 i 的货物重量为 d_i，客户 i 与客户 j 的距离为 c_{ij}。每辆车都需要从配送中心出发给若干客户送货，最终回到配送中心。求车辆行驶路程最短的配送方案。

特别地，旅行商问题（Traveling Salesman Problem, TSP）* 也属于车辆路径问题，TSP 问题是 VRP 问题在 $m = 1$ 情况下的特例。

*旅行商问题，即假设有一个旅行商人需要经过一系列城市，他需要规划适当的路径，从某个城市出发，恰好每个城市经过一次，并最终回到出发的城市，使得整个旅行过程中的总距离最小。旅行商问题也是 NP-hard 问题。

根据具体的应用场景和约束条件，除了带基本车辆路径问题和带容量限制的车辆路径问题外，VRP 还有一系列常见的变体，现列举部分如下：

- 带时间窗的车辆路径问题（Vehicle Routing Problem with Time Windows, VRPTW）：每个客户都有一个可接受的服务时间范围，车辆需要在规定的时间内到达客户处进行服务。
- 多车型车辆路径问题（Heterogeneous Fleet Vehicle Routing Problem, HFVRP）：使用多种不同类型的车辆进行配送，每种车辆可能有不同的容量和成本。
- 动态车辆路径问题（Dynamic Vehicle Routing Problem, DVRP）：在配送过程中，客户需求或交通状况可能会发生变化，需要实时调整车辆路径。

车辆路径问题被广泛应用于物流、配送、运输等领域，它的研究不仅具有理论价值，还有重要的实际意义。车辆路线的实际问题包括配送中心配送、公共汽车路线制定、信件和报纸投递、航空和铁路时间表安排、工业废品收集等。

随着经济全球化和信息化进程的不断加快，物流作为具有广阔前景和增值功能的新兴服务业，正在全球范围内迅速发展。运输服务是物流组成中的重要环节，降低运输成本、提高运输质量和效率成为加快物流发展的有效途径。车辆路径问题作为运输服务优化的核心问题，对于提高物流系统的效率和效益具有重要意义。

2 算法简介

VRP 问题有很多求解算法，包括精确算法与启发式算法等。精确算法有分支定界法、动态规划法 [3] 等；启发式解法有遗传算法 [1]、模拟退火 [4]、蚁群算法、与基于神经网络的机器学习解法 [2] 等。由于 VRP 是 NP-hard 问题，我们难以用精确算法求解，因此启发式算法是求解车辆运输问题的主要方法。

本研究中，我们采用精确算法中的 0-1 整数规划和启发式解法中的模拟退火算法进行求解。以下是对两个算法的简要介绍。

- 0-1 整数规划

  规划类问题是常见的数学建模问题，常见的规划模型包括线性规划、非线性规划和整数规划三个种。当规划中的变量限制为整数时，称为整数规划。0-1 整数规划是整数规划的一个子问题，这里所有的决策变量 x_i[†] 只能取 0 或 1 这两个值，这里决策变量又可称为二进制变量。0-1 整数规划及一些常见的规划模型为许多科学与工程问题的解决提供了基础性的支撑。

[†]这里讨论一般情形，后文具体问题中我们使用 x_{ij}^k 表示决策变量。

- 模拟退火算法

  模拟退火算法（Simulated Annealing, SA）是一种基于概率的随机寻优算法，模拟了物理中固体物质的退火过程：在加温阶段，固体内部粒子随温度上升变得无序，内能增大；在冷却阶段，粒子逐渐趋于有序，内能减小，最终达到常温时的基态，此时内能最小。

  模拟退火算法从某一较高初始温度出发，伴随温度参数的不断下降，结合一定的概率突跳特性，随机寻找目标函数的全局最优解，即在局部最优中能有概率地跳出并最终趋于全局最优。

