

分类号

学校代码 10487

学号 M202073281

密级

# 华中科技大学

# 硕士学位论文

(学术型☒ 专业型☐)

## 求解带时间窗的车辆路径问题 的启发式算法研究

学位申请人：李云皓

学科专业：计算机软件与理论

指导教师：吕志鹏 教授

答辩日期：2023 年 月 日

---

**A Dissertation Submitted in Partial Fulfillment of the Requirements  
for the Master Degree in Engineering**

**Heuristic Algorithm for Solving Vehicle Routing Problem  
with Time Windows**

**Candidate : LI Yunhao**

**Major : Computer Software and Theory**

**Supervisor : Prof. LÜ Zhipeng**

**Huazhong University of Science and Technology**

**Wuhan 430074, P. R. China**

**May, 2023**

## 独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的科研成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期： 年 月 日

## 学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保 密 ☐，在 \_\_\_\_\_ 年解密后适用本授权书。

本论文属于 不保密 ☐。

（请在以上方框内打“√”）

学位论文作者签名：

指导教师签名：

日期： 年 月 日

日期： 年 月 日

## 摘要

随着我国电商行业的迅速发展,物流行业也迎来了新的机遇和挑战。在物流配送的过程中,如何提升效率同时减少能源消耗,是当前面临的主要难题,也是国家碳中和发展战略的内在需求。带有时间窗的车辆路径问题(The Vehicle Routing Problem with Time Windows, VRPTW)是运筹优化领域的经典问题之一,是车辆路径问题(The Vehicle Routing Problem)的变种。VRPTW 有两个优化目标,最小化路径总数量和最小化所有路径的总长度。

针对 VRPTW 的两个目标,本文分别提出了不同的求解思路,并且设计相应的算法。在优化路径总数量问题上:(1)算法使用了自适应加权引导的搜索,为解中的每条路径设置了权重,增强了算法搜索过程中的疏散性。(2)为每个顾客设置了权重,配合弹射池结构,从局部解弹出顾客来松弛解,显著提升了算法搜索的效果。(3)将 VRPTW 松弛之后转化为最大团问题,寻找 VRPTW 的路径数的下界。在最小化路径总长度的目标上:(1)算法应用了闪烁机制改进了边组装的交叉算子。(2)将破坏重建算法和边组装模因算法进行了融合,设计了混合局部搜索算法,该算法在求解优度与求解速度上均有良好表现。

在路径数量最小化问题上,算法在给定的时间限制内持平了 Gehring and Homberger 的基准算例集中的所有当前最优解,并且使用求解最大团启发式算法发现了 10 个算例的当前最优解已经达到了路径数的下界。在最小化路径总长度问题上,算法在第十二届美国离散数学和理论计算机科学中心的实现挑战赛中取得了第二名的成绩。

**关键词:** 车辆路径问题; 时间窗; 启发式搜索; 加权

## Abstract

With the rapid development of China's e-commerce industry, the logistics industry has also ushered in new opportunities and challenges. In the process of logistics and distribution, how to improve efficiency and reduce energy consumption is the main problem currently facing, and it is also the internal demand for China's carbon-neutral development strategy. The vehicle routing problem with time windows is one of the classic problems in the field of operational research optimization, and it is a variant of the vehicle routing problem. VRPTW has two optimization objectives, minimizing the total number of routes and minimizing the total distance of all routes.

This paper proposes different solutions for these two objectives and designs corresponding algorithms. On the objective of minimizing the number of routes: (1) The algorithm uses adaptive weighting-based local search, and sets weights for each route in the solution, which enhances the dispersibility of the algorithm in the search process. (2) The weight is set for each customer, and with the structure of the ejection pool, the customer is ejected from the partial solution to relax the constraints, which improves the effect of the algorithm search. (3) Transform VRPTW into a maximum clique problem after relaxation, and find the lower bound of the routes number of VRPTW. On the goal of minimizing the total distance of the routes: (1) The algorithm applies the blink mechanism to improve the edge assembly memetic algorithm. (2) Combining the destruction reconstruction algorithm and the edge assembly memetic algorithm, a hybrid local search algorithm is designed, which has a good performance in the solution optimality and solution speed.

On the number of routes minimization, the algorithm reaches all best-known solutions of Gehring and Homberger's benchmark within a given time limit. We use the heuristic algorithm of maximal clique and find the best-known solutions for 10 instances that have reached the lower bound of the number of routes. On minimizing the total distance of routes, the algorithm won second place in the 12th DIMACS Implementation Challenge.

**Keywords:** Vehicle routing problem, Time windows, heuristic search, Weighting

目 录

摘 要 .....	I
Abstract.....	II
<b>1 绪论</b>	
1.1 研究背景与意义 .....	1
1.2 VRP 国内外研究现状.....	3
1.3 研究对象及目标 .....	9
1.4 论文结构安排.....	9
<b>2 带有时间窗的车辆路径问题描述与分析</b>	
2.1 VRPTW 的问题描述 .....	11
2.2 VRPTW 的符号定义和数学模型 .....	13
2.3 带有时间窗的车辆路径问题分析 .....	15
2.4 本章小结.....	16
<b>3 求解路径数量最小化问题的自适应加权搜索算法</b>	
3.1 自适应加权局部搜索算法总体框架.....	17
3.2 初始解生成.....	18
3.3 顾客加权的路径移除算法.....	19
3.4 路径加权的修复算法.....	22
3.5 分支限界的弹射算法.....	29
3.6 带有禁忌表的扰动算法.....	31
3.7 探索路径数的最大下界.....	33
3.8 结果分析.....	34
3.9 本章小结.....	40
<b>4 求解路径总长度最小化问题的混合局部搜索算法</b>	

# 华中科技大学硕士学位论文

---

4.1	混合局部搜索算法总体框架.....	41
4.2	种群集合初始化.....	42
4.3	简单局部搜索.....	43
4.4	寻找最佳种群.....	43
4.5	改进的 SISR 算法.....	44
4.6	改进的 EAMA 算法.....	46
4.7	混合局部搜索算法.....	47
4.8	算法实现及单元测试.....	48
4.9	结果分析.....	50
4.10	本章小结.....	61
<b>5</b>	<b>总结与展望</b>	
5.1	主要内容以及结论.....	62
5.2	主要创新点.....	62
5.3	未来工作展望.....	63
	<b>致 谢</b> .....	<b>65</b>
	<b>参考文献</b> .....	<b>66</b>

## 1 绪论

### 1.1 研究背景与意义

带有时间窗的车辆路径问题（The Vehicle Routing Problem with Time Windows, VPRTW），是经典 NP-hard 组合优化问题之一，是车辆路径问题（The Vehicle Routing Problem, VRP）的变种问题。VRP 在 1959 由 Dantzig and Ramser 提出至今<sup>[1]</sup>，被广泛研究了超过半个世纪。VRP 来源于实际的物流运输中的路径优化难题，该领域的研究成果可以直接应用于物流配送中的成本优化。图 1.1 是国家统计局发布的近五年的快递行业数据<sup>1</sup>，我国 2022 年度快递量突破 1089 亿件，除此之外，相比于前几年，快递业务量的增速也在持续增加。为适应市场需求，物流行业的运输投入也在迅速增加。

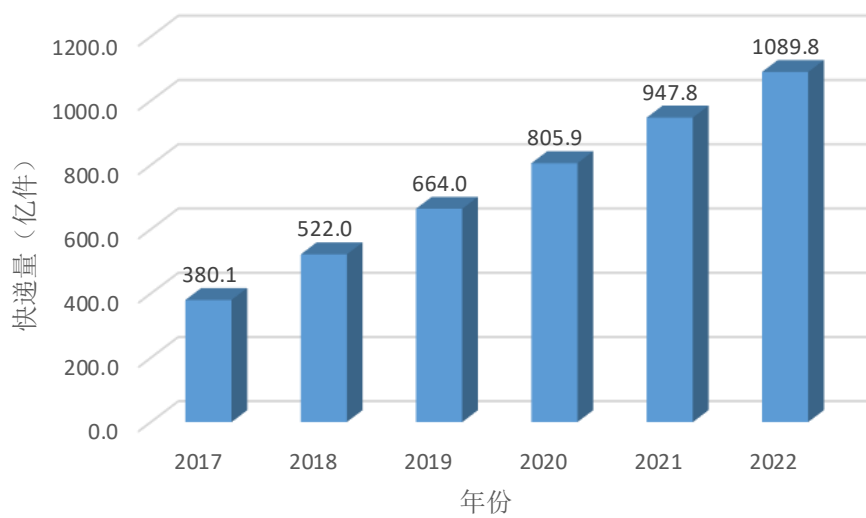


图 1.1 国家统计局 2017-2021 年度快递业务数据图

与此同时，运输过程的成本优化也引起了国内电商企业的浓厚兴趣。阿里巴巴公司成立了菜鸟物流部门，在 2019 年前后，菜鸟网络的算法部门刷新了 VRPTW 的世

<sup>1</sup> <https://data.stats.gov.cn/easyquery.htm?cn=C01&zb=A0G0H&sj=2022>



界纪录榜单中 48 个已知的最优解<sup>1</sup>。京东公司建立了整套的仓储物流配送体系。除此之外 VRP 问题也因为其研究年限长,改进难度大,常常作为算法效果的试金石,科技公司也会针对这类 NP-hard 问题专门设计算法,用以提升和检测算法团队的能力。华为的华为云擎天架构调度算法团队在带有时间窗的取送货路径规划问题更新了 51 个已知最优解<sup>2</sup>。

VRP 有上百种变体问题,在 Google 学术中的数据中,仅仅在 2022 年就有超过 17000 篇文献中包含车辆路径 (Vehicle Routing) 关键字。在 VRP 中的多个变种问题中,带有时间窗的车辆路径问题 (VRPTW) 是约束最强,求解难度最大的变种之一。VRPTW 主要的特点是在带有容量约束的车辆路径问题 (Capacitated Vehicle Routing Problem, CVRP) 的基础上增加了顾客的时间窗的约束。配送的车辆到达顾客的时刻,需要满足时间窗的要求。这一约束的引入,增加了问题求解方法设计的复杂度,求解的方法不能从其他问题做简单迁移,需要针对时间窗约束专门设计有效的方法与策略进行优化。

VRPTW 最早提出于 20 世纪 80 年代,在 1985 年被证明是 NP-hard 问题<sup>[2]</sup>。经过四十多年的研究,许多高效的算法被提出和应用。主要包括启发式算法和精确算法两大类,也有一些机器学习的方法被应用于求解 VRPTW,例如 Google 公司的 OR-Tools 等<sup>3</sup>,将强化学习应用于求解 VRP<sup>[3]</sup>。机器学习类的算法与启发式算法相比,优势更多的在于方法的创新性,因为算法没有在公开的基准算例集上与最好的启发式算法进行比较,所以本文中没有将该类算法作为对比对象。

精确算法在求解 VRPTW 上得到了良好的效果,其中在顾客规模在 400 个及以下,使用精确算法可以得到并证明最优解。但是精确算法在求解规模较大的 VRPTW 上仍然存在挑战,启发式算法填补了这一块的空白。不同于精确算法,启发式算法在求解优度和求解速度之间做了一个折中,在有限的时间限制内,求解得到一

---

1 <https://developer.aliyun.com/article/697250>

2 <https://www.huaweicloud.com/news/2020/20201207073226.html>

3 <https://developers.google.com/optimization/routing/vrptw?hl=zh-cn>

个可以满足工业应用需求的高质量解。启发式算法能在给定时间内求解得到符合要求的解是以牺牲数学完备性作为代价的，算法本身并不能证明求解到的解是全局的最优解，即使已经通过计算得到了最优解。即便如此，启发式的求解方法仍然有很大的应用空间。这是因为在实际的工业生产中，不仅要求求解的优度，对求解的时间也有严格的限制，而这些限制与启发式的优势和缺陷恰好契合，所以以启发式方法求解 NP-hard 问题，从而解决实际工业问题，也是学术界多年来研究的重点方向之一。

为了针对性提高 VRP 求解算法的性能，美国离散数学和理论计算机科学中心(the Center for Discrete Mathematics and Theoretical Computer Science, DIMACS)在 2022 年举办了第十二届 DIMACS 实现挑战赛，竞赛的主题是 VRP，包括五个赛道，分别 CVRP, VRPTW 等 VRP 相关的经典问题，VRPTW 是比赛的赛道之一。

## 1.2 VRP 国内外研究现状

与多数 NP-hard 组合优化问题一样，VRPTW 的求解方法也可以分为精确算法和启发式算法。

### 1.2.1 精确算法

在 1992 年，Desrochers 等人将列生成的算法用于求解 VRPTW<sup>[4]</sup>。此后这种方法被继续研究，Lozano 等人也使用列生成方法<sup>[5]</sup>求解 VRPTW。列生成的主问题和子问题方法被 Kallehauge 等人提出<sup>[6]</sup>。基于拉格朗日松弛的精确算法由 Kohl 和 Madsen 在 1997 年提出<sup>[7]</sup>。由 Ruslan Sadykov 以及 Eduardo Uchoa 等人使用一种基于桶图的标记算法<sup>[8]</sup>求解得到了若干公开算例集上的最优解。

近年来，在精确求解 VRP 方面取得了重大的进展，一个重要的里程碑是分支限界的算法<sup>[9]</sup>被应用于求解 CVRP，与之前的精确算法不同的是，这种分支限界算法模型可以求解更大规模的算法，求解的问题规模由之前的 150 个顾客增加到了 360 个顾客，求解问题的规模得到了很大的提升。同时将该模型应用于求解 VRPTW 问题，也取得了很大的成功，一种基于有限记忆化的剪枝方法<sup>[10]</sup>在 VRPTW 的标准算例集上也得到了相同的提升。

精确算法求解 VRPTW 的优势在 400 个顾客以及以下的中小算例，使用精确的

限制规则，可以算到 VRPTW 的下界，是完备的可以用于证明的一类算法。但是求解算法求解 VPRTW 的用时都比较长，在实际的工业应用中往往对算法求解的时间有限制，所以精确算法的使用场景比较有限。

## 1.2.2 启发式算法

有关启发式算法研究 VRPTW 的文献很多，从该问题提出到现在，大量的算法被提出用于优化改进该问题。启发式算法的发展是循序渐进的，从开始的简单贪心构造，到后期逐步建立搜索的框架，从单个解的搜索，到群体的搜索，以及精确算法的发展，都是基于前人的基础上发展起来。回顾早年间的启发式算法，有些看似简单的搜索策略与技术，在经过几十年的发展之后，还依然能够看到这些基础算法的影子。在求解该算法的研究中，有不少学者是基于分层的目标进行求解的，优化的第一步是最小化路径的总数量<sup>[11][12]</sup>，在最小化总数量的基础上优化所有路径的总长度<sup>[13]</sup>。

### 1.2.2.1 构造算法

在 20 世纪 60-80 年代，学术界在求解 VRP 问题，大多是基于构造的方法。算法起始的时候将每个顾客单独作为一条路径作为初始解，在后续迭代的过程中通过合并路径来优化，通过简单地计算收益值来选择合并的路径<sup>[14]</sup>。

20 世纪 70-80 年代，Sweep 算法<sup>[15]</sup>使用与顾客与仓库的射线夹角的顺序进行构造路径，一直到满足一条路径的容量约束之后创建一条新的路由。Beasley 等人使用一种基于旅行商（Traveling Salesman Problem, TSP）回路的方法构造解<sup>[16]</sup>，先不考虑仓库将所有的顾客连接成为一个回路，之后使用动态规划模型将一整条回路切分成一条条的路径。Vidal 等人在此基础上进行优化，切分算法的时间复杂度优化到  $O(n)$ <sup>[17]</sup>。

