基于模拟退火的容量受限弧路径问题的求解

王宇航 12012208

I. 项目概述

容量受限制的弧路径问题作为一个十分经典的问题，在现实生活中有着许多的应用，例如：信件投递规划，城市垃圾清理规划等等。同时，由于问题本身为NP-hard，所以这一问题很难在可接受的时间范围内寻得最优解。因此，如何在可接受时间内，通过局部搜索，寻求表现较好的可行解，一直是学界前沿所关注的问题。

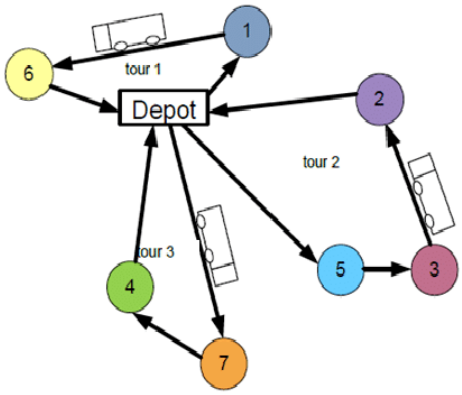

图 1. 容量受限的弧路径问题

容量受限的弧路径问题具体的定义为：在车辆承载容量受限制的情况下，针对给定的图和任务，规划出最优路线，使车辆可以完成全部任务且总成本最低。是一个典型的组合优化问题。

本项目采用模拟退火作为局部搜索的算法，在给定的时间限制下，采用多种传统算子与 merge-split 算子，对给定的图展开搜索。且采用多进程的方式，对程序进行速度优化。

II. 实验准备

1. 符号说明

符号	符号意义
V	车辆行驶过程中站点的集合
E	站点之间连接路段的集合
C	站点之间直连距离的集合
D	任意两站点最短距离的集合
S	需要执行的任务路段的集合
dem	需要执行任务的装载量的集合
L	车辆的容量
T	模拟退火过程中的温度
$\triangle t$	程序运行时间
v_{cur}	当前路径规划的成本
v_{best}	已规划过的路径中最优的成本
v_{next}	新一轮路径规划的成本
$\triangle V$	新规划与当前规划成本差值

2. 问题的数学描述

CARP 问题的具体定义在项目概述部分有所提及，然而，该定义并未将问题抽象化，公式化，难以运用数学工具，已有算法求解。因此，下面对问题进行抽象表示，便于后续求解。

通过本项目 CARP 问题的输入，可以得到一张无向加权图 G=(V,E,C)，$C_{i,j}$ 可表示站点 i 与 j 直接连接路段的距离。根据该图 G，本项目采用了 Floyd 算法，计算出该图 G 中任意两点的最短路径，表示为 $D_{i,j}$，其中，i，j 分别为起点和终点。注意 $D_{i,j}$ 和 $C_{i,j}$ 不一定相等。针对本问题，本项目需要做的是对问题给出的需要执行任务的路段集合做出规划，得出任务集合的一个

先后顺序的序列。针对每一个任务 S_k，其起点为 S_{ki}，终点为 S_{kj}。则此序列的总成本为

$$V(S) = \sum_{k=1}^{length(S)-1} C_{S_{ki},S_{kj}} + D_{S_{kj},S_{(k+1)i}}$$

同时，因受车辆容量 L 的限制，故需将总的任务序列划分成若干个子序列，每个子序列需满足条件：

$$restrict: \sum_{k=1}^{length(S_{sub})} dem(S_k) <= L$$

因此，本项目所研究的 CARP 问题可表示为：

$$min\ V(S)\ under\ restrict$$

III. 研究方法

1. 工作流程

- 解析输入的.dat 文件，初始化变量。
- 采用 Floyd 算法求最短路，采用 Path—Scanning 算法寻找初始解。针对存在多条最短路的情况下，采用随机函数随机选取。迭代 Path-Scanning 算法 10000 次，选取代价最小的路径规划作为初始解。
- 开启多进程，采用模拟退火算法。运用多种算子寻找当前解的邻域是否存在合适的解。退火结束后，选取退火过程中出现的代价最小的路径规划作为输出。

2. 模拟退火算法解析

CARP 问题为 NP-hard 问题，若采用全局搜索，理论时间复杂度为 $O(n!)$，是现有计算机算力无法承受的。模拟退火作为一种局部搜索算法，在面对搜索空间巨大的问题时，往往有着良好的表现。该算法改进了爬山法的不足之处，使得当前情况下较差的解也有几率被接受，从而增强了算法的探索性，避免过早的陷入局部最优。该算法最核心的地方就是如何设置接受函数。本项目采用 Metropolis 准则作为接受函数。Metropolis 准则具体如下所示，其中 $\triangle V = v_{next} - v_{cur}$：

$$p = \begin{cases} 1 & \triangle V < 0 \\ e^{\frac{-\triangle V}{T}} & \triangle V > 0 \end{cases}$$