  模拟退火算法的过程简述如下：

  1. 初始化各种参数，如初始温度，最终温度，退火速率等。
  2. 重复以下步骤：
     (a) 产生新路径（新解）；
     (b) 计算新的总距离 x' 与当前路径的总距离 x；
     (c) 以概率 P 决定是否接受新路径；
  3. 逐步降低温度 T。

  直至温度降至截断温度，算法结束。

  上面算法中的概率 P 由 Metropolis 准则给出：

$$P(x', x) = \begin{cases} 1, & \text{if } x' < x \\ \exp\left(-\frac{x'-x}{T}\right), & \text{otherwise} \end{cases} \tag{1}$$

  这里 x 是当前时刻的系统能量（目标函数），x' 是新解下的系统能量（目标函数）。这意味着在搜索过程中，算法会根据一定的概率接受较差的解，从而避免陷入局部最优。

  模拟退火算法是一种通用的优化算法，具有概率的全局优化性能。目前已在工程中得到了广泛应用，如 TSP 问题、VRP 问题、VLSI 布局问题、排课问题、机器学习中的参数优化、图像处理等。

3 整数线性规划法求解 VRP 问题

3.1 问题建模

3.1.1 基本车辆路径问题

给定一组客户点、车辆容量、车辆数量、起始点和终点，目标是找到使得所有客户点都被访问一次的最短路径方案。基本车辆路径问题 (Vehicle Routing Problem, VRP)

的数学模型可以使用整数线性规划 (Integer Linear Programming, ILP) 来表示。以下是一个简化的 VRP 模型示例:

参数:

n: 客户点的数量（不包括起始点）

m: 车辆的数量

c_{ij}: 客户点 i 到客户点 j 的距离或成本

变量:

x_{ij}^k: 车辆 k 在访问客户点 i 后移动到客户点 j, 则变量 $x_{i,j}^k$ 是 1, 否则是 0.

目标函数:

$$\min \sum_{k=1}^{m} \sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij} \cdot x_{ij}^k$$

约束条件:

1. 每个客户点必须被访问且仅被访问一次:

$$\sum_{k=1}^{m} \sum_{j=1}^{n} x_{ij}^k = 1, \forall i = 1, \ldots, n$$

2. 每辆车的路径必须从起始点开始，并在终点结束:

$$\sum_{j=1}^{n} x_{0j}^k = 1, \forall k = 1, \ldots, m$$

$$\sum_{i=1}^{n} x_{i0}^k = 1, \forall k = 1, \ldots, m$$

3. Binary(0-1) 约束:

$$x_{ij}^k \in \{0, 1\}, \forall i = 0, \ldots, n, \forall j = 0, \ldots, n, \forall k = 1, \ldots, m$$

4. 每辆车不能停留在同一个客户点：

$$x_{ii}^k = 0, \forall i = 1, \ldots, n+1, \forall k = 1, \ldots, m$$

3.1.2 容量限制车辆路径问题 (CVRP)

与基本 VRP 类似, 但每个客户点有一个特定的需求量, 车辆需要满足总容量限制且在不超过容量的情况下为客户提供服务。

参数:

n: 客户点的数量（不包括起始点）

m: 车辆的数量

Q: 每辆车的容量限制

d_i: 客户点 i 的需求量

c_{ij}: 客户点 i 到客户点 j 的距离或成本

变量:

x_{ij}^k: 车辆 k 在访问客户点 i 后移动到客户点 j, 则变量 $x_{i,j}^k$ 是 1, 否则是 0.