### 1.2.2.2 局部搜索算法

VRP 发展历史上一个重要的里程碑，就是局部搜索算法，而局部搜索算法中的重要内容就是邻域动作的设计。邻域动作的算子可以大致分为两种，一种是普通的邻域结构，这类邻域评估范围通常是多项式的，另一种是，例如交换两个顾客，包括同一条路径之内的两个顾客和不同路径之间的两个顾客<sup>[18]</sup>。VRP 中经典的邻域动作主

要包括单个顾客的移动,以及两个顾客的交换,顾客的交换动作和移动动作可以发生在一条路径之内,也可以发生在路径之间。路由类问题的邻域动作很多基本可以通用,VRP中有部分的邻域动作是直接从TSP中移植过来的。例如2-opt算子,在VRP问题中可以在单条路径之内只用2-opt算子,也可以在路径之间使用2-opt\*算子<sup>[19]</sup>。Maximo等人提出的混合自适应迭代局部搜索与多样化控制算法<sup>[20]</sup>,在迭代搜索的过程中对多样性进行控制。高效的移动评估和邻域结构的限制对于求解VRP实例至关重要,Santana等人尝试使用学习的方法获得一些可用信息用于邻域限制,但是在该方法应用不是很成功<sup>1</sup>。

与绝大多数局部搜索算法一样,在求解VRP时,算法也很容易陷入局部最优。这是绝大多数启发式算法都需要面临和解决的问题。因此在仅使用经典的邻域动作进行搜索陷入局部最优之后,研究者设计出了大邻域搜索解决这一问题。例如Stefan Ropke等人提出了自适应的大邻域搜索算法<sup>[21]</sup>,算法可以自动选择算子对解进行破坏和修复。以下详细介绍几种带有大邻域算子的搜索算法,包括破坏和重建方法,弹射链算法,以及swap\*。

(1) 破坏和重建方法 (Ruin and Recreate), 算法的第一阶段通过一定的规则将局部解中的路径进行破坏,将破坏之后的顾客从解中移除。破坏局部解的方法使得解的约束得到了松弛,从而更容易跳出局部最优解。破坏的策略包括随机找一些顾客,在圆形区域中的顾客。算法的第二阶段将移除出局部解的顾客从解中重新插入进去,使用不同的贪心策略对于待插入的顾客进行排序,在插入顾客的时候首先保证合法性,其次考虑优度。通过连续移除引导的松弛算法 (Slack Induction by String Removals for Vehicle Routing Problems) 中提出了一种移除一条路径上的相邻顾客来对可行解进行松弛。在插入顾客的阶段提出了一种闪烁的机制<sup>[22]</sup>,该机制可以以一定的概率跳过最好的插入位置,从而增加了算法的疏散性。

(2) Glover 在局部搜索中加入了弹射链 (Ejection Chain) <sup>[23]</sup>,这种邻域首先会选择一个或者若干个顾客从解中排除,为移除顾客形成的空位置寻找其他顾客,从而

---

<sup>1</sup> <https://arxiv.org/abs/2210.12075>

再次形成空位置，以此循环，最后将开始排除的顾客重新放入解中。这类邻域动作的运作机理是，在排除第一个顾客的时候，路径的总长度变短了，在接下来的每次移动顾客的过程中都要获得新的收益，最后放入顾客的时候需要额外增加代价，只要保证获得的收益比增加的代价更大，那么就是一次有效的弹射链动作。

(3) Vidal 提出了  $\text{swap}^*$ 算子<sup>[24]</sup>，传统的交换动作是选择两个顾客进行交换， $\text{swap}^*$ 的交换选择的对象是两条路径，对于任意的两条路径，从第一条和第二条路径中各自选择一个想要移除的顾客，之后将第一条路中移除的顾客放入第二条路的最佳插入位置。同理将第二条路径中移除的顾客放入第一条路径中最佳的插入位置。这样的移动与  $\text{swap}$  的不同点在于，首先将顾客移除出去所在的路径之后，再考虑将其他顾客放入这条路径。此时备选的位置可以是路径中的任意位置。

任何一种算法都会收敛最后陷入局部最优，大邻域搜索也不例外。在大邻域搜索算法的潜力被开发完之后，算法再次面临局部最优问题。一种更加智能引导类的局部搜索应运而生。引导类算法的一般做法是给要求解的问题的某种属性加上惩罚值，或者改变目标函数的计算方式。通过这些改变，原问题解空间的地貌发生了变化，之前因为算法陷入局部最优而看不到的解空间，很可能因此就被暴露出来，从而比较容易搜索到这些解。以下介绍几种引导类局部搜索算法：基于顾客惩罚值引导的高效的路径数最小化算法，重新定义时间窗惩罚的算法，以及对于不良路径连边进行惩罚的算法。

(1) Nagata 在 2009 年基于弹射池 (Ejection Pool) 设计了一个高效的路径数最小化算法就是这种思想的一个典型应用<sup>[12]</sup>。算法中为每一个顾客设置了一个惩罚值，在算法运行的过程中动态更新，用于标记该顾客与其他顾客的排斥程度。算法根据惩罚值来挑选顾客，将他们排除在当前解中。因为惩罚值是动态变化的，所以算法关注的顾客总是在动态变化，陷入局部最优的概率就小。

(2) Koskosidis 等人<sup>[25]</sup>对于时间窗的惩罚进行了重新的定义，不仅在晚于时间窗的右端点有惩罚值，在早于时间窗的左端点也加上了惩罚值。这样做的好处是使得每一条路径对于时间窗的利用达到最大值，使得车辆等待的时间最短。通过改变目标函数值的计算方式，来达到引导算法向着不同的搜索方向进行搜索的目的。

(3) Arnold 和 Sörensen 等人<sup>[26]</sup>提出了基于知识引导的局部搜索,通过惩罚不良的路由连接边,来引导局部搜索。不良的路由连边的识别来自历史搜索过程中发现的质量较差的解中的某些连边。Arnold 和 Vidal 使用模式挖掘用于识别有用的高阶移动用于引导局部搜索的搜索方向<sup>[27]</sup>。

### 1.2.2.3 种群进化算法

种群进化的算法(文化基因算法),最早由 Moscato P 等人提出<sup>[28]</sup>。遗传算法的框架在启发式求解 VRP 问题上获得了很大的成功,不管是 CVRP 问题还是 VRPTW 问题,目前最好的算法都是基于群体遗传的框架上进行搭建的。遗传算法模拟了基因进化的过程,优良的基因从父代传给子代,群体的进化方向是确定的,进化若干代之后种群收敛。这类算法的框架通常包括一个解池(部分算法也有设计两个解池的),解池中包含若干个解,每次交叉的过程从解池中挑选两个解作为父代解,使用交叉算子生成新的子代,子代经过局部搜索优化之后重新放入解池,然后使用一定规则从解池中删除某些解。

种群进化中有两个重要的部件,一个是交叉算子,另一个是局部搜索算法负责个体的优化提升,负责生成新的子代解。交叉算子的要求是继承两个父本中的优点,但是要两个父本保持距离。在 VRP 问题中对于局部搜索的要求是高效快速,因为局部搜索通常是算法中占用整体执行时间最长的一个算法,局部搜索的效率直接影响着种群进化的速度。

在 VRP 历史上对于交叉算子的研究有很多,交叉算子的设计方案可以有很多种,但是设计一个高效并且优秀的交叉算子难度还是不小。通过研究可以发现,一个优秀的交叉算子的成功,不是因为算子的复杂性,只是因为其抓住了问题的主要矛盾,设计思路往往非常简单。以下介绍两种典型的交叉算子: Nagata 在 2010 用在 VRPTW 上的 EAX<sup>[13]</sup>和 Vidal 在混合生成的搜索算法(Hybrid Genetic Search)中使用的 OX 交叉算子<sup>[24]</sup>。

(1) Nagata 提出的 EAX 交叉算子(The Edge Assembly Crossover)<sup>[13]</sup>,最初是用于求解 TSP 问题,在 TSP 问题上取得良好效果之后,又将这一算子扩展到多个 VRP 变种问题上,也取得了显著成功。在论文发表时刷新了当时将近一半的最优解。

EAX 交叉过程中首先给来自两个父本的边集进行分类, 去掉两个父本中相同的边, 形成差异边集, 差异边集关注的对象是父本中只出现在某一个解中的连边。在差异边集中通过一定的范围的边集与其中一个父本进行交叉。

(2) Vidal 使用的 OX 交叉算子同样是两个父本解<sup>[24]</sup>, 思路借鉴 TSP 的回路构造方法<sup>[16]</sup>, 在一个解中将仓库排除在外, 把所有顾客排列成一个染色体。第一步用两个解构造了两个染色体, 子代的染色体首先在第一个染色体中继承一个连续的片段, 放入子代相同的位置。然后按照第二个染色中顾客出现的顺序, 将子代中缺少的顾客依次填入子代的染色体。子代的染色体构造结束之后, 需要使用 split 算法将染色体分成单条路径, 此过程中使用了动态规划算法, Vidal 在此算法中的贡献是将原来复杂度为 $O(n^3)$ 的算法降低为 $O(n)$ 。

除了经典的遗传的框架之外, 还有一些其他的框架。例如通过学习蚂蚁觅食的过程, 通过使用在优质解附近留下信息素的方式, 引导算法朝着最优希望改进的空间进行探索<sup>[29][30]</sup>。

国内关于 VRP 的研究集中在 VRP 的变种问题, 例如黄楠研究了多个 VRPTW 的变种, 主要研究了多行程的 VRP 中的七个变种问题<sup>[31]</sup>。在经典的 VRP 中加入动态信息<sup>[32]</sup>。在 VRP 中考虑配送人员的资源分配的问题, 苏欣欣在此领域研究了四类变种问题<sup>[33]</sup>。在经典 VRP 上的方法创新方面, 宋亮研究近似算法求解该问题<sup>[34]</sup>, 宋强设计了一种群体智能优化算法<sup>[35]</sup>。

### 1.2.3 VRP 最新研究动态

VRP 领域研究的问题的规模在不断地变大, 最近的一些新的研究扩大了问题求解算例的启发式算法的研究上, Vigo 等人目前研究的算例规模可以达到 10000 个顾客以上<sup>[36][37]</sup>。

2022 年结束的 12th-DIMACS 挑战赛中, VRPTW 赛道中的第一名的求解器 HGS\_VRPTW 的代码已经开源<sup>1</sup>, HGS\_VRPTW 算法是基于 Vidal 的 HGS 算法<sup>[24]</sup>进

---

<sup>1</sup> <https://github.com/ortec/euro-neurips-vrp-2022-quickstart>

行开发的，在 HGS 算法上加入了求解时间窗部分的处理。由此可以看出 HGS 算法框架解决 VRP 相关问题的潜力还值得进一步挖掘。

在 2022 年结束的 EURO-NeurIPS Challenge<sup>6</sup> 中，加入了动态的 VRPTW，求解器事先不知道顾客的派单信息，随着算法的运行逐步从环境中获取这些信息。Soeffker 等人也做了相关的研究<sup>[38]</sup>，逐步发布求解算例中的不确定信息。为了引入基于学习类的算法的发展，需要大量的训练数据作为模型的输入，为此 Quecoga 等人建立了基于学习算法的标准化 CVRP 基准测试集，该测试集提供了 10000 个已知最优解用于训练机器学习的模型<sup>1</sup>。

## 1.3 研究对象及目标

VRPTW 是一个经典的组合优化问题，它可以应用到物流配送、城市交通等方面，具有广泛的实际应用价值。VRPTW 问题不仅要考虑顾客的需求、车辆容量约束等因素，还需要考虑顾客和仓库时间窗口的限制。在实际应用中，合理安排车辆的配送路径，使得总成本最小化，是一个具有挑战性的问题。

本文的研究目的是通过启发式优化算法求解 VRPTW 问题。具体来说，本文在路径数量最小化问题和路径总长度最小化问题这两个问题，提出对应的解决方案。当前研究中的局部搜索算法在求解大规模 VRPTW 时还依旧存在挑战，特别是在求解路径数量最小化问题中，存在求解过程时间消耗长等问题。本论文旨在寻找高效的算法来解决 VRPTW 问题，以减少车辆的运行成本，提高配送效率，并为物流行业的发展做出贡献。

## 1.4 论文结构安排

第一章介绍了 VRPTW 问题的研究背景与意义，阐述了 VRP 问题的研究现状，介绍了精确算法和启发式算法两种优化方法。

第二章介绍了带有时间窗的车辆路径问题的描述和分析。该章节首先介绍了

---

<sup>1</sup> <https://openreview.net/pdf?id=yHiMXKN6nTl>



VRPTW 的问题描述,包括顾客的容量约束、顾客的时间窗的约束以及车辆容量限制。接着,根据约束给出了 VRPTW 的符号定义和数学模型。确定了本文的研究的两个主要的方向为 VRPTW 两个问题:路径数量最小化问题和路径总长度最小化问题。

第三章介绍了路径数量最小化问题的求解方法。详细介绍了自适应加权搜索算法(AWLS)的总体框架,对 AWLS 中涉及的关键技术进行了详细说明。包括初始解的生成、顾客加权的路径移除算法、路径加权的修复算法、分支限界的弹射算法、带有禁忌表的扰动算法和探索路径数的最大下界的方法,并对算法的求解结果进行了分析。

第四章详细描述了路径总长度最小化问题的求解算法。该章节首先介绍了混合局部搜索算法(HLS)的总体框架,然后详细讲解了种群集合初始化、简单局部搜索、寻找最佳种群、改进的 SISR 算法、改进的 EAMA 算法和混合局部搜索算法等关键步骤。最后对算法进行了实现及单元测试。最后,本章对算法的结果进行了分析。

第五章是总结与展望。该章节首先总结了本文的主要内容以及结论,然后总结了本文的主要创新点,并且对未来工作进行了展望。

## 2 带有时间窗的车辆路径问题描述与分析

### 2.1 VRPTW 的问题描述

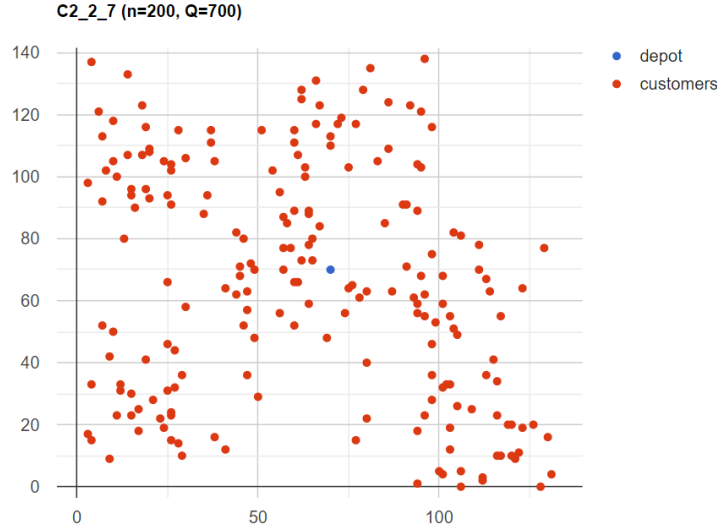


图 2.1 VRPTW 输入的示意图（算例 C2\_2\_7）<sup>1</sup>

VRPTW 来源于实际的物流配送的场景，在 VRPTW 中包括三个要素：中心仓库、若干需要配送的顾客和若干相同规格的车辆。经典的 VRPTW 中只包含有一个中心仓库，配送车辆需要从仓库出发，配送完沿途的所有顾客最后回到仓库。在图 2.1 中是 VRPTW 的测试算例的一个输入的示意图，图中蓝色的结点表示仓库，红色的结点表示需要配送的顾客。 $n = 200$  表示输入的算例的规模是 200 个顾客， $Q = 700$  表示一辆车的容量上限是 700。