当温度较高时，该算法表现为愿意接受当前较差的解，当温度逐渐下降，渐渐趋于稳定后，该算法表现为不情愿接受较差解。同时，本项目在模拟退火算法的基础上，进行了一些修改。退火的终止条件不再是温度 T 下降至某一阈值，而是最大化利用限定的运行时间，当剩余时间小于 1 秒时终止退火（如伪代码行 2 所示）。对于当前解偏离已找到的最优解过大的情况下，重置当前解与温度。即将目前已找到的最优解作为初始解，重新开始退火（如伪代码行 16 所示）。

Algorithm 1 Simulated Annealing

Input: input List(S),*time*

Output: output List(S)

1: $initial\ T, rate, v_{cur}, vbest, vnext$
2: **while** $\triangle t < time - 1$ **do** ▷ 最大化利用时间，不以温度作为退火结束标志
3: $v_{next} \leftarrow operator\ on\ List(s)$
4: $\triangle V \leftarrow v_{next} - v_{cur}$
5: **if** $\triangle V$ <0 **then**
6: $vcur \leftarrow vnext$
7: Update List(S)
8: **else if** $\triangle V$ >0 **then**
9: $v_{cur} \leftarrow vnext$ only with probabilitye$^{\frac{-\triangle V}{T}}$
10: Update List(S) only with v_{next} is accept
11: **end if**
12: **if** $v_{cur} > v_{best}$ **then**
13: $v_{best} \leftarrow v_{cur}$
14: Update List(S)
15: **end if**
16: **if** $v_{cur} > 1.2 \times v_{best}$ **then** ▷ 重置退火温度
17: $v_{cur} \leftarrow v_{best}$
18: Update List(S)
19: $reset\ T$
20: **end if**
21: $T \leftarrow T \times rate$
22: **end while**
23: return List(S)

3. 路径规划的算子解析

由于 CARP 问题为一个 NP-hard 问题，搜索空间巨大，故应采用局部搜索的策略。在局部搜索中，程序需要不断地从一个可行解转移到另一个可行解，对于这种转化，本项目实现了若干算子，这些算子作用于某一路径规划，可以得到新的路径规划。这些算子包括：

flip，Single Insertion；Double Insertion；Swap；2-opt 与 merge-split。其中，前 5 种算子为传统算子，每次对可行解的改变较小（步长较小）。而 merge-split 为一种步长较大的算子，对可行解的改变较大，在帮助跳出局部最优方面具有良好的表现。这里以该算法作为例子分析。该算法的具体步骤为，随机抽取路径序列 S 中的若干子序列，以这些子序列构成的图作为初始图，并进行 Path-Scanning 算法，重新排列子序列中的任务并返回。如下是实现该算子的伪代码

Algorithm 2 merge-split

Input: input List(S)

Output: output List(S)

1: $List(S)_{sub} \leftarrow$ split some tasks in List(S)
2: $List(S)_{sub} \leftarrow$ Path-Scanning on $List(S)_{sub}$
3: Merge $List(S)_{sub}$ into $List(S)_{sub}$
4: return $List(S)$

对伪代码进行分析，从序列中随机挑选 n 个任务的时间复杂度为 O(n)，将 n 个任务合并进任务序列中复杂度为 O(n)，而 Path-Scanning 算法的时间复杂度为 $O(n^2)$，故该算法的时间复杂度为 $O(n^2)$

4. 多进程加速

根据问题描述，用于计算的 CPU 为 8 核，因此，本项目采用了多进程的方式为程序加速，进程设置数为 8。值得注意的是，由于模拟退火算法的数据并不是相互独立的，因此无法采用多线程的方式，否则会造成数据错误。而多进程则为程序分配多个独立的内存空间，虽然无法针对一次模拟退火的全过程进行加速，但是可以同时进行多次独立的模拟退火算法，最后选取退火结果最好的作为输出。由于模拟退火算法每次迭代选取何种算子是完全随机的，通过这种方式，程序有更多的机会去搜索邻域，探索性大大增强。

本项目调用了 python 的 multiprocessing 包来实现该需求。

IV. 实验测试

1. 实验环境

本地硬件环境:

OS Windows11

CPU: AMD Ryzen 5 4500U，6-core total

本地软件环境:

python 3.9

numpy 1.22.2

服务器环境：

Operation System: Debian 10

Server CPU：2.2GHz*2, 8-core total

Python version: 3.9.