目标函数:

$$\min \sum_{k=1}^{m} \sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij} \cdot x_{ij}^k$$

约束条件:

1. 每个客户点必须被访问且仅被访问一次:

$$\sum_{k=1}^{m} \sum_{j=1}^{n} x_{ij}^k = 1, \forall i = 1, \ldots, n$$

2. 每辆车的路径必须从起始点开始，并在终点结束:

$$\sum_{j=1}^{n} x_{0j}^k = 1, \forall k = 1, \ldots, m$$

$$\sum_{i=1}^{n} x_{i0}^k = 1, \forall k = 1, \ldots, m$$

3. 车辆容量限制:

$$\sum_{j=1}^{n} d_j \cdot x_{ij}^k \leq Q, \forall i = 1, \ldots, n, \forall k = 1, \ldots, m$$

4. Binary(0-1) 约束:

$$x_{ij}^k \in \{0, 1\}, \forall i = 0, \ldots, n, \forall j = 0, \ldots, n, \forall k = 1, \ldots, m$$

5. 每辆车不能停留在同一个客户点：

$$x_{ii}^k = 0, \forall i = 1, \ldots, n+1, \forall k = 1, \ldots, m$$

3.2 算法

因为 CVRP 问题是 VRP 问题的一个延伸, 因此我们在讨论解决 ILP 问题的算法时可以只讨论对于 CVRP 问题的算法。

3.2.1 数据载入

我们编写了一个简单的 python 程序来随机生成测试数据, 以便于我们调试代码. 在代码中，我们将所有的城市（客户点）分在了四个区域内，这将有利于后续进行的车辆路线的可视化。代码的输出将会把数据存储在一个相同目录下的 $.mat$，我们只需要将数据在 $.m$ 文件中直接将数据导入。

3.2.2 数据处理

除了现有的 n 个城市, 我们还需要考虑货车的出发点, 所以需要处理邻接矩阵。由于下面的演示示例中车辆起点到每一个客户点的距离已经定义，否则对于 n 阶方阵 c 来说，需要添加矩阵元素 c_{0j} 和 c_{i0}；由于车辆的总数是 m，所以要产生 m 个分量，每一个分量都是一个二维的矩阵 c。

```
x = optimvar('x', n+1, n+1, m, 'Type', 'integer', '
    LowerBound', 0, 'UpperBound', 1);
c_expanded = repmat(c, 1, 1, m);
obj = sum(sum(sum(c_expanded .* x)));
```

3.2.3 约束条件

等式约束和不等式约束需要分别进行处理

3.2.4 等式约束

```
eq_cons = [];
for i = 2:n+1
    eq_cons = [eq_cons, sum(sum(x(i, 2:n+1, :))) == 1];
end
for k = 1:m
    eq_cons = [eq_cons, sum(x(1, 2:n+1, k)) == 1, sum(x(2:
        n+1, 1, k)) == 1];
end
for i = 1:n+1
    for k = 1:m
        eq_cons = [eq_cons, x(i, i, k) == 0];
    end
end
```

3.2.5 不等式约束

```
ineq_cons = [];
for i = 2:n+1
    for k = 1:m
        ineq_cons = [ineq_cons, sum(d(2:n+1) .* x(i, 2:n
            +1, k)) <= Q];
```

```
5           end
6       end
```

3.2.6 解决 ILP 问题

在 Matlab 中，我们可以直接调用 intlinprog

```
1       prob = optimproblem;
2       prob.Objective = obj;
3       prob.Constraints.Equality = eq_cons;
4       prob.Constraints.Inequality = ineq_cons;
5       options = optimoptions('intlinprog', 'Display', 'off');
6       [sol, fval, exitflag, ~] = solve(prob, 'Options', options)
            ;
```

3.3 结果显示

```
1       if (exitflag == 0)
2           disp('No feasible resolution');
3       else
4           routes = cell(1, m);
5           for k = 1:m
6               route = [];
7               for i = 2:n+1
8                   for j = 2:n+1
9                       if sol.x(i, j, k) == 1
10                          route = [route, i];
11                          break;
12                      end
13                  end
14              end
15              routes{k} = route;
16          end
17
18          % Total distance
19          totalDistance = fval;
20      end
```