图 2.2 中是当前的已知最优解，在最优解中一共有 6 条路径，分别用不同颜色的曲线表示，其中，虚线表示和仓库的连边，每条路径的起点和终点都是仓库。每一条路径上的顾客都由一个车辆进行配送，配送的所有顾客的需求之和不能超过车辆的容量上限。特别地，每条路径上的所有顾客都有一个时间窗，车辆到达每一个顾客的时间不能晚于该顾客时间窗的右端点。

<sup>1</sup> [http://vrp.atd-lab.inf.puc-rio.br/index.php/en/plotted-instances?data=C2\\_2\\_7](http://vrp.atd-lab.inf.puc-rio.br/index.php/en/plotted-instances?data=C2_2_7)

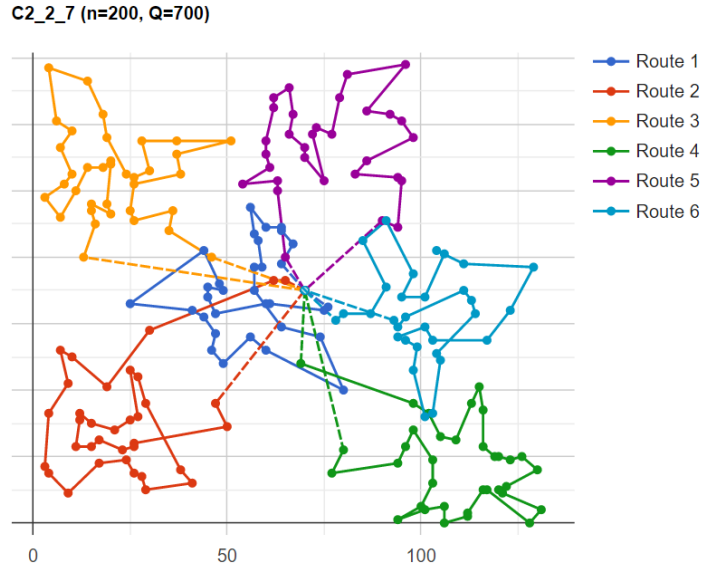


图 2.2 VRPTW 的当前最优解示意图（算例 C2\_2\_7）<sup>1</sup>

以下将介绍 VRPTW 问题的详细定义，与其他 NP-hard 问题的数学模型一样，VRPTW 的求解模型，本文从已知，约束，决策，目标四个方面对这三个要素进行详细说明。

#### 2.1.1.1 已知条件

中心仓库的位置坐标，仓库的时间窗，即仓库打开和关闭两个时刻构成的时间区间。对于每一个顾客，已知的条件包括顾客的位置坐标，顾客的货物需求量，服务该顾客所需的时间，顾客的时间窗，即顾客希望配送的最早时间和最晚时间。对于所有车辆，已知所有车辆的数量的上限，每一辆车拥有属性载重上限。通过这些已知条件，可以计算出，任意两个顾客的距离，使用欧氏距离表示。因此 VRPTW 是定义在完全图上的，任意两个顾客以及顾客和仓库之间都是可达的。

#### 2.1.1.2 约束条件

中心仓库要求所有车辆都从仓库出发，最后回到仓库。车辆在仓库出发的时间不能早于仓库的打开时间（即时间窗的左端点），不能晚于仓库的关闭时间（即仓库的时间窗右端点）。车辆的约束包括数量限制和容量限制。数量限制是指配送方案使用

<sup>1</sup> [http://vrp.atd-lab.inf.puc-rio.br/index.php/en/plotted-instances?data=C2\\_2\\_7](http://vrp.atd-lab.inf.puc-rio.br/index.php/en/plotted-instances?data=C2_2_7)

的所有车辆不能超过车辆的数量限制。车辆的容量限制是指每一辆车配送的所有货物不能超过容量限制。配送车辆的数量与配送方案中路径的数量相等,每一辆车负责配送一条路径上的顾客,每一条路径上顾客的货物需求量不能超过车辆的容量上限。对于顾客来说,每一个顾客都只能由一辆车配送一次。配送的货物不能拆分到多个车辆进行配送。车辆到达每一个顾客的时刻,不能晚于该顾客时间窗的右端点,但是可以早于时间窗的左端点。车辆到达一个顾客之后,需要消耗该顾客所需的服务时间之后才能离开该顾客。服务时间开始的时刻不能早于时间窗的左端点,所以如果车辆到达时刻早于顾客时间窗的左端点,需要等待一段时间。这里将配送的时间和距离进行了单位时间上的统一,任务车辆的速度是单位 1,即路径和配送时间可以直接做加法运算。

### 2.1.1.3 决策变量

该问题需要决策的是所有顾客的路径,一个完整的解决方案中需要包括配送的所有路径,每一条路径上的顾客按照配送顺序依次给出。

### 2.1.1.4 优化目标

VRPTW 有两个优化目标,第一个是最小化所有路径的数量,由于路径数量与车辆数相同,该目标也可以表述为最小化所用的车辆的数量。第二个优化目标是最小化所有路径的总长度。最小化路径的数量的意义,对于一家订货量不断增加的物流公司来说,可以减少公司所需的车辆数量,或者说同样的车辆可以服务更多的顾客。而最小化路径的总长度,意味着同样的顾客和车队规模,可以使得行驶距离更小,可以减少燃料消耗和缩短配送顾客交付的时间,可以帮助降低运营成本和改善顾客体验。

## 2.2 VRPTW 的符号定义和数学模型

### 2.2.1.1 符号定义

已知:在有向图 $G(V, A)$ 上,其中 $V = V' \cup \{0\}$ 是顾客和仓库的并集。对于每个顾客 $v \in V'$ 具有需求 $q_v$ ,顾客的时间窗 $[e_v, l_v]$ ,服务时间表示为 $st_v$ 。仓库(由 $v_0 = 0$ 表示)可以被视为需求 $q_0 = 0$ 且服务时间 $st_0 = 0$ 的特殊顾客。一条路径表示为 $\{v_0, v_1 \dots v_n, v_{n+1}\}$ ,其中起点和终点顶点 $v_0 = v_{n+1} = 0$ 都表示仓库。结点是仓库和顾

客的统称，既可以指代仓库也可以指代顾客。边集 $A$ 是一个有向边的集合，其中从结点 $v$ 到结点 $w$ 的距离由 $c(v,w)$ 表示。车辆的容量上限用 $Q$ 表示的车辆。所以车辆的集合用 $K$ 表示。

决策：两组决策变量 $x_{ij}^k$ 和 $T_{ik}$ 。如果车辆 $k$ 从顾客 $i$ 行驶到 $j$ ，则定义 $x_{ij}^k = 1$ ，否则 $x_{ij}^k = 0$ 。 $T_{ik}$ 表示车辆 $k$ 服务的顾客 $i$ 的开始时间。

### 2.2.1.2 数学模型

VRPTW 问题定义的数学模型可以表述如下：

$$\min f_1 = |K| \quad (2.1)$$

$$\min f_2 = \sum_{i \in V} \sum_{j \in V} \sum_{k \in K} c_{ij} x_{ij}^k \quad (2.2)$$

$$\sum_{k \in K} \sum_{j \in V, j \neq i} x_{ij}^k = 1, \forall i \in V \quad (2.3)$$

$$\sum_{i \in V, i \neq j} x_{ij}^k - \sum_{i \in V, i \neq j} x_{ji}^k = 0, \forall k \in K, j \in V \quad (2.4)$$

$$\sum_{j \in V} x_{0j}^k = 1, \forall k \in K \quad (2.5)$$

$$\sum_{i \in V} x_{i,n+1}^k = 1, \forall k \in K \quad (2.6)$$

$$T_{ik} + st_i + c_{ij} - T_{jk} \leq (1 - x_{ij}^k)M, \forall k \in K, (i, j) \in A \quad (2.7)$$

$$e_i \leq T_{ik} \leq l_i, \forall k \in K, i \in V \quad (2.8)$$

$$\sum_{i \in V} q_i \sum_{j \in V, j \neq i} x_{ij}^k \leq Q, \forall k \in K \quad (2.9)$$

$$x_{ij}^k \in \{0,1\}, \forall k \in K, (i, j) \in A \quad (2.10)$$

$$T_{ik} \geq 0, \forall i \in V, k \in K \quad (2.11)$$

目标函数在公式（2.1）和（2.2）中定义， $f_1$ 定义为最小化路径数量以及 $f_2$ 定义为最小化所有路径的总长度。公式（2.3）中的约束表示每个顾客 $i$ 的出度都是1。公式（2.4）表示对于任意的顾客 $j$ 的入度等于该顾客的出度，公式（2.3）和公式（2.4）共同约束了一个顾客只能被服务一次。公式（2.5）和公式（2.6）表示每辆车都经过一次仓库，确保每辆车从仓库出发，并在为顾客服务后返回仓库。公式（2.7）表示的约束条件确保每辆车应有足够的时间在两个连续顾客之间行驶，其中 $M$ 是足够大的正整数。同时公式（2.7）还可以消除解中出现子回路（不经过仓库的回路）的情

况。如果存在子回路，那么在子回路上的某一个顾客必然要违反这一条约束，即到达时间不是严格升序的。公式(2.8)表示约束条件顾客必须在其时间窗口内获得服务，即开始服务的时间 $T_{ik}$ 要满足顾客的时间窗需求。公式(2.9)用于表示车辆的容量约束，确保不会超过车辆的容量 $Q$ 。公式(2.10)和(2.11)定义了变量的取值范围。

## 2.3 带有时间窗的车辆路径问题分析

从 VRPTW 的问题定义可以看出，这个问题的约束很多，求解的目标明确。因此可以将问题的分析分为满足约束的思路和优化目标思路两类。

### 2.3.1 约束满足

对于仓库以及顾客的访问次数的约束比较好处理，可以构造一个合法的不存在子回路的初始解，然后在后续的变换之中不去破坏这些约束就可以，这个是容易做到的。比较难以处理的是关于车辆的数量上限约束，车辆的容量上限约束，以及车辆的时间窗约束。

针对车辆的数量上限的约束，可以专门设计一个算法用来减少可行解中的路径数量，在达到输入的约束要求之后就停止。当获得一个符合路径数上限约束的可行解之后，如何在后续的搜索中一直保持该约束不被违反，有两个思路可以考虑。第一种思路是在后续的搜索过程中不允许解的路径数量发生变化，一直保持这条约束是成立的，这样的做法是可行的，但是局部搜索可能因此受到限制，因为不允许暂时违反这条约束，容易让搜索陷入停滞，从而不能探索更多的解空间。第二种思路是允许在搜索过程中路径数量发生改变，在后续搜索的过程中去尝试修复这个解。这样做虽然放松了车辆数上限的约束，带来的好处是使得一个解更容易合法化，在搜索的过程中，如果遇到不合法的解，可能违反车辆容量约束或者顾客时间窗的约束，可以通过增加路径来使得这两种违反消除。

判断车辆的数量上限是否违反约束比较简单，可以直接对已有的可行解中的路径数量进行统计即可，但是统计车辆的容量约束和时间窗约束稍显复杂。对于这两个约束的计算方式，可以使用定义进行计算。即对于每一条路径都去从头到尾计算一遍到达每个顾客的时刻，然后查看是否满足时间窗的需求，对于容量约束也可以将一条

路径上的顾客需求进行累加，这样计算的时间复杂度就是 $O(n)$ 。而在局部搜索算法中，一般不使用定义直接计算和判断是否违法。局部搜索算法从一个质量较差的初始解出发，在邻域范围内有多项式复杂度和解可以挑选，如果对于邻域内的每一个解都使用定义进行评估解的质量，这样的时间复杂度是不可接受的。所以需要对邻域内的解进行增量评估以加速局部搜索算法的运行。

首先是容量惩罚的增量评估，需要将每条路径单独考虑。动态维护每条路径当前经过的所有顾客的需求量，然后再针对不同的邻域动作，分析每种邻域动作对于路径内顾客的更改，动态更新路径内的需求总和。采用这样的方式对于有限个数顾客的移动和交换都可以做到 $O(1)$ 的时间复杂度，但是对于 2-opt\* 邻域动作还是不能做到常数时间复杂度的代价评估，具体求解思路在 3.4 节详细介绍。

对于时间窗的代价评估更加复杂一些，因为时间窗的定义中包含了路径的方向信息，每一个顾客的到达时间与前序的所有顾客都有关系，所以一个节点的移动会导致产生循环影响，从而难以做到常数时间复杂度的评估，这一部分的处理方法在算法 3.4 节详细进行说明。

## 2.3.2 目标优化

根据 2.1.1.4 小节中对于优化目标的定义，需要针对这两个目标分别进行优化。这两个目标的求解难度体现在不同的方面。路径数量最小化问题可以将优化问题转为判断，对于一个给定的解决方案每次减少一条路径，把路径数作为一个约束进行求解，难点在于找到一个合法解。而路径总长度最小化则不能方便地转化为判定问题，因为路径的长度是一个连续的量，很难找到一个合适的值作为限定。

## 2.4 本章小结

这一章详细介绍了 VRPTW 的问题描述以及问题的分析。问题描述部分从已知、约束、决策和目标四个维度进行详细的说明和解释。对于 VRPTW 的多种约束进行了分类的分析，对于路径数量上限的约束满足进行了分析，给出了初步的解决思路，对于对容量和时间窗的约束满足问题进行了简单的分析。

### 3 求解路径数量最小化问题的自适应加权搜索算法

在 VRPTW 中, 最小化路径数作为一个优化目标的意义在于, 通过减少配送所需的车辆数, 可以降低运输过程的成本, 提高资源利用率。具体来说, 对于物流公司如果需要服务的顾客数量很大, 那么可以通过将这些顾客分配到尽可能少的路径中, 这样就可以减少车辆数量, 以优化物流公司的车辆费用的支出。

最小化路径数问题中存在的限制和挑战是在保证路径数量最小的情况下, 还需要满足所有顾客的时间窗口限制和容量约束。相当于在已有的强约束的基础上, 又加上了一条新的强约束。因此在设计启发式算法时候, 要针对这样的强约束提出专门的应对策略。针对问题的约束较强, 在搜索过程中经常出现局部最优等问题, 本文中提出了自适应加权局部搜索算法来加以优化。以下具体介绍这一算法的运行流程。

#### 3.1 自适应加权局部搜索算法总体框架

算法使用了一种自适应的基于加权的局部搜索(Adaptive Weighting-based Local Search, AWLS), 用于求解 VRPTW 中路径数量最小化的问题。AWLS 从一个启发式的初始解构造算法开始, 不断重复删除路由, 直到达到时间限制。返回可以优化到的最小路径数的解。

AWLS 为顾客和路径分别设置了一个权重引导局部搜索。随着算法的运行, 权重不断增加, 使得解空间的地貌也在不断变化。每个顾客和路线都与反映顾客和路线重要性的自适应权重相关联。这种多样化机制使得能够逃离局部最优, 从而提高了算法的搜索效率。

算法的主要流程如下: 首先使用贪心构造的初始解算法生成一个初始解, 之后调用顾客加权的路径移除算法对初始解进行路径数量最小化过程, 每次调用路径移除算法都可以在当前解中移除一条路径。顾客加权的路径移除算法每次移除一条路径, 将这条路径上的顾客放在弹射池(Eject Pool, *EP*)中保存, 算法的目标就是清空*EP*。对于*EP*中的顾客, 如果可以通过贪心插入将顾客放入当前解中, 就直接插入, 如果贪心插入失败, 形成的非法解, 算法调用路径加权的修复算法对非法解加以修复, 如



果修复算法不能使得局部解合法，则在路径加权的修复算法返回的最优解中调用分支限界的弹射算法来使得算法强制修复到合法的状态。在此之后调用带有禁忌表的扰动算法对局部解的格局加以调整，增加局部解的多样性，以增加后续插入算法的成功概率。