2. 数据集

本项目未用到除了给定的.dat 文件以外的数据集。同时，根据本地与服务器上的日志调整模拟退火的各项参数。

3. 模拟退火算法的表现分析

这一部分，笔者将针对模拟退火中的三个重要指标，程序运行时间，温度与当前可行解的总代价，进行分析，以此评估此项目中模拟退火算法实现的质量。采用 egl-s1-A.dat 作为输入，随机种子选定为 61。

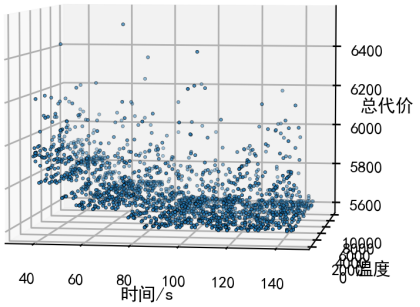

图 2. 模拟退火过程中各指标示意图

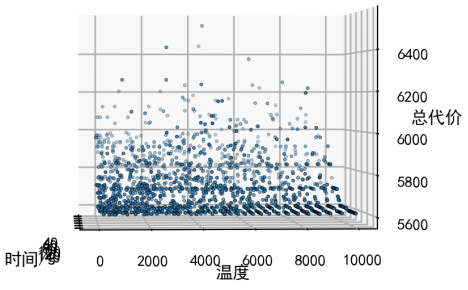

图 3. 模拟退火过程中温度与接受率示意图

由图 1 所示，随着程序的运行，当前解的总代价整体趋于不断减少的趋势。但由于模拟退火算法有一定几率可以接受较差的解，故各个时间段，总代价均有回升的情况。同时，模拟退火的接受几率对温度十分敏感，

温度越小，程序的表现越贪婪，不愿意接受差的解。由图 2 所示，当温度较高时，程序依然愿意接受偏离当前解较大的可行解，然而，温度下降后，程序虽然也会接受较差的解，但此解偏离较大的情况却大大减少。但温度降到 2000 以下时，可以发现 z 方向上的点已经十分密集了，很难找到离群较大的点。因此，本程序模拟退火算法的实现是基本符合逻辑的。

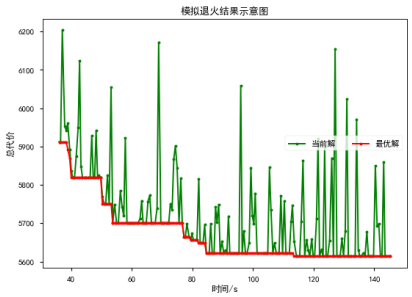

图 4. 模拟退火过程中当前解与最优解示意图

本程序在退火过程中，维护了目前所有找到的可行解中最优的解，称为最优解。图 4 展示了最优解与当前解随时间的变化关系。可以发现，最优解不断地降低，而当前解总是不断地跳跃，这与本项目不断重置退火温度有着紧密关联。本程序当当前解偏离最优解的值到达某一阈值时，便重置退火温度，使得退火算法能够一直保持较好的探索性。

4. 多进程对程序的提升

本项目采用 8 进程之后，由于探索的机会变为了原来的 8 倍，因此，程序的性能也有了较大的提升。针对大图 egl-s1-A.dat，在随机种子为 61 的情况下，结果为 5312。而单进程结果仅为 5614。由于本程序对于算子的选取均采用随机策略，故本程序对随机种子的依赖性较高。本地测试环节，选取了 11，21，31，41，51 等五个随机种子，运行 600s，所得出的最优解最好能达到 5200 左右，最差仅能达到 5600 左右。

V. 实验结论与展望

1. 实验结论

通过本项目的实验，笔者得出的结论为

- 模拟退火算法作为经典的局部搜索算法，针对 CAPR 问题有着良好的表现。然而，一开始解的质量提升的很快，后面就越来越慢了，甚至于很长的一段时间都没有任何提升。故该算法也十分依赖于每次更改状态的步长，帮助该算法搜索到更远的领域。而本项目所实现的算子大部分为小步长算子，还有待改进。
- 对于小图，仅通过迭代 Path-Scanning 算法若干次，便能直接寻得最优解。后面的退火过程并没有任何改善，这也说明了步长较大的算子存在的必要性。
- 程序的效率由算法的复杂度直接决定，但是在采用正确的算法后，最大化的利用硬件资源同样十分重要。常数级别的优化，也能给程序的表现带来很大的提升，

2. 项目有待完善的地方

- 由于时间原因，未针对大图和小图进行分别处理。
- 模拟退火的参数均手动设置，没有学习的过程。
- 代码结构较为混乱，未做到易读易懂。

3. 项目展望

CARP 问题作为经典的 NP-hard 问题，对于启发学界改进局部搜索算法有着巨大意义。同时，由于未来城市数字化，甚至元宇宙的来临，必然对路径规划，组合优化类问题的解提出更高的要求。有理由相信 CARP 问题的解会在将来不断被优化，而笔者也很高兴能动手实践，这确实是一段有趣的经历。

References

参考文献

[1] Ke tang, Yi mei, Xin yao. (n.d.). Memetic Algorithm with Extended Neighborhood Search for Capacitated Arc Routing Problems. doi:10.1109/TEVC.2009.2023449

[2] Yhcrown. (n.d.). CS303-AI-Projects. Retrieved from https://github.com/Yhcrown/CS303-AI-projects