3.4 完整代码

```
% Example parameters
data = load("final_project/model.mat");
n = data.city;
m = data.veh;
c = data.maps;
n = 5; % Number of customer locations (excluding the depot)
m = 2; % Number of vehicles
Q = 10; % Vehicle capacity constraint
d = [0, 1, 1, 1, 1, 1]; % Demand of each customer point (excluding depot)
c = [0  17 29 20 21 16;
     17 0  19 15 17 16;
     29 19 0  15 18 16;
     20 15 15 0  22 16;
     21 17 18 22 0  16
     16 16 16 16 16 0]; % Distance matrix (including depot)

% Call the function
[routes, totalDistance] = solve_CVRP(n, m, Q, d, c);

% Display Results
fprintf('Optimal total distance: %.2f\n', totalDistance);
for k = 1:m
    fprintf('Route for vehicle %d: ', k);
    fprintf('%d ', routes{k});
    fprintf('\n');
end

function [routes, totalDistance] = solve_CVRP(n, m, Q, d, c)
    % SOLVECVRP Solves the Capacitated Vehicle Routing Problem (CVRP)
    %
    % Inputs:
    %   n - Number of customer locations (excluding the
```

```
    depot)
%   m - Number of vehicles
%   Q - Vehicle capacity constraint
%   d - Demand of each customer point (including depot
    as 0)
%   c - Distance matrix
%
% Outputs:
%   routes - Cell array containing the routes for each
    vehicle
%   totalDistance - Total distance of the optimal
    solution

% Include depot in the demand array if not already
    included

% Decision variables: x(i,j,k) indicates if vehicle k
    travels from i to j
x = optimvar('x', n+1, n+1, m, 'Type', 'integer', '
    LowerBound', 0, 'UpperBound', 1);

% Expand the distance matrix to match the dimensions
    of x
c_expanded = repmat(c, 1, 1, m);

% Objective function: minimize the total distance
obj = sum(sum(sum(c_expanded .* x)));

% Constraints
eq_cons = [];
ineq_cons = [];

% Constraint 1: Each customer point must be visited
    exactly once
for i = 2:n+1
    eq_cons = [eq_cons, sum(sum(x(i, 2:n+1, :))) ==
        1];
end

```

```
62          % Constraint 2: Each vehicle must start and end at the
                depot
63          for k = 1:m
64              eq_cons = [eq_cons, sum(x(1, 2:n+1, k)) == 1, sum(
                    x(2:n+1, 1, k)) == 1];
65          end
66
67          % Constraint 3: Vehicle capacity constraint
68          for i = 2:n+1
69              for k = 1:m
70                  ineq_cons = [ineq_cons, sum(d(2:n+1) .* x(i,
                        2:n+1, k)) <= Q];
71              end
72          end
73
74          % Constraint 4: Ensure x_ii^k = 0 for all i and k
75          for i = 1:n+1
76              for k = 1:m
77                  eq_cons = [eq_cons, x(i, i, k) == 0];
78              end
79          end
80
81          % Solve the problem
82          prob = optimproblem;
83          prob.Objective = obj;
84          prob.Constraints.Equality = eq_cons;
85          prob.Constraints.Inequality = ineq_cons;
86          options = optimoptions('intlinprog', 'Display', 'off')
                ;
87          [sol, fval, exitflag, ~] = solve(prob, 'Options',
                options);
88
89          % Extract routes
90          if (exitflag == 0)
91              disp('No feasible resolution');
92          else
93              routes = cell(1, m);
94              for k = 1:m
95                  route = [];
```

```
96                      for i = 2:n+1
97                          for j = 2:n+1
98                              if sol.x(i, j, k) == 1
99                                  route = [route, i];
100                                 break;
101                             end
102                         end
103                     end
104                     routes{k} = route;
105                 end
106
107                 % Total distance
108                 totalDistance = fval;
109             end
110         end
```