## 3.2 初始解生成

---

### 算法 3.1 初始解生成

---

**输入：** 所求解算例的信息，包括仓库、顾客以及车辆的所有信息

**输出：** 合法的初始解 $S_{init}$

---

```
1:  $S_{init} \leftarrow \emptyset$  /* 算法运行开始，初始解路径集合为空 */
2:  $EP \leftarrow V$  /* 将所有顾客都放入  $EP$  中 */
3: while  $EP \neq \emptyset$  do
4:   从  $EP$  中随机挑选一个顾客  $v_{in}$ ，并且从  $EP$  中移除  $v_{in}$ 
5:   在  $S_{init}$  中为  $v_{in}$  挑选最佳的插入位置  $p$ 
6:   if 将  $v_{in}$  插入到  $p$  没有违反约束 then
7:     将  $v_{in}$  插入到位置  $p$ 
8:   else
9:     创建一条新的空路径放入  $v_{in}$ 
10:  end if
11: end while
12: return  $S_{init}$ 
```

---

算法 3.1 描述了初始解生成算法的详细过程。初始解生成算法的输入是整个算法的输入，算法从算例输入的文件中读取信息，包括车辆、仓库和顾客的所有信息。并且经过处理计算出任意两个顾客之间的欧氏距离，以及仓库和所有顾客之间的欧氏距离。算法的输出是一个合法的初始解，使用算法 3.1 中描述的算法进行构造初始解可以控制路径的数量不会很大。一个合法的初始解由若干条路径构成，覆盖了所有顾客，满足仓库以及顾客对于访问次数的要求。

算法运行初始化,  $S_{init}$  中没有路径, 路径集合为空 (第 1 行)。定义一个弹射池  $EP$  (Ejection Pool),  $EP$  用来存放当前未被安排路径的顾客。初始状态下, 所有顾客都处于弹射池中 (第 2 行)。

从算法的第 3 行到 11 行, 描述了如何清空  $EP$ 。首先从  $EP$  中随机挑选一个顾客  $v_{in}$  (第 4 行), 在解  $S_{init}$  中考虑所有可以放入的位置 (第 5 行), 即放入  $v_{in}$  之后不违反容量约束和时间窗的约束 (第 6-7 行), 如果这样的位置有多个, 选择路径长度增加最小的位置将  $v_{in}$  插入进去, 路径增长变化量计算方法为:  $\Delta c = c(v_{in}, v_{in}^-) + c(v_{in}, v_{in}^+) - c(v_{in}^-, v_{in}^+)$ , 如果  $S_{init}$  中没有不产生约束违反的合法位置, 需要新建一条路径专门用来放置  $v_{in}$  (第 8-9 行)。当  $EP$  被清空之后, 算法返回得到的初始解 (第 12 行)。

在设计初始化算法的时候, 需要注意的是引入随机性, 优秀的启发式算法不应该因为初始解的不同而产生较大优度上的差异。一个质量差的初始解可能对算法的求解时间产生影响, 算法最终收敛的优度的差异不应该很大。为了增加初始化的多样性, 同时在初始化的过程中尽量利用输入算例的信息, 算法在挑选顾客的顺序部分, 增加了多种挑选规则: (1) 按照顾客和仓库的夹角的顺序依次挑选顾客。(2) 按照顾客需求量从大到小的顺序挑选顾客。(3) 按照顾客的服务时间从长到短的顺序挑选顾客。(4) 按照顾客时间窗右端点从小到大的顺序挑选顾客。

### 3.3 顾客加权的路径移除算法

算法 3.2 描述了顾客加权的路径移除算法, 该算法可以在一个可行的解中减少一条路径。算法的输入要求是一个合法的解, 这个合法的解可以调用初始解生成算法去产生, 也可以是调用路径移除算法得到的返回值。路径移除算法会被反复调用, 一直删除无法减少路径数为止。路径移除算法只在运行的一开始移除了一条路径, 在后续的优化过程中, 不再允许路径数量发生变化。这种方法, 相当于把一个优化问题, 转化为了一个判定问题, 即将路径数视为一条约束, 在求解过程中一直满足这条约束, 该方法也是启发式算法中一种常见的优化手段。

算法变量初始化, 首先将传入的  $S$  赋值给  $S^*$ , 在后续减少路径数的过程中使用  $S^*$

进行搜索，因为无法预测能否删除一条路径，如果算法运行失败，则需要将初始传入的解直接返回，因此这里做了一个备份（第 1 行）。算法为每一个顾客都设置了一个权重， $W_{cus}$  是一个一维数组，数组中保存了每一个顾客的权重。该权重的调整方式在下面会进行详细介绍。顾客权重的初始值都设置为 1（第 2 行）。

路径移除算法首先从初始解中随机挑选一条路径，将这条路径从  $S^*$  中删除，之后将该路径中的顾客放入  $EP$  中保存（第 3 行）。此时算法的目标转化为清空  $EP$ ，当  $EP$  中不再有顾客的时候，相当于路径移除算法就成功删除了一条路径。

算法 3.2 的第 4-15 行，详细描述了清空  $EP$  的全过程。该过程可以大致分为三个阶段，首先是贪心插入（第 5-6 行），如果贪心插入失败了，接下来进行修复算法（第 7-8 行），如果修复算法再次失败，最后调用弹射算法和扰动算法。

贪心插入部分（第 5-6 行）与初始解生成过程类似，算法迭代地在  $EP$  中随机挑选顾客  $v_{in}$ （第 5 行），尝试插入到当前解  $S^*$  中。与初始解生成过程不同的是，如果最佳的插入位置违反容量或时间窗口约束，算法不会为  $v_{in}$  创建专用的新路径。算法会直接将  $v_{in}$  放入这个最佳的插入位置。这产生了不可行的解  $S^*$ 。

在贪心插入部分产生非法解之后，算法调用路径加权的修复算法（第 7-8 行），对当前解进行合法化。通过迭代的邻域搜索去消除时间窗口约束的违反和容量约束的违反。如果路径加权的修复算法可以消除  $S^*$  的约束违反。那么算法就可以进行下一轮的贪心插入。如果修复算法不能成功修复解，算法将调用弹射算法，使得非法的解  $S^*$  通过弹射顾客的方式达到合法的状态。路径加权的修复算法的具体过程在 3.4 节中进行详细介绍。

弹射算法将一定数量的顾客从解  $S^*$  中转移到弹射池（第 10 行），用来消除容量约束违反和时间窗口约束违反。每次从  $EP$  中取出顾客  $v_{in}$  插入  $S^*$ ，将权重  $W_{cus}(v_{in})$  设置为  $W_{cus}(v_{in}) + W_{up}^1$ 。调用分支限界的弹射算法（Eject）从非法解中弹出  $m$  个顾客，顾客的集合用  $EJ$  表示， $EJ$  中的每一个节点的权重值增加  $W_{up}^2$  个单位。弹射算法中  $m$  个顾客如何选择，在下面的 3.5 节进行详细介绍。

顾客加权的移除算法的灵感来自 Nagata 的多样化标准和挤压算法<sup>[12]</sup>（Diversification Criterion and Squeeze Procedure, DS），DS 算法为每一个顾客设置了

---

## 算法 3.2 基于顾客加权的路径移除算法

---

输入：一个合法解 $S$

输出：路径数更少的解 $S^*$

---

```
1:  $S^* \leftarrow S$ 
2:  $W_{cus}(v_i) \leftarrow 1, \forall i \in V'$  /* 将所有顾客的初始权重设置为 1 */
3: 从 $S^*$ 随机挑选一条路径删去，将路径上所有顾客放入 $EP$ 
4: while  $EP \neq \emptyset$  and 计时器没有超时 do
5:     从  $EP$ 中随机移除顾客 $v_{in}$ ,  $W_{cus}(v_{in}) \leftarrow W_{cus}(v_{in}) + W_{up}^1$ 
6:     将 $v_{in}$ 插入到 $S^*$ 中惩罚最小的位置
7:     if  $P_{en}(S^*) > 0$  then
8:          $S^* = RouteWeightedRepair(S^*)$ 
9:         if  $P_{en}(S^*) > 0$  then
10:             $EJ = Eject(S^*)$ 
11:             $W_{cus}(v_i) \leftarrow W_{cus}(v_i) + W_{up}^2, v_i \in EJ$ 
12:            将 $EJ$ 中的顾客从 $S^*$ 中移除，放入 $EP$ 中
13:             $S^* = Perturb(S^*)$ 
14:         end if
15:     end if
16: end while
17: if  $P_{en}(S^*) > 0$  then
18:     return  $S$ 
19: else
20:     return  $S^*$ 
```

---

惩罚值，在搜索的过程中动态调整。DS 算法与顾客加权的路径移除算法的不同点是：

(1) 顾客权重的定义不同。DS 算法中对顾客定义了惩罚值，该惩罚值调整的时机仅在贪心插入的顾客被再次弹射出局部解中的时候进行调整，在顾客放入 $EP$ 的时候不对其惩罚值进行调整。(2) DS 算法采用栈的数据结构挑选顾客进行后续的插入操作，

顾客加权的路径移除算法采用的是随机挑选顾客的策略。(3) DS 算法的弹射算法需要为 $v_{in}$ 试探解中的每一个位置，每次都进行一个弹射操作，这是一个极其耗费时间的算法。顾客加权的路径移除算法是在修复算法结束之后获得的局部最优解进行弹射操作。不仅提高了算法运行的效率，也使得弹射算法的时机更加合理。经过修复算法还不能调整到合法状态下的顾客，一般都是难以处理的，需要将这部分顾客进行识别加以特别关注。在修复算法进入局部最优的时候，这部分顾客恰好被暴露出来，然后通过弹射算法将他们排除在局部解之外，进行单独处理。

### 3.4 路径加权的修复算法

在修复算法中，首先要对一条路径违反约束的程度进行量化。一个非法的解的量化进行了重新的定义。算法为每一个解定义了一个惩罚值，用于量化每一个非法解违反时间窗约束和车辆的容量约束的程度。需要注意的是，这章节提到的解是求解问题的一个规模更小的一个子问题，即在解中没有包含所有的顾客，因为有若干顾客被放置在了 $EP$ 中。这种设计相当于求解了一个原本问题更小的问题，在一定程度上减小了求解问题的难度，也是弹射池这种求解结构的巧妙之处。修复算法是使用经典邻域动作进行修复的，这里用到的邻域动作有 2-opt\*[19]，exchange 以及 out-relocate 动作[18]。算法在修复的过程中，只考虑有约束违反的路径，不考虑没有约束违反路径间的邻域动作。在修复算法中最占用时间的操作就是对邻域动作进行评估，邻域评估为修复算法指引了惩罚值下降的方向。所以如何快速评估一个邻域动作带来的收益非常重要。

一个解的约束违反可以分为两个部分，对于时间窗约束的违反，另一个是对于容量约束的违反。所以惩罚值函数的定义要包含两个部分。 $P_{en}(S)$ 等于 $P_{tw}(S)$ 和 $P_c(S)$ 两部分构成，因为容量和时间窗的量纲不统一，故加一个系数 $\alpha$ 加以调和。

$$P_{en}(S) = P_{tw}(S) + \alpha P_c(S) \quad (3.1)$$

$$P_{tw}(S) = \sum_{r \in S} w_r(r) \times P_{tw}(r) \quad (3.2)$$

$$P_c(r) = \sum_{v \in r} q_v - Q \quad (3.3)$$

一个解的时间窗惩罚来自解中所有路径时间窗惩罚的总和，容量惩罚同理，所以

只要定义好单条路径的时间窗惩罚和容量惩罚，再使用公式（3.1）就可以计算一个解的惩罚值。

根据公式（3.3）容量惩罚的定义，算法只需要为每一个路径维护一个当前在路径上的所有顾客的需求量总和，然后根据每种邻域动作的改变，分别计算路径移出顾客和移入顾客的差值，就可以知道路径上需求量的变化。当路径上移动动作是在有限数量顾客之间进行的时候，容量的惩罚都可以在 $O(1)$ 时间复杂度中计算得到。对于2-opt\*动作，交换的是两个路径的后半部分，这时仅维护每条路径上的顾客的需求量之和就不能做到 $O(1)$ 复杂度计算变化量，还需要为路径上的每一个顾客维护一个累加的需求量，记录从路径开始到该顾客的需求量之和是多少。计算过程类似于数组前缀和的引用，可以在 $O(1)$ 时间复杂度内，计算一个区间所有顾客的需求。

针对一个非法解来说，时间窗的惩罚相对于容量的惩罚更加难以消除，需要专门设计来加以关注。因此算法为每一条路径定义了一个权重 $w_r$ ，每一条路径的惩罚值要与该条路径的权重相乘之后再相加得到整个解的时间窗的惩罚值，详细的定义在公式（3.2）中。这样做的动机是修复算法容易陷入局部最优值，从而修复失败。当修复算法陷入局部最优，算法允许暂时地增加时间窗和容量的违反，而更进一步的对解进行修复。算法对路径的权重加以调整，此时搜索的解空间的地貌也随之发生了变化，使得搜索可以继续进行。下面详细介绍路径权重在计算邻域动作的收益时如何使用，以及路径权重何时调整。

如图 3.1 所示，以路径间 exchange 邻域动作举例，将路径R1上的顾客A与路径R2上的顾客B进行交换。假设R1路径的权重是 $W1$ ，R2路径的权重是 $W2$ 。路径R1在变换前后时间窗惩罚变化量为 $(P_{tw}(R1') - P_{tw}(R1)) * w_r(R1)$ ，同理路径R2的在变换前后时间窗惩罚值的变化量为 $(P_{tw}(R2') - P_{tw}(R2)) * w_r(R2)$ 。

对于不带权重的时间窗惩罚快速评估，算法采用了 Nagata 在 EAMA 算法中的快速评估技术<sup>[13]</sup>。该技术的主要思想是对于到达顾客的配送时间进行修正。在一条路径中 $\{v_0, v_1, v_2, \dots, v_x, v_{x+1}\}$ ，如果到达 $v_x$ 的时间 $t_1$ 超过了 $v_x$ 的最晚配送时间 $l_{x_v}$ ，计算 $v_{x+1}$ 顾客的到达时间不是从 $t_1$ 开始计算，而是在 $l_{x_v}$ 开始计算。这样的计算虽然与实际情况不符合，但是可以带来实际的好处：（1）首先不是对于路径的合法性判断产生影

响，即一条有惩罚的路径，用这种方法进行计算惩罚值一定是大于 0 的，反之，一条没有惩罚的路径惩罚值的计算也一定是 0。(2) 到达时间的修正第二个好处就是保护良好的路径子路径。还是假设顾客 $v_x$ 的到达时间不满足需求，将到达时间进行修正之后，可以让后续顾客违反时间窗约束的可能性降低，如果不进行修正，时间窗违反就会进行累积的影响，导致后续所有的顾客都违反时间窗约束。而有可能此时顾客 $v_x$ 后面的子路径刚好是一个良好的结构，这样就产生了误判，因此造成修复算法可能极易将这个子路径破坏掉。(3) 使用这种定义方式，路径间的邻域动作，包括两条路径之间的 2-opt\*动作，以及路径间的顾客的交换动作和移动动作都可以在 $O(1)$ 时间复杂度内计算得到。具体的计算公式可以参考 EAMA 算法中的描述。

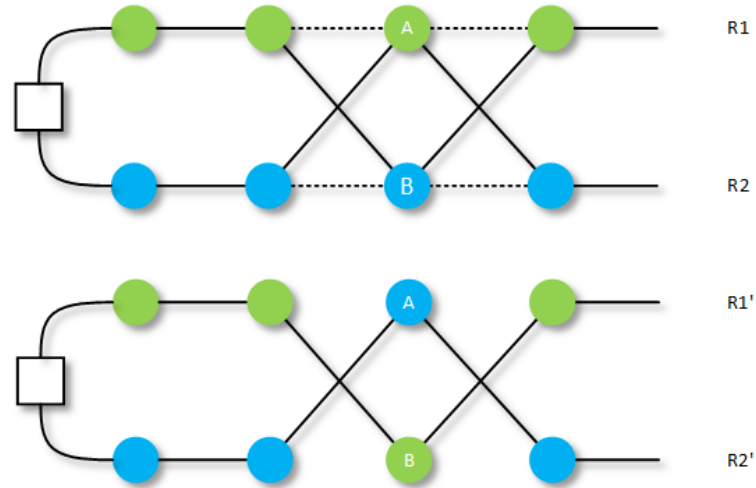


图 3.1 路径加权 exchange 动作示意图

算法 3.3 中描述了路径加权的修复算法的具体过程。算法的输入是一个非法的解，算法的输出是一个经过修复得到的局部最优解，这个解可能是非法的。修复算法如果能成功修复解，就会返回一个合法解，否则返回能得到的惩罚值最小的解。修复算法在开始运行时，将所有路径的权重置为 1（第 1 行），声明变量 $S^{best}$ ，保存修复算法过程中得到的最好解。 $stagnation$ 变量用于记录算法连续没有改进局部最优解迭代步数（第 2 行）。第 3-18 行是修复算法的核心逻辑，如果修复算法不能修复当前解，算法也不是立刻退出，在连续 $N_{step}$ 次迭代都不能改进历史最优解，算法才停止修复当前解（第 3 行）。

修复算法在当前解的邻域内找到一个惩罚值最小的解 $S^*$ ，如果 $P_{en}(S^*) < S^{best}$ ，那么更新 $S^{best}$ （第 5-6 行）。并且将 $stagnation$ 计数器置为 0，否则将 $stagnation$ 自增 1（第 7-8 行）。

---

### 算法 3.3 路径加权的修复算法(RouteWeightedRepair)

---

输入：一个非法解 $S$

输出：修复算法能得到的最优解 $S^*$

---

```
1:  $W_{rt}(r_i) \leftarrow 1, r_i \in S$ 
2:  $S^{best} \leftarrow S, stagnation \leftarrow 0$ 
3: while  $P_{en}(S^{best}) > 0$  and  $stagnation < N_{step}$  do
4:   在 $S$ 的邻域内找到惩罚值最小的解 $S^*$ 
5:   if  $P_{en}(S^*) < P_{en}(S^{best})$  then
6:      $S^{best} \leftarrow S^*, stagnation \leftarrow 0$ 
7:   else
8:      $stagnation \leftarrow 0$ 
9:   if  $P_{en}(S^*) > P_{en}(S)$  then
10:    for  $r_i \in S$  do
11:      if  $P_{en}(r_i) > 0$  then
12:         $W_{rt}(r_i) \leftarrow W_{rt}(r_i) + 1$ 
13:      end if
14:    end for
15:  endif
16:   $S \leftarrow S^*$ 
17: end if
18: end while
19: return  $S^{best}$ 
```

---

如果这个解的惩罚比当前解 $S$ 的惩罚值还大（第 9 行），说明修复算法陷入了局部最优，此时修复算法将解 $S$ 中有时间窗惩罚的路径的权重增加 1 个单位（第 10-12



行)。修复算法依旧接受解 $S^*$  (第 16 行), 在达到退出条件之后, 算法将修复过程中获得的局部最优解返回。

需要注意的是, 在修复算法的第 4 行, 算法没有评估整个邻域的所有可能的动作, 而是根据一定的规则去缩小邻域范围。邻域范围的缩减对于修复算法至关重要。优化算法效率的第一步就是要抓住主要矛盾。对于修复算法而言, 如果不加以限制, 邻域动作的范围会特别大。拿顾客规模为 1000 举例, 任意两个顾客之间包含三种动作, 这样邻域动作的数量在 $10^6$ 的数量级。而最粗糙的做法就是评估邻域内所有的动作, 然后去执行其中最好的动作, 这样的算法是低效的。这样做也是没有必要的, 每次评估百万级别的邻域动作却只执行一个动作, 是收益率极低的一种做法。启发式算法对于邻域限制方面, 也有很多方法。在 DS 算法中使用的是根据顾客排序去限制邻域<sup>[12]</sup>。具体的做法是先选中一个顾客 $v$ , 然后按照其他顾客与 $v$ 距离的从小到大排序, 只在距离的前 100 个顾客与 $v$ 进行邻域动作的评估。

DS 算法这样的邻域限制有可取之处, 但是也存在一定的问题。根据顾客距离排序的设计是合理的, 相距越远的顾客如果在同一条路径的相邻位置, 那么违反时间窗约束的概率就越大。但是关于 100 这个数量限制是存在问题的, 首先 100 这个数组是人为规定的数字, 只能根据经验设置一个数值, 这一数值不仅与输入的地理信息有关, 还与当前解的状态有关, 同一个顾客在不同的路径上, 对于邻域评估范围大小的要求是不一样的。其次对于不同规模的都取相同的值也不合理。

路径加权修复算法部分, 在邻域缩减部分进行了精细的设计, 针对不同的邻域动作给出了不同的缩减规则。修复算法首先只关注有惩罚的路径, 在进行邻域评估时, 只针对与这些路径有关的邻域动作。

对于邻域动作 2-opt\*, 交换动作 (exchange) 以及移动动作 (relocate), 都需要选择两个顾客作为基准。不失一般性, 不妨把首先选定的顾客叫做顾客 $v$ , 再次选中的顾客叫做顾客 $w$ 。2-opt\*是交换两条路径的后半部分顾客, 需要在两条路径中各自选取一个顾客作为断点, 在第一条路径中选取顾客 $v$ , 在第二条路径中选取顾客 $w$ , 之后在顾客 $v$ 和 $w$ 的位置将两条路径拆分开, 再将这两个路径进行重新连接。同理, 交换的邻域需要挑选两个顾客进行交换。移动的邻域动作需要先选定被移动顾客

$v$ ，然后为 $v$ 选择移动的目的地 $w$ ，将 $v$ 放入 $w$ 所在路径的相邻位置。如何选取顾客 $v$ 和顾客 $w$ ，以下将进行详细介绍。

对于任意一条存在惩罚的路径，顾客 $v$ 首先在该路径上选取，但是不需要考虑这条路径上全部的顾客，算法只关注路径上最后一个违反约束的顾客以及其同路径上前面的所有顾客。也就是说，对于一条路径 $\{v_0, v_1, v_2, \dots, v_x, v_{x+1}\}$ ，如果 $v_x$ 之后配送的顾客时间窗都没有违反，那么算法只在 $v_0 - v_x$ 之间选择顾客 $v$ 。

当顾客 $v$ 确定之后，顾客 $w$ 的确定与顾客 $v$ 的确定有关，按照不同的邻域动作进行不同的设计。因为对于任意一条有惩罚的路径，将节点移入该路径不能降低路径的惩罚值，所以算法不考虑移入的邻域动作（in-relocate）。对于其他类型的邻域动作分类进行介绍，这里定义符号顾客 $v^-$ ，是顾客 $v$ 同路径上的前序顾客，顾客 $v^+$ 是顾客 $v$ 同路径上的后序顾客。

（1）对于交换动作的邻域限制：对于任意一条有惩罚的路径，选择将 $v$ 换出路径，那么对于换入的 $w$ 的要求是： $w$ 与 $v^-$ 距离要比 $v$ 更近。因为如果换入的 $w$ 距离 $v^-$ 更远，并不能缓解这条路径的时间窗的惩罚，反而可能会使得该路径的时间窗惩罚进一步加大。但是这里的更近，不能仅把距离的作为排序的依据，还需要考虑顾客的服务时间。设想这样一种情景，顾客 $w$ 比顾客 $v$ 在地理位置上更靠近 $v^-$ ，但是 $w$ 的服务时间很长，将 $w$ 换入路径中依然可能很大概率造成后续顾客时间窗惩罚的增加。因此需要在计算 $v^-$ 和 $w$ 的距离的时候，加上 $w$ 的服务时间，同理，在计算 $v^-$ 和 $v$ 的时候也要加上 $v$ 的服务时间。这个信息可以在算法的预处理阶段进行处理。

（2）对于 2-opt\* 动作的邻域限制：如图 3.2 所示，对于 2-opt\* 动作，在路径 $R1$ 上选定顾客 $v$ 之后，将 $v$ 和 $v^-$ 断开，然后再挑选顾客 $w$ ，将 $w$ 和 $w^+$ 断开，之后连接顾客 $v^-$ 和顾客 $w^+$ ，连接顾客 $w$ 和 $v$ 。当选定第一个顾客 $v$ 之后，因为顾客 $v$ 以及 $v$ 之后的顾客违反了时间窗约束，这时需要挑选可以改善 $v$ 的时间窗惩罚的，要求与 $v$ 连接的 $w$ 顾客比 $v^-$ 距离 $v$ 更近。这里的更近不仅要求在地理位置更近，而是对于服务完 $w$ 顾客的时刻，加上 $w$ 到 $v$ 顾客的距离消耗之后的时间。因为服务顾客 $w$ 的结束时间是由到达顾客 $w$ 的时间相关的，但是这个时间是动态变化的，不能在预处理阶段提前进行处理。为此，算法使用顾客 $w$ 的时间窗的左端点，作为顾客 $w$ 的到达时间的估计这样的估计

不会出现漏掉可能改进的顾客 $w$ ，但是可能存在邻域过大的问题。

(3) 对于 out-relocate 动作，是将顾客 $v$ 移出到顾客所在 $w$ 的路径中，这个动作等价于 $w$ 所在路径的 in-relocate 动作，但是因为算法的顾客 $v$ 是在有惩罚的路径中选取的，不会评估没有惩罚的路径，因此只保留了顾客 $v$ 所在路径的 out-relocate 动作，而删掉了顾客 $w$ 所在的路径的 in-relocate 动作。也因此，out-relocate 动作没有很好的邻域缩减策略，依然沿用了 DS 算法中找地理位置 100 近的思路，但是对不同的算例，对 100 这个数字重新进行了修正，一个大小有 $|V|$ 个顾客的算例，算法只取最靠近顾客 $v$ 的前 $\frac{|V|}{8}$ 个顾客。

加入了邻域缩减策略之后，修复算法的运行效率得到了很大的提升。关于邻域缩减的原则有两个方面：首先是一定没有希望的邻域就肯定不要去评估，其次是有希望的邻域都不放过。而邻域缩减的核心难点就在于，如何划清有希望的邻域范围和没有希望的邻域范围之间的界限。这个需要根据具体的问题结构，甚至搜索过程中的中间状态加以分析。这也是一个高效的启发式算法往往只能解决一个或者一类专属问题，不能移植的一大原因之一。

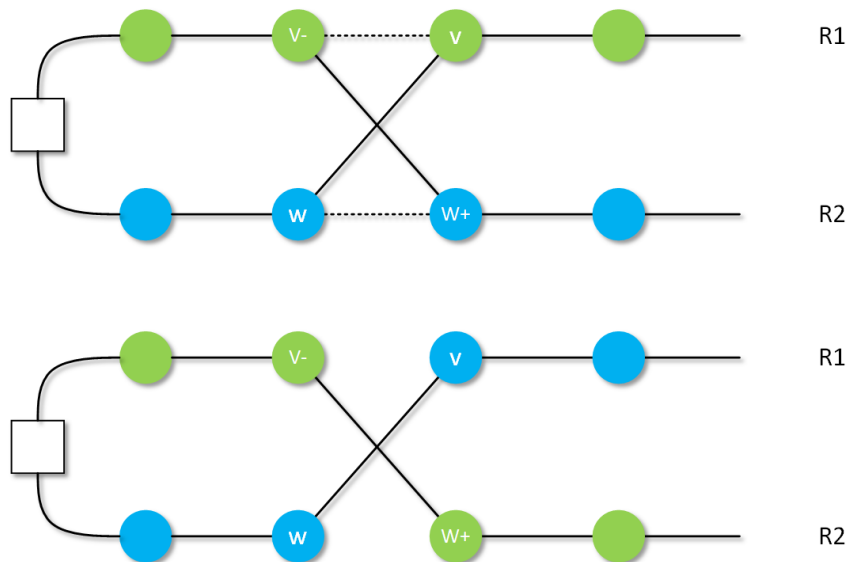


图 3.2 2-opt\*邻域动作邻域缩减示意图

### 3.5 分支限界的弹射算法

分支限界的弹射算法 (Eject) 作为顾客加权的路径移除算法, 在修复算法不能修复非法解的时候调用, 将部分顾客移除出当前解的办法来使得当前修复到合法的状态。分支限界的弹射算法是在 DS 算法的弹射池 (Ejection Pool) [39] 的基础上进行了改进。弹射算法的输入是一个非法解  $S$ , 弹射算法的返回值是需要被弹射出当前解的顾客集合  $EJ$ , 之后顾客加权的路径移除算法将被选中的顾客 ( $EJ$  中的顾客) 从当前解  $S$  中移除, 使得  $S$  被修复到合法的状态。同理这里的合法仅仅是在不包括  $EP$  中的顾客的情况下, 顾客子集  $S$  中的路径上没有时间窗惩罚和容量惩罚。

在 DS 算法中的  $Eject$  过程, 在一条路径  $r = \{v_0, v_1, v_2, \dots, v_i, \dots, v_n\}$  中, 一共有  $n$  个顾客, 在 DS 算法中需要从中挑选最多  $K_{max}$  个顾客移除出路径, 使得路径达到合法的状态。此过程如果不加以限制, 时间复杂度与  $n$  是指数关系。DS 算法使用字典序遍历的方式, 去依次查找所有的可能构成的  $EJ$  的顾客集合。遍历的过程包括固定集合的大小, 更换集合中最后一个元素 (即原文中  $i(k)$  ejected), 或者增大集合  $EJ$  的大小 (即  $i(k)$  increment) [12]。使用  $P_{sum}$  表示  $EJ$  中当前顾客的惩罚值之和,  $P_{best}$  表示当前为止找到的最小的惩罚值, 这个惩罚值对应了当前最佳的  $EJ$  集合。

针对这两个自增的过程提出了两条剪枝的策略: (1) 对于当前集合想要加入一个顾客的时候, 即  $i$  增加的时候, 如果加入顾客  $i$  超过了  $P_{best}$ , 就放弃加入顾客  $i$ 。并且剪掉此后当前集合加入顾客  $i$  之后的所有可能的组合。(2) 在  $i$  自增的过程中, 如果不能提前后续顾客的服务开始的时间, 那么就不会搜索加入顾客  $i$  的分支。

该算法的剪枝策略非常有效, 在分析该算法之后, 发现以下可以改进的地方: (1) 首先是  $K_{max}$  的定义, 有可能一条路径中仅弹射  $K_{max}$  和顾客不能使得路径合法, 需要对这种情况进行特殊的处理。(2) 弹射的顾客的数量没有进行限制, 如果仅仅找  $P_{sum}$  最小的顾客集合, 可能会经常将  $K_{max}$  顾客扔出解中, 而算法目标是清空  $EP$ , 所以这样的设计和目标是相违背的。

针对这样的分析, 分支限界的弹射算法中, 算法做了如下改进: (1) 算法对目标函数进行了重新的定义。公式 (3.4) 中将  $EJ$  的评估函数由原来的只统计所有顾客的惩罚值之和, 改为统计所有顾客的权重之和与  $EJ$  集合中顾客的数量乘积。(2) 在