4 模拟退火方法求解 VRP 问题

4.1 搜索问题简介

搜索问题指的是在一个问题空间中寻找解决方案的过程。这个问题空间可以是各种形式的，例如图形、状态集合或搜索树等。搜索问题通常由以下要素构成：

1. 初始状态（Initial State）：描述问题的起始状态。
2. 目标状态（Goal State）：描述问题的目标或解决方案。
3. 操作（Actions）：定义在每个状态下可以采取的行动或操作。
4. 状态转移（State Transition）：描述从一个状态到另一个状态的转移规则，即在执行某个操作后，如何从一个状态转移到下一个状态。
5. 路径成本（Path Cost）：定义从一个状态到另一个状态的成本。

对于一般的搜索问题，我们可以先形式化地描述问题，然后使用搜索算法来寻找问题的解决方案。

一般地，搜索算法的目标是找到从初始状态到目标状态的路径，使得路径的成本最小。我们通常会从一个初始状态开始，然后通过不断地尝试各种操作，直到找到目标状态为止。

Algorithm 1: General Search Algorithm

Input : Initial State

Output: A path to the goal state

while *not reach the goal state* **do**

Choose an action;

Execute the action;

Update the state;

end

return *Path to the goal state*

4.2 模拟退火算法

模拟退火（Simulated Annealing，SA）是一种启发式搜索算法，用于在大规模搜索空间中寻找全局最优解。它的灵感来源于固体物理中的退火过程，即通过逐渐降低温度来使得物质达到低能量状态。在搜索问题中，" 能量" 可以被理解为目标函数的值，" 温度" 控制了搜索过程中接受更差解的概率。

在这个算法中，我们从一个初始状态开始，然后在每一步中生成当前状态的一个邻居。如果这个邻居的成本更低，或者即使成本更高但根据当前的温度和成本差异计算出的概率1 仍然决定接受这个邻居，我们就将当前状态更新为这个邻居。然后我们降低温度，并重复这个过程，直到温度降到 0 为止。最后，我们返回当前状态作为找到的最优解。

我们可以将模拟退火算法的过程简述如下：

Algorithm 2: Simulated Annealing Algorithm

Input : Initial State, Initial Temperature, Cooling Rate, Cutoff Temperature

Output: Optimal State

current_state ← Initial State;

temperature ← Initial Temperature;

while *temperature > Cutoff Temperature* **do**

next_state ← Generate a neighbor of current_state;

cost_difference ← Cost of next_state - Cost of current_state;

if *cost_difference < 0 or exp(-cost_difference / temperature) > random number between 0 and 1* **then**

current_state ← next_state;

end

temperature ← temperature * Cooling Rate;

end

return *current_state*

4.3 VRP 问题的搜索问题建模

对于 VRP 问题这样的 NP-hard 问题，我们可以将其建模为一个搜索问题，使用启发式搜索算法来寻找解决方案。

在 VRP 问题中，我们可以将每个状态定义为一个车辆的路径，每个操作定义为对路径的修改。我们可以通过不断地调整车辆的路径，直到找到一个最优的解决方案。

根据2，我们需要定义以下要素：

1. 初始状态：每个车辆的路径都是从配送中心出发，经过若干客户点，最终回到配送中心。
2. 目标状态：每个车辆的路径都满足约束条件，使得总成本最小。
3. 操作：对车辆的路径进行调整，例如交换两个客户点的顺序。
4. 状态转移：调整车辆的路径。
5. 路径成本：每个车辆的路径的总成本。

在算法实现中，我们需要确定初始状态的生成方式，确定当前状态的邻居的生成方式，确定如何计算当前状态的成本，以及确定如何根据当前状态的成本和温度来决定是否接受邻居。

对于初始状态，我们一般通过随机排列的方式来生成；对于邻居的生成，我们可以通过交换或插入客户点来实现。为了行文方便，我们采用下面记号：

getNext：生成当前状态的邻居。

getCost：计算当前状态的成本。

accept：根据当前状态的成本和温度来决定是否接受邻居。

4.4 模拟退火算法求解 VRP 问题

有了上文的框架，应用模拟退火求解 VRP 就变得十分简单了，我们可以直接套用模拟退火算法的框架2，只需要实现上述的三个函数即可。

需要注意的是，由于我们的邻居是通过随机交换或插入客户点来生成的，因此我们不一定在每一轮迭代中都能找到更优的解，为了保证算法的收敛性，我们在每一轮退火时都多次搜索迭代邻居，这样可以增加解收敛的概率。