3.3 节中介绍了如何调整顾客的权重，因此这里的权重不再是惩罚的意义，这里的权重表示了顾客移入移出 $EP$ 的频繁程度。故分支限界的弹射算法改为统计 $EJ$ 集中所有的顾客的权重之和。(3)当一条路径无法通过选择 $K_{max}$ 个点使其消除惩罚的时候，弹射算法会根据贪心选取规则选择顾客，对其进行合法化。基于贪心选取规则比较简单，首先按照不同的标准给顾客排序，例如先移除顾客的权重小的顾客，或者先移除能给路径惩罚值减小最多的顾客。这样依次选择顾客，直到路径合法为止。

$$f(EJ) = |EJ| * \sum_{c \in EJ} W_{cus}(c) \quad (3.4)$$

---

**算法 3.4** 分支限界的弹射算法(Eject)

---

**输入：** 一个非法解 $S$

**输出：** 弹射顾客集合 $EJ$

---

```
1:  $EJ \leftarrow \emptyset$ 
2: for  $r \in S$  do
3:    $EJ_r \leftarrow \emptyset$ 
4:   if  $P_{tw}(r) > 0$  then
5:     使用时间窗信息进行分支限界构造 $EJ_r$ 
6:     if  $EJ_r$  不是一个合法的移除方案 then
7:       使用贪心选取规则构造 $EJ_r$ 
8:     end if
9:   else if  $P_c(r) > 0$  then
10:    使用贪心选取规则构造 $EJ_r$ 
11:   end if
12:   for  $c \in EJ_r$  do
13:     将 $c$ 加入 $EJ$ 中
14:   end for
15: end for
16: return  $EJ$ 
```

---

经过以上的改进, 分支限界的弹射算法详细过程在算法 3.4 中进行了详细地描述。该算法将非法解中违反惩罚的路径分类处理。首先初始化 $EJ$ 为空集(第 1 行), 算法第 2-15 行, 遍历了 $S$ 中的所有路径, 找到所有违反时间窗惩罚的路径, 这样的路径中可能也可能同时存在容量惩罚(第 4 行), 以及仅违反容量约束的路径(第 9 行)。对于任意一条有惩罚的路径, 算法都会构造一个 $EJ_r$ 集合, 初始化为空集(第 3 行)。对有时间窗惩罚的路径调用分支限界的弹射算法得到需要弹射的顾客集合 $EJ_r$ (第 5 行), 但是由于减低时间复杂度的需求, 对于弹射顾客的个数 $K_{max}$ 进行了限制, 例如 DS 算法的 $Eject$ 算法限制了 $K_{max} = 5$ 个顾客, 会存在弹射 5 个顾客不能使得恢复到合法状态的情况。如果从违反时间窗惩罚的路径中弹射 $K_{max}$ 顾客之后, 路径不能恢复到合法的状态, 则重新使用贪心选择的方式, 选择超过 $K_{max}$ 个顾客来构造 $EJ_r$ (第 6-7 行)。针对仅含有容量惩罚的路径, 采用贪心选择的方式获得需要弹射的顾客集合 $EJ_r$ (第 9-10 行)。最后将 $EJ_r$ 中的所有顾客都放入 $EJ$ 集合中(第 12-14 行)。分支限界的弹射算法与 DS 算法中 $Eject$ 算法不同点有:

(1) 弹射顾客的时机不同, DS 算法中当一个解通过贪心策略不能插入, 并且经过 $squeeze$ 过程也不能成功插入的时候, 会将解退回到初始的合法状态。分支限界的弹射算法的弹射时机是在修复算法的局部最优解中进行弹射顾客, 而此时解 $S$ 中可能有多条路径同时存在时间窗惩罚和容量惩罚。

(2) 将违反时间窗约束的路径和仅违反容量约束的路径进行分类。

(3) 分支限制的弹射算法不去试探所有的插入位置, 只针对一个有惩罚的路径构造一个 $EJ_r$ , 这样的改变去掉了 DS 算法中最耗时的试探之后再行弹射的操作。

(4) 分支限界的弹射算法改变了 $EJ$ 集合目标函数的计算方式, 控制算法每次弹射顾客的数量不会太多。

## 3.6 带有禁忌表的扰动算法

扰动算法( $perturb$ )是在分支限界的弹射算法执行结束之后, 对解 $S$ 进行增加多样性的操作, 是在不改变解 $S$ 的合法性基础上进行的。DS 算法中是随机选择不会为解 $S$ 增加惩罚的动作进行的, 执行一定的次数之后退出。这里可能存在一个问题就是,

一个不增加惩罚动作的反动作也是不会增加惩罚值的。如果做了全部动作的反动作，解 $S$ 会回到最初的状态，这样扰动就失效了，即使没有恢复到最初的状态，也会减弱扰动的效果。基于这个现象，算法为扰动算法设计了一个禁忌表 $TL$ ，将做过的动作进行禁忌，这样就可以避免以上问题。

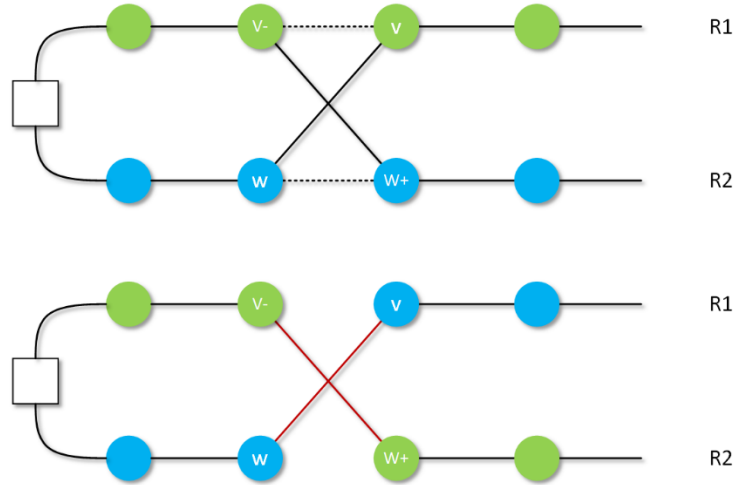


图 3.3 2-opt\*禁忌表更新示意图

禁忌表是一个二维的方阵，里面的 $|V| * |V|$ 个元素用来记录任意两个顾客，以及顾客和仓库之间的有向边的连接信息。用 $iter$ 记录扰动算法执行的步数，每次在邻域内随机挑选一个邻域动作就给 $iter$ 自增 1，并且把该邻域动作需要断开的有向边更新到禁忌表中。以 2-opt\*动作更新禁忌表举例，如图 3.3 所示，2-opt\*动作需要断开有向边 $(v^-, v)$ 和 $(w, w^+)$ ，这时将两条边更新到禁忌表中： $TL[v^-][v] = iter + len + rand(10)$ ,  $TL[w][w^+] = iter + len + rand(10)$ 。其中 $rand(10)$ 是一个 $[0,10)$ 区间的随机数， $len$ 是禁忌步长。即在 $len$ 步之内，不希望断开的边重新连接。在挑选邻域动作的时候，需要看这个动作是需要连接哪些边，如果这些连边最近被断开，就禁止扰动算法再次将这些边进行连接。还是以 2-opt\*动作举例，在图 3.3 中，2-opt\*动作想要连接有向边 $(v^-, w^+)$ 和有向边 $(w, v)$ ，需要在 $TL$ 中查看这两条连边是否被禁忌。因为不同的动作对于有向边的改变的数量是不一样的，所以单纯地将连接边的禁忌值相加是不合理的，算法根据连接边的数量求了平均值。图 3.3 中该动作的年龄值就是 $Tabu_{step} = (TL[v^-][w^+] + TL[w][v])/2$ 。如果 $iter < Tabu_{step}$ ，那么当前动作就是

处于禁忌状态，否则该动作没有被禁忌，可以作为执行动作。其他的邻域动作都用相同的方式更新 $TL$ ，以及判断动作的是否处于禁忌状态。

### 3.7 探索路径数的最大下界

探索 VRPTW 路径数的最大下界，可以为路径数最小化算法设置预期的目标函数值。当算法已经搜索到路径数下界之后，及时地停止搜索，否则继续进行无效的搜索，会导致计算资源的浪费。解决 NP-hard 问题下界的一般方法是松弛约束，忽视或减弱已知问题的约束，并求解其他问题的下界。再结合当前已知的最优解，如果当前已知的最优解已经达到求得的下界，那么就可以证明当前问题达到了全局最优。例如，最小生成树问题可以看成是 TSP 问题的下界，通过 TSP 问题中一个节点只能经过一次的约束，从而去求解最小生成树问题，将最小生成树问题的最优解作为 TSP 问题的一个下界，虽然这个下界可能不是最大的下界，但是可以作为一个基准去衡量非精确算法的求解优度。

同理，在 VRPTW 中，也可以采用松弛约束的思路求解路径数的下界，再结合 SINTEF 平台上的当前已知的最优解<sup>1</sup>，去确定某个算例是否已经达到了理论的路径数的下界。本文中使用了两种方法用于求解路径数下界。

(1) 仅考虑容量约束的路径数下界，仅仅满足车辆的容量约束，可以得到一个最小的路径数量，即用所有顾客的需求总和除以车辆的容量上限求得。(2) 对于 VRPTW 问题，通过松弛路径数量的约束，将问题转化为最大团问题，再使用求解最大团问题的启发式算法，可以获得一个路径数的下界。将使用这两种方法得到的路径数下界结合 SINTEF 平台的当前最小的路径数，可以确定绝大部分的算例的路径数的最大下界。

使用最大团算法求解路径数量的下界。定义顾客结点对集合  $E_{mcp}$ ，对于任意一对顾客  $(vi, vj) \in E_{mcp}$ ，都有路径  $\{v_0, v_i, v_j, v_{n+1}\}$  和路径  $\{v_0, v_j, v_i, v_{n+1}\}$  都存在时间窗惩罚或者容量惩罚。即这两个顾客互不相容，仅仅将这两个顾客放入一条路径都不能

---

<sup>1</sup> <https://www.sintef.no/projectweb/top/vrptw/>



让路径合法。那么定义图 $G'(V', E_{mcp})$ ，在图 $G'$ 上求解最大团问题，在得到的最优解中，最大团中的顾客数量 $N_{mcp}$ 就是 VRPTW 的路径数量的下界。因为在这个团中，任意两个顾客都不可以放入同一条路径，因此就至少需要 $N_{mcp}$ 条路径来分别放置这 $N_{mcp}$ 个顾客。本文中，直接使用了 Chu M. Li 等人的最大团算法求解<sup>[40]</sup>。

有趣的是，借助最大团算法求得的团中的结点，也可以用顾客加权的路径移除算法找到。在顾客加权的路径移除算法求解到 $N_{mcp}$ 条路径的时，当算法运行结束后，将得到每一个顾客的权重值 $W_{cus}$ ，按照 $W_{cus}$ 值降序顺序选取 $N_{mcp}$ 个结点刚好就是最大团求解出的最大团顾客集合。这种方法发现了当前已知的最优解中，已经有 10 个算例达到了路径数的下界。在结果分析中的 3.8.4 节，详细列出了求解的结果。

### 3.8 结果分析

#### 3.8.1 基准算例集

AWLS 算法在 300 个 Gerhring and Homberger 的基准算例上<sup>[41]</sup>进行了测试。算例按照顾客的分布分为三种，R 类型中顾客的分布是随机的，C 类型算例中顾客的分布是有聚类（cluster）特征的，RC 类型的算例中的顾客分布，部分顾客呈现聚类特征，其他顾客呈现随机分布。按照车辆的容量和时间窗分为两种，1 类型的是车辆容量小的，时间窗也较窄，2 类型的车辆容量较大和时间窗较宽松。组合之后一共有 C1、C2、R1、R2、RC1 和 RC2 一共 6 种不同类型的算例，顾客规模为 200、400、600、800、1000，每种规模下有 10 个使用不同随机数生成的实例。一共有 300 个算例。

#### 3.8.2 运行环境以及算法参数配置

在表 3.1 中列出了算法运行的软硬件配置，算法运行在一台负载不高的计算机上，采用单线程方式进行测试。算法参数设置如下：在 3.3 节中 $W_{cus}$ 的初值设置为 1， $W_{up}^1$ 设置为 1， $W_{up}^2$ 设置为 2。在 3.4 节路径加权的修复算法中容量惩罚的乘积系数 $\alpha$ 设置为 1.0， $W_r$ 的初值设置为 1，每次调整增加一个单位， $N_{step}$ 设置为 100。在 3.6 节中的禁忌表的参数 $len$ 设置为 10。

表 3.1 AWLS 算法运行的环境配置



软硬件项目	版本和参数
CPU	Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.30GHz
计算机内存	16 GB
硬盘	40 GB
操作系统	Ubuntu 21.04
编程语言	C++ 11
编译器	G++ 10.3.0
构建工具	CMake
编译优化等级	-O3

### 3.8.3 与 DS 算法的对比

在 SINTEF 平台上维护了 VRPTW 问题中 Gehring and Homberger 算例集的已知最优解<sup>1</sup>，该榜单采用分层的目标函数，即优先优化以路径数量，在最小化路径数量的基础上优化路径长度。在 Combopt 平台上<sup>12</sup>记录了 SINTEF 榜单的更新历史<sup>2</sup>。可以看到 VRPTW 问题在路径数量最小化的目标上优化难度较大，更新频率较小。最近的一次更新是 2021 年 12 月 20 日，距离上一次更新的时间的 2016 年，已经过去了 5 年。在 2021 年那优化公司 SCR 更新了两个算例的最优路径数：C2\_8\_7，以及 C2\_10\_9。使用本文的算法也可以持平这两个算例的结果。

在表 3.2 中列出了 DS 算法当时改进的 18 个最优解，以及 SCR 更新的两个最优解的运行结果。表中的第一列表示算例，第二列表示算例中的顾客数量。考虑到 CPU 单核的运算能力的差异，运行时间应该根据 cpu 的单线程分数进行折算。算法 DS 运行的硬件环境为：AMD Opteron 2.4 GHz (4 GB memory)，分数为 936，本文算法运行的 CPU 分数为 1918。表中的运行时间在 Time 列给出，单位分钟是折算之前的时间，折算之后的时间在 Time-C 列给出。应该是  $1918/936=2.04$  倍。CPU 分数信息来自 CPU-

<sup>1</sup> <https://www.sintef.no/projectweb/top/vrptw/>

<sup>2</sup> <http://combopt.org/>

Benchmarks 平台<sup>1</sup>。AWLS 算法与 DS 算法中改进的算例在运行时间上进行了比较。表中的结果是算法重复运行了五次可以命中最优解的次数。可以看到算法在运行时间上优势明显。

表 3.2 路径数最小化与 DS 算法对比

算例	数量	DS			AWLS hit#5				
		BKS	Time(m)	Time-C	BKS	1m	5m	1h	5h
C2_4_8	400	11	60	29	11	0	5	5	5
RC2_4_5	400	8	10	5	8	2	5	5	5
C1_6_6	600	59	60	29	59	0	1	5	5
C1_6_7	600	57	60	29	57	1	5	5	5
RC2_6_5	600	11	10	5	11	5	5	5	5
C1_8_2	800	72	1	0	72	5	5	5	5
C1_8_6	800	79	10	5	79	1	5	5	5
C1_8_8	800	73	240	117	73	2	5	5	5
C2_8_6	800	23	60	29	23	0	1	5	5
C2_8_7	800	24	n/a	n/a	<b>23</b>	0	0	0	<b>5</b>
RC2_8_1	800	18	1	0	18	5	5	5	5
C1_10_6	1000	99	10	5	99	0	3	5	5
C1_10_7	1000	97	60	29	97	0	5	5	5
C1_10_8	1000	92	300	146	92	0	4	5	5
C2_10_3	1000	28	10	5	28	4	5	5	5
C2_10_6	1000	29	10	5	29	5	5	5	5
C2_10_7	1000	29	10	5	29	4	5	5	5
C2_10_8	1000	28	300	146	28	0	5	5	5
C2_10_9	1000	29	n/a	n/a	<b>28</b>	0	0	0	<b>1</b>
C2_10_10	1000	28	10	5	28	5	5	5	5
Total						39	79	90	96

### 3.8.4 路径数下界结果分析

在表 3.3 中列出了本文使用最大团算法求解路径数下界的详细结果。Time 列中，每个算例的求解时间均小于 1 秒钟。表中 SINTEF 列表示的是当前 SINTEF 平台上的最小路径数，MCP 列表示的是使用最大团求解出合法解路径数的下界。RN-C 列是不考虑顾客的时间窗约束，将问题视为 CVRP 时候的路径数下界，这样可以确定绝

<sup>1</sup> <https://www.cpubenchmark.net/singleThread.html>

大部分算例的下界（表中没有列出）。但是有少部分算例的路径数最大下界依然无法确定，将最大团应用于这个求解路径数下界，可以发现更多算例已经达到了路径数的下界，使用最大团方法可以确定的路径数最大下界的在表中列出，其中加粗显示的是只使用容量约束无法确定的算例，要借助最大团方法才可以确定最大下界的算例。

表 3.3 最大团算法求解路径数下界结果

算例	SINTEF	MCP	NR-C	Time(m)
C1_2_1	20	<b>20</b>	18	<1
C1_4_1	40	<b>40</b>	36	<1
C1_6_1	60	<b>60</b>	56	<1
C1_8_1	80	<b>80</b>	72	<1
C1_8_2	72	72	72	<1
C1_10_1	100	<b>100</b>	90	<1
R1_2_1	20	<b>20</b>	18	<1
R1_4_1	40	<b>40</b>	36	<1
R1_6_1	59	<b>59</b>	54	<1
R1_8_1	80	<b>80</b>	72	<1
R1_10_1	100	<b>100</b>	91	<1
RC2_2_5	4	4	4	<1

### 3.8.5 AWLS 中加权策略对比

为了验证文章中策略的有效性，设计三组对照实验进行分析。结果在表 3.4 中。使用的策略分别是：（1）策略 1 设置 $W_{up}^1$ 和 $W_{up}^2$ 都是 1，其他的条件和 3.8.2 节中的配置保持一致。（2）策略 2 设置 $W_{up}^1$ 为 2， $W_{up}^2$ 为 1，其他的条件和 3.8.2 节中的配置保持一致。（3）策略 3 的设置为：将 3.5 节分支限界的弹射算法的 $EJ$ 的度量函数 $f(EJ)$ 设置为 $f(EJ) = \sum_{c \in EJ} W_{cus}(c)$ ，其他条件和 3.8.2 节中的配置保持一致。

表 3.4 表中列出了这三种策略下，AWLS 算法在不同时间上限中，使用不同的随机种子运行五次命中最优解的次数。表中的 NR 列表示算例的最优路径数。通过对比分析可以得出以下结论：（1）从策略 1 和策略 2 的对比可以看出，最好的 $W_{up}^1:W_{up}^2$ 的权重比是 1:2，其次是 2:1，最差的权重比是 1:1。（2）从策略 3 与最佳的配置比例可以看出，将 Eject 算法中 $EJ$ 的目标函数换为 $f(EJ) = |EJ| * \sum_{c \in EJ} W_{cus}(c)$ 求解的效果更好。

表 3.4 AWLS 算法策略对比

算例	数量	NR	Strategy 1 hit#5				Strategy 2 hit#5				Strategy 3 hit#5			
			1m	5m	1h	5h	1m	5m	1h	5h	1m	5m	1h	5h
C2_4_8	400	11	0	4	5	5	0	4	5	5	1	4	5	5
RC2_4_5	400	8	1	5	5	5	0	5	5	5	1	5	5	5
C1_6_6	600	59	0	1	1	1	0	0	1	1	0	0	1	2
C1_6_7	600	57	0	4	5	5	0	2	5	5	0	4	5	5
RC2_6_5	600	11	5	5	5	5	5	5	5	5	5	5	5	5
C1_8_2	800	72	5	5	5	5	5	5	5	5	5	5	5	5
C1_8_6	800	79	0	0	5	5	0	0	5	5	0	2	5	5
C1_8_8	800	73	2	5	5	5	0	5	5	5	0	3	5	5
C2_8_6	800	23	1	4	5	5	0	3	5	5	1	5	5	5
C2_8_7	800	23	0	0	<b>1</b>	<b>3</b>	0	0	<b>0</b>	<b>3</b>	0	0	<b>1</b>	<b>5</b>
RC2_8_1	800	18	5	5	5	5	5	5	5	5	5	5	5	5
C1_10_6	1000	99	0	1	5	5	0	0	5	5	0	0	5	5
C1_10_7	1000	97	0	4	5	5	0	3	5	5	0	5	5	5
C1_10_8	1000	92	0	2	5	5	0	0	5	5	0	1	5	5
C2_10_3	1000	28	5	5	5	5	2	5	5	5	3	5	5	5
C2_10_6	1000	29	5	5	5	5	5	5	5	5	5	5	5	5
C2_10_7	1000	29	5	5	5	5	3	5	5	5	5	5	5	5
C2_10_8	1000	28	0	5	5	5	0	1	5	5	1	5	5	5
C2_10_9	1000	28	0	0	0	0	0	0	<b>0</b>	<b>1</b>	0	0	<b>0</b>	<b>1</b>
C2_10_10	1000	28	5	5	5	5	5	5	5	5	5	5	5	5
Total			39	70	87	89	30	58	86	90	37	69	87	93

### 3.8.6 所有算例集上的比较

表 3.5 和表 3.6 是算法在所有的测试集上测试的结果，为了与先前研究中路径数比较指标保持一致，本文算法也采用累积路径数（Cumulative Number of Vehicles, CNV）进行评估。表中一起比较的算法有车辆路径问题的通用启发式（RP）算法<sup>[42]</sup>，求解车辆路径问题的具有凸的时间惩罚函数的迭代局部搜索算法（An Iterated Local Search Algorithm for the Vehicle Routing Problem with Convex Time Penalty Functions, IIN）算法<sup>[43]</sup>，两阶段带有弹射池的启发式（LZ）算法<sup>[44]</sup>，和 DS 算法。表中的 Computer 行中的 P 代表奔腾处理器，O 代表皓龙处理器，X 代表至强处理器。CPU 主率也在表中给出，单位 GHz。t（min）行是算法的运行时间，单位分钟。Runs 行是算法运行的次数，如果运行多次，表中给出的是多次运行的最好结果。

# 华中科技大学硕士学位论文

表 3.5 AWLS 算法与同类型算法在小规模算例上的比较

200	PR	IIN	LZ	DS				AWLS			BKS
				1m	10m	60m	1h	60m	120m	300m	
R1	18.2	18.2	18.2	18.2	18.2	18.2	18.2	18.2	18.2	18.2	18.2
R2	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0
RC1	18.0	18.0	18.0	18.0	18.0	18.0	18.0	18.0	18.0	18.0	18.0
RC2	4.3	4.3	4.3	4.3	4.3	4.3	4.3	4.3	4.3	4.3	4.3
C1	18.9	18.9	18.9	18.9	18.9	18.9	18.9	18.9	18.9	18.9	18.9
C2	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0	6.0
CVN	694	694	694	694	694	694	694	694	694	694	694
Computer	P3.0G	P2.8G	P2.8G	O2.4G	O2.4G	O2.4G	O2.4G	X2.3G	X2.3G	X2.3G	n/a
t (min)	7.7	n/a	10	1	10	60	60	60	120	300	n/a
Runs	10	1	2	1	1	1	1	1	1	1	n/a
400	PR	IIN	LZ	DS				AWLS			BKS
				1m	10m	60m	1h	60m	120m	300m	
R1	36.4	36.4	36.4	36.4	36.4	36.4	36.4	36.4	36.4	36.4	36.4
R2	8.0	8.0	8.0	8.0	8.0	8.0	8.0	8.0	8.0	8.0	8.0
RC1	36.0	36.0	36.0	36.0	36.0	36.0	36.0	36.0	36.0	36.0	36.0
RC2	8.5	8.6	8.5	8.5	8.4	8.4	8.4	8.4	8.4	8.4	8.4
C1	37.6	37.7	37.6	37.6	37.6	37.6	37.6	37.6	37.6	37.6	37.6
C2	12.0	12.0	11.7	12.0	11.8	11.6	11.6	11.6	11.6	11.6	11.6
CVN	1385	1387	1382	1385	1382	1380	1380	1380	1380	1380	1380
Computer	P3.0G	P2.8G	P2.8G	O2.4G	O2.4G	O2.4G	O2.4G	X2.3G	X2.3G	X2.3G	n/a
t (min)	15.8	n/a	20	1	10	60	120	60	120	300	n/a
Runs	5	1	4	1	1	1	1	1	1	1	n/a
600	PR	IIN	LZ	DS				AWLS			BKS
				1min	10m	60m	3h	60m	120m	300m	
R1	54.5	54.5	54.5	54.5	54.5	54.5	54.5	54.5	54.5	54.5	54.5
R2	11.0	11.0	11.0	11.0	11.0	11.0	11.0	11.0	11.0	11.0	11.0
RC1	55.0	55.0	55.0	55.0	55.0	55.0	55.0	55.0	55.0	55.0	55.0
RC2	11.6	11.6	11.5	11.6	11.4	11.4	11.4	11.4	11.4	11.4	11.4
C1	57.5	57.5	57.4	57.4	57.4	57.2	57.2	57.2	57.2	57.2	57.2
C2	17.5	17.4	17.4	17.4	17.4	17.4	17.4	17.4	17.4	17.4	17.4
CVN	2071	2070	2068	2069	2067	2065	2065	2065	2065	2065	2065
Computer	P3.0G	P2.8G	P2.8G	O2.4G	O2.4G	O2.4G	O2.4G	X2.3G	X2.3G	X2.3G	n/a
t (min)	18.3	n/a	30	1	10	60	180	60	120	300	n/a
Runs	5	1	6	1	1	1	1	1	1	1	n/a

DS 算法和 AWLS 运行时间单位 h 表示小时，m 表示分钟。算法的运行时间限制分别为 1 小时，2 小时，5 小时，在表中可以看出 AWLS 算法在 1 小时之内持平了 DS 算法所有的最佳结果。在表 3.6 中 800 个顾客的实例在运行 2 小时突破了 DS 的记录，将算法上限设置成 5 小时，可以持平 SINTEF 平台上的所有结果。表中的 BKS

列是 SINTEF 平台上的当前最优解的路径数。对于 1000 个顾客规模的算例，设置每个算例运行上限 1 小时，可以持平 DS 的所有记录。设置每个算例运行时间上限为 5 小时，可以持平 SINTEF 平台上的所有记录。

表 3.6 AWLS 算法与同类型算法在大规模算例上的比较

800	PR	IIN	LZ	DS				AWLS			BKS
				1m	10m	60m	4h	60m	120m	300m	
R1	72.8	72.8	72.8	72.8	72.8	72.8	72.8	72.8	72.8	72.8	72.8
R2	15.0	15.0	15.0	15.0	15.0	15.0	15.0	15.0	15.0	15.0	15.0
RC1	73.0	72.4	72.0	72.1	72.0	72.0	72.0	72.0	72.0	72.0	72.0
RC2	15.7	15.7	15.6	15.6	15.4	15.4	15.4	15.4	15.4	15.4	15.4
C1	75.6	75.7	75.4	752.0	75.1	75.0	74.9	74.9	74.9	74.9	74.9
C2	23.7	23.4	23.4	23.4	23.4	23.3	23.3	23.3	23.2	23.1	23.1
CVN	2758	2750	2742	2741	2737	2735	2734	2734	<b>2733</b>	<b>2732</b>	2732
Computer	P3.0G	P2.8G	P2.8G	O2.4G	O2.4G	O2.4G	O2.4G	X2.3G	X2.3G	X2.3G	n/a
t (min)	22.7	n/a	40	1	10	60	240	60	120	300	n/a
Runs	5	1	8	1	1	1	1	1	1	1	n/a
1000	PR	IIN	LZ	DS				AWLS			BKS
				1m	10m	60m	5h	60m	120m	300m	
R1	92.2	91.9	91.9	91.9	91.9	91.9	91.9	91.9	91.9	91.9	91.9
R2	19.0	19.0	19.0	19.0	19.0	19.0	19.0	19.0	19.0	19.0	19.0
RC1	90.0	90.0	90.0	90.0	90.0	90.0	90.0	90.0	90.0	90.0	90.0
RC2	18.3	18.3	18.3	18.4	18.2	18.2	18.2	18.2	18.2	18.2	18.2
C1	94.6	94.5	94.4	94.1	94.0	93.9	93.8	93.8	93.8	93.8	93.8
C2	29.7	29.4	29.3	29.3	28.9	28.9	28.8	28.8	28.8	28.7	28.7
CVN	3438	3431	3429	3427	3420	3419	3417	3417	3417	<b>3416</b>	3416
Computer	P3.0G	P2.8G	P2.8G	O2.4G	O2.4G	O2.4G	O2.4G	X2.3G	X2.3G	X2.3G	n/a
t (min)	26.2	n/a	50	1	10	60	300	60	120	300	n/a
Runs	5	1	10	1	1	1	1	1	1	1	n/a

### 3.9 本章小结

在这一章介绍了一种自适应的基于加权的局部搜索算法（AWLS），用于解决 VRPTW 中路径数量最小化问题。该算法从一个启发式的初始解构造算法开始，不断删除路径，直到达到时间限制，从而返回可优化到的最小路径数的解。AWLS 采用权重来引导局部搜索，这个加权内容包括两部分的内容，首先是顾客加权，在外层的移除算法中使用，其次是路径加权算法，在修复算法中使用，两种加权算法形成了一种两层的加权格局，从而使得算法获得了较好的适应性，搜索过程的疏散性显著提高。AWLS 在标准测试集上表现出了良好的求解效果。

## 4 求解路径总长度最小化问题的混合局部搜索算法

通过论文调研,在 VRPTW 论文中,通常来说优化目标是分级的,算法首先需要最小化所有路径的数量,在此基础上优化路径的总长度,使用分级目标记录的已知最优解被挪威科技工业研究所(SINTEF)管理和维护<sup>1</sup>,在 SINTEF 的 VRPTW 榜单的平台上可以看到 VRPTW 问题最新的最好结果。该机构会对提交的最新的求解结果进行验证,并且更新最好的已知最优解。在 12-th DIMACS-VRP 实现挑战赛中<sup>2</sup>,VRPTW 赛道的优化目标被定义为了单目标优化,即最小化所有路径的总长度,不把路径的数量作为优化目标,路径数的上限只要满足最大的车辆数即可。本章针对挑战赛的目标设计了混合局部搜索算法。

### 4.1 混合局部搜索算法总体框架

混合局部搜索算法在两个已经发表的 SISR 算法,以及 EAMA 算法的基础上进行设计。路径长度最小化将两个算法进行了有机地结合,构建了混合局部搜索(Hybrid Local Search, HLS)用于求解路径总长度最小化的问题。

针对这样的目标函数,EAMA 需要做出相应的改变,因为 EAMA 算法根据分层的目标函数设计的。EAMA 也是一种标准的模因算法,算法内部维护了一个种群。并且 EAX 交叉算子要求交叉的两个解中的路径数是相等的。在 12th-DIMACS 挑战赛中没有将车辆数作为第一优先级,所以在种群中,可以同时存在路径数不同的解,但是这样与 EAX 的要求冲突。为此算法中设计了多个种群,每个种群中维护路径数相等的解。这样的做法一方面可以适应 EAX 的需求,另一方面,不同路径的数的解空间是互不重叠的,使用路径对解空间进化了划分,算法在搜索的时候就可以更有针对性进行搜索。

路径总长度最小化算法的主流程如下:首先使用 4.2 节中的种群集合初始化算法

---

<sup>1</sup> <https://www.sintef.no/projectweb/top/vrptw/>

<sup>2</sup> <http://dimacs.rutgers.edu/programs/challenge/vrp/>



生成多个种群，每个种群中的所有解都具有相同数量的路径数。这样的分组可以保证 EAMA 操作的正确性。接着，对种群集中的每个个体都使用 4.3 节中的简单局部搜索进行局部搜索，以提高解的质量。