Algorithm 3: Simulated Annealing Algorithm for VRP

```
Input  : Initial State, Initial Temperature, Cooling Rate, Cutoff Temperature
Output: Optimal State
current_state ← Initial State;
current_cost ← getCost(current_state);
temperature ← Initial Temperature;
while temperature > Cutoff Temperature do
    for i ← 1 to N do
        next_state ← getNext(current_state);
        next_cost ← getCost(next_state);
        cost_diff ← next_cost - current_cost;
        if accept(cost_diff, temperature) then
            current_state ← next_state;
            current_cost ← next_cost;
        end
    end
    temperature ← temperature * Cooling Rate;
end
return current_state
Function accept(cost_diff, temperature):
    if cost_diff < 0 then
        return True
    end
    return exp(-cost_diff / temperature) > random number between 0 and 1
end
```

4.5 模拟退火方法求解 CVRP 问题

当我们考虑 CVRP 问题时，我们需要增加约束条件，即车辆的容量约束。

上文我们提到，我们需要应用 *getNext* 函数来生成当前状态的邻居，但我们在生成邻居时采用了随机的方式，如何保证生成的邻居满足约束条件呢？

我们可以在每次随机时应用 *success* 函数来判断当前状态是否满足约束条件，如果不满足，则重新生成。

Algorithm 4: Generate Neighbor for VRP

```
Input  : Current State
Output: Next State
Function getNext (current_state):
    next_state ← copy of current_state;
    while True do
        Randomly select two customer points;
        current_state ← Swap the two customer points or insert one point to
         another;
        if success(current_state) then
            break;
        end
    end
    return current_state
end
```

那么，无论我们给 VRP 问题添加什么约束条件，只要我们能够修改 *success* 函数与其他相关数据结构的定义，我们就可以应用一般搜索问题的框架，进而使用模拟退火算法来求解。

5 总结

本文主要介绍了 VRP 问题的定义以及两种解决 VRP 问题的方法，分别是整数线性规划法和模拟退火法。

在整数线性规划法的部分，我们首先对问题进行数学建模，逐步找出问题的目标函数与问题约束，之后应用线性规划算法，得到最优解。在模拟退火法的部分，我们首先介绍了一般的搜索问题，之后对搜索问题的解决算法提出了一般性框架，之后介绍模拟退火的算法，最后针对 VRP 问题进行了模拟退火算法的具体实现。

两种方法各有优缺点，整数线性规划法可以得到最优解，但计算复杂度较高，适用于小规模问题；模拟退火法计算速度快，适用于大规模问题，但无法保证得到最优解。在实际应用中，我们可以根据问题的数据规模和精确度要求选择合适的算法来求解 VRP 问题。

参考文献

[1] Barrie M Baker and MA1951066 Ayechew. A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, 30(5):787–800, 2003.

[2] Jieyi Bi, Yining Ma, Jiahai Wang, Zhiguang Cao, Jinbiao Chen, Yuan Sun, and Yeow Meng Chee. Learning generalizable models for vehicle routing problems via knowledge distillation. In *Advances in Neural Information Processing Systems*, 2022.

[3] Imdat Kara and Tolga Bektas. Integer linear programming formulation of the generalized vehicle routing problem. In *Proc. of the 5-th EURO/INFORMS Joint International Meeting*, pages 06–10. Citeseer Istanbul, 2003.

[4] F Yu Vincent, AAN Perwira Redi, Yosi Agustina Hidayat, and Oktaviyanto Jimat Wibowo. A simulated annealing heuristic for the hybrid vehicle routing problem. *Applied Soft Computing*, 53:119–132, 2017.