然后，使用 4.4 节中的寻找最佳种群算法找到一个合适的种群，并调用 4.7 节中的混合局部搜索（Hybrid Local Search, HLS）对这个种群其进行优化。HLS 是一个针对单个种群进行的优化算法，这个算法中包括 4.5 节中介绍的改进的 SISR 算法以及 4.6 节中改进的 EAMA 算法两个重要的组成部分。HLS 中通过局部搜索、基于种群进化等流程对种群中的解进行改进，直到种群达到收敛状态。

当一个种群收敛之后，需要切换到其他种群进行搜索。此时，再次使用 4.4 节中的寻找最佳种群算法选择下一个需要搜索的种群，并使用混合局部搜索算法进行改进。该算法能够有效地求解路径总长度最小化问题，获得了较好的求解效果。

## 4.2 种群集合初始化

HLS 的种群不同于经典模因算法的种群，经典的模因算法一般只维护一个种群，HLS 算法中维护了若干个种群，在合适的路径数的解空间里分别维护了一个种群。因此种群集中有多个种群。种群集合的初始化要依赖于路径数量最小化的子算法，在 3 章中已经进行了详细的介绍。以下具体介绍如何生成种群集合。

为了简化算法，在每个不同的路径数下的种群大小都设置为相等的值，种群的大小使用  $popSize$  表示。首先算法重复调用  $popSize$  次初始解生成算法，生成  $popSize$  个初始解  $\sigma = \{S_1, S_2, S_3, \dots, S_{popSize}\}$ 。之后对  $\sigma$  中的每个解都调用顾客加权的路径移除算法，进行路径最小化过程，每次如果顾客加权的路径移除算法成功移除一条路径就将这个解保存下来。以任意一个解  $S_i \in \sigma$  为例。如果  $S_i$  中有  $k$  条路径将  $S_i$  重新表示为  $S_i^k$ ，那么减少一条路径就是  $S_i^{k-1}$ ，一直到路径数不能减少为止。此时使用顾客加权的路径移除算法就获得了一系列的解  $S_i^k, S_i^{k-1}, S_i^{k-2}, \dots, S_i^m$ 。同理，在  $\sigma$  中的任意一个解都可以生成一系列的解。将这些解按照路径数进行分类，因为  $\sigma$  中的每个解初始化都采用不同的方式，所以使用  $\sigma$  中不同的初始解生成的解的个数也不同。将这些解按照路径数分类之后，每个路径数下有的解个数可以达到  $popSize$  个，但是有的达不到，这时

候, 算法会将那些不足 $popSize$ 个解的种群删掉, 只保留个体数为 $popSize$ 的种群。这样算法的种群集合生成完毕, 这个种群集合中包含了若干个种群, 每个种群中的个体解的路径数都是相等的。

种群生成结束之后, 算法使用一个简单局部搜索对解进行局部搜索, 用于提高解的质量。简单局部搜索在 4.3 节中进行详细介绍。

## 4.3 简单局部搜索

简单的局部搜索是一个经典的局部搜索的过程, 主要包括邻域评估以及做邻域动作两个步骤。简单局部搜索被设计成了一个单纯的下降算法, 在算法搜索过程中, 不增加扰动和禁忌等其他策略。

简单局部搜索中的经典邻域动作包括: (1) 交换两条路径的后半部分 (2-opt\*)。 (2) 两个顾客的交换 (swap 或 exchange)。 (3) 单个顾客的移动 (relocate 动作), 包括顾客的移入和移出。

除此之外, 简单局部搜索中还加入了新的邻域动作: (1) 从路径中移出两个连续的顾客 $out(2)$ 。(2) 在路径中 2 个邻接顾客的路径和 3 个邻接的顾客交换 $swap(2,3)$ 。在路径中 2 个邻接顾客的路径和 2 个邻接的顾客交换 $swap(2,2)$ 。在路径中 3 个连续的路径和 1 个顾客交换 $swap(3,1)$ 。交换动作不仅可以发生在不同的路径之间, 还可以发生在同一条路径内部。

简单局部搜索在邻域评估部分与 3.4 节的路径加权的修复算法一样, 在邻域动作的选择时也要先选择顾客 $v$ , 之后选择顾客 $w$ , 顾客 $v$ 是输入算例中所有的顾客, 顾客 $w$ 选择与 $v$ 在地理位置前 10 近的顾客。当在邻域内找不到可以改进当前解的动作, 简单局部搜索就会退出。

## 4.4 寻找最佳种群

经过种群集合初始化, 路径集合中的种群的数量一般来说比较大, 但是最优解一定是只产生在某一路径数下的, 在算法搜索的过程中, 需要对搜索路径数的范围不断缩小, 在路径集合中不断淘汰种群, 增加算法搜索的集中性。本文的算法采用的是根

据每个种群中的最优解的顺序来选择搜索的顺序。在种群初始化结果之后,使用一个数据结构存储每个种群下面的最优解的具体的路径总长度,根据这个路径总长度给种群进行排序,路径总长度短的种群优先搜索。

## 4.5 改进的 SISR 算法

破坏和重建方法 (Ruin and Recreate), 是一种将解进行破坏重组的方法。该方法将部分顾客移除出当前解, 从而松弛当前解  $S$ , 再将移除的顾客重新插入到局部解中。该方法与弹射池的相同点都是移除顾客, 不同点是破坏和重建方法使用了算例的地理位置信息, 并且单次移除顾客的数量比弹射池多, 从而使得当前解  $S$  获得更多的约束松弛。SISR 算法<sup>[22]</sup>中提出了一种新的破坏方式, 移除一条路径上的连续的顾客 (String Removal)。该算法获得在 VRP 的多个变种问题上都取得了很好的效果, 更新了若干已知最优解。

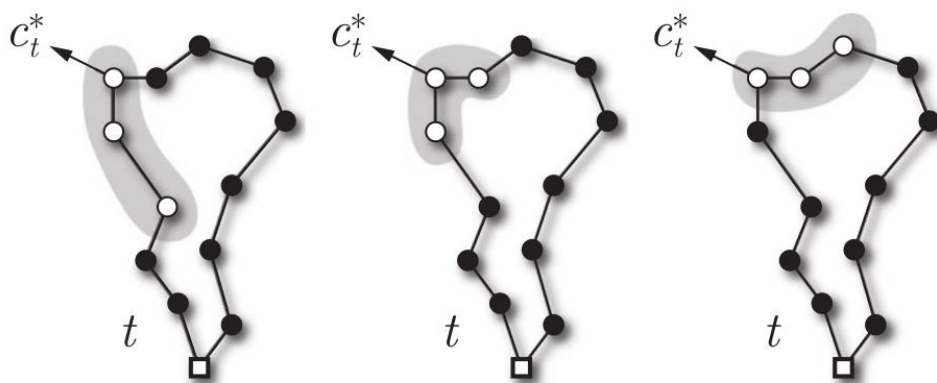


图 4.1 移除路径内连续顾客示意图<sup>[22]</sup>

如图 4.1 所示的路径连续顾客的移除过程的一个例子, 算法首先选择一个种子顾客  $C_t^*$ , 在  $C_t^*$  所在的路径中移除一段包含种子顾客的路径。然后在  $C_t^*$  的附近再次选择下一个种子顾客  $C_{t2}^*$ , 下一个种子顾客的选取要求与  $C_t^*$  不在同一条路径上。这样迭代地选择下一条路径, 直到选择的顾客满足给定的数量需求。这种移除方式的优点是移除的顾客集合在不断变化。单纯利用顾客地理位置信息移除的顾客集合可能存在重复, 搜索的多样性比路径上连续移除顾客要差。

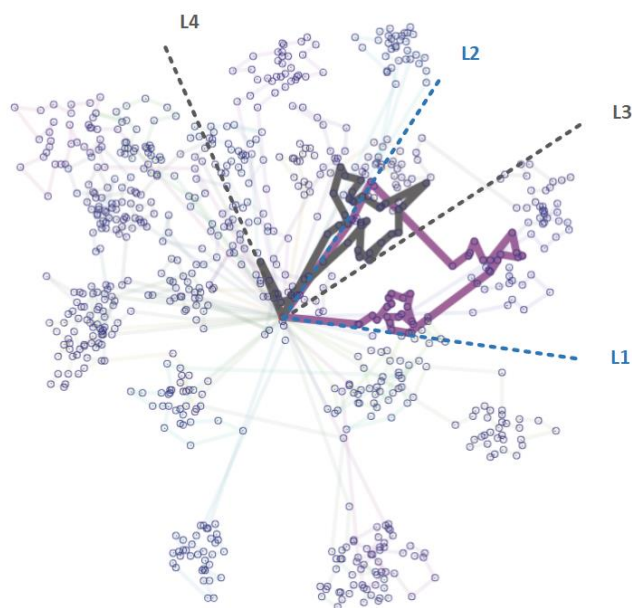


图 4.2 移除有扇形重叠区域顾客示意图（算例 C2\_8\_7）<sup>1</sup>

算法在以上的移除方式之外，增加了两种新的移除方式：

（1）第一种方法是删除整个路径，虽然移除路径中一段顾客（String Removal）也有一定的概率可以移除整条路径，但是概率较低，单独添加这样的一条移除规则有助于当前解在其他更少的路径数的解空间进行探索，增加了搜索的多样性。

（2）第二个是删除两条路径上有扇形重叠部门的顾客。如图 4.2 中，是输出算例 C2\_8\_2 的一个解的示意图，虚线 L1 和 L2 之间的路径与虚线 L3 和 L4 之间的路径存在扇形重叠区域 L2 和 L3 之间的区域。在 L2 和 L3 之前的扇形重叠区域中的顾客，是最有可能被移动到其他路径上的顾客。因此将扇形重叠区域的顾客从解中移除是一个合理的设计。算法随机选择两条扇区重叠的路径，并移除重叠区域中的顾客，重叠扇形区域的思路来自 Vidal 等人<sup>[24]</sup>。

破坏和重建方法在移除完顾客之后，需要将移除的顾客再次贪心地插入当前解。再次贪心插入的步骤大致可以分为两步，首先是对顾客按照一定顺序排序，按照不同的优先级对顾客进行重新插入：（1）优先插入距离仓库近的顾客。（2）优先插入需求量大的顾客。（3）优先插入时间窗较窄的顾客。（4）用随机的顺序插入顾客。算法每

<sup>1</sup> <http://combopt.org/>

次会随机选取一种挑选的顺序在当前解中为顾客挑选位置。在确定好要插入的顾客之后，算法在当前解中依次寻找合适的位置，对插入位置的要求首先是合法性，就是放入顾客之后，不会产生时间窗约束和容量约束的违反，其次是路径长度的增量，优先选择路径长度增加最少的位置放入。为了在挑选位置的部分加入一定随机性，算法采用了 SISR 算法中的闪烁机制<sup>[22]</sup>，使用一定的概率跳过最佳的插入位置。如果在当前解中找不到合法的插入位置，算法会重新建立一条路径专门用来存放该顾客。

## 4.6 改进的 EAMA 算法

改进的 EAMA 算法基于 Nagata 等人（2010）提出的 EAMA 算法做的提高和改进。EAMA 算法框架是一个经典的模因算法框架，算法内部维护和更新了一个大小固定的种群。EAMA 算法每次从种群中随机选择一对父代解  $P_a$  和  $P_b$ ，之后 EAX 交叉算子利用  $P_a$  和  $P_b$  生成新的后代  $P_c$ 。如果  $P_c$  的目标函数值比  $P_a$  更小，它将取代  $P_a$ 。该算法的主要贡献以及关键的部件是 EAX 交叉算子，EAX 交叉算子的主要流程包括以下三个步骤：

（1）产生边集  $G_{ab}$ ， $G_{ab}$  是用  $P_a$  和  $P_b$  两个父代解的边集的并集减去两个边集的交集得到的。因此  $G_{ab}$  是保存着来着两个父代解中的所有差异边，差异边意为只在  $P_a$  或者  $P_b$  中出现过一次的边。

（2）生成 AB 环，AB 环是通过在  $G_{ab}$  中通过遍历找到的环路。在  $G_{ab}$  中的结点的特征是度数都为偶数，度数为 2 或者 4。因为对于任意一个结点来说，它的度数取决于来自两个父代解的边的指向情况。在父代解中任意一个顾客的度都是 2，所以在两个父本解的并集中，顾客结点的度数为 4，当去掉公共边的时候，度数的减少量是偶数，因为会同时去掉两条边。所有剩下的顾客度数就都是 0、2 或者 4，然后孤立的顾客结点被删掉之后，保存在  $G_{ab}$  中的顾客结点的度就都是 2 或者 4。对于仓库节点来说，度数取决于路径数，仓库的度数是路径数的两倍。如果两个父代解中的路径数不相同，会导致仓库的出度不能将  $G_{ab}$  中所有顾客相连的情况，从而导致 AB 环的形成失败。这也是 EAX 要求两个父代解路径数相同的原因。

（3）应用 AB 环，AB 环是一个包含偶数个数边的环路，分别有相同数量的边来

自 $P_a$ 和 $P_b$ 。应用 $AB$ 环的过程是将 $AB$ 环上来自 $P_a$ 的边移除，并且加上 $AB$ 环中来自 $P_b$ 的边形成 $P_c$ 。此时 $P_c$ 可能会产生不经过仓库的子回路，需要对子回路从某一处断开然后接到已有的路径上。经过去除子回路之后得到的解 $P_c$ 如果违反了时间窗或者容量约束，要对新解 $P_c$ 进行修复。

在 EAMA 算法中， $AB$ 环的选择有两种策略：（1）选择单个 $AB$ 环。（2）选择具有共同顾客节点的单个 $AB$ 环。如果可以使用父母 $P_a$ 和 $P_b$ 生成 $n$ 个 $AB$ 环。如果 $n$ 个 $AB$ 环都应用于 $P_a$ ， $P_a$ 将变为 $P_b$ 。算法尝试将多个 $AB$ 环应用于 $P_a$ 会增加 EAX 的多样性，尽管这样做会使得 $P_a$ 更难修复。算法使用不同的概率来选择应用于 $P_a$ 的 $AB$ 环的数量。该想法灵感来自 Christiaens 等人提出的闪烁机制<sup>[22]</sup>。算法使用闪烁频率 $\gamma$ 来控制 EAX 中选择的 $AB$ 环的数量，其中接受 $k$ 个 $AB$ 环数的概率为 $\gamma^{k-1} \cdot (1 - \gamma)$ 。

## 4.7 混合局部搜索算法

---

### 算法 4.1 混合局部搜索算法

---

输入：单个种群 $population$

输出：混合局部搜索可以得到的最好解 $S^{best}$

---

```
1:  $S^{best}$ 用种群中当前的最好解进行初始化
2: while 不满足退出条件 do
3:    $EAMA(population)$ 
4:   for  $S_i \in population$  do
5:     对 $S_i$ 调用改进的 SISR 算法
6:   end for
7:   用种群中的最好解更新 $S_{best}$ 
8:   for  $S_i \in population$  do
9:      $S_i = perturb(S_i)$ 
10:  end for
11: return  $S^{best}$ 
```

---

混合局部搜索（HLS）算法将 SISR 算法<sup>[22]</sup>与 EAMA 算法<sup>[13]</sup>相结合。HLS 算法

的主要流程伪代码在算法 4.1 中。算法的输入是种群集合中的某个种群 $population$ ，算法的返回值是混合局部搜索能找到的最好解。算法的结束条件是，连续 2 次迭代如果没有改进种群中的最好解 $S^{best}$ ，就会退出混合局部搜索算法（第 2 行）。之后 EAMA 算法会首先对种群进行模因进化算法的优化（第 3 行）。等 EAMA 算法收敛退出后，算法会对每一个个体调用改进的 SISR 算法（第 4-6 行）。如果调用 SISR 算法路径数发生了改变，与当前 $population$ 个体的路径不相等，算法会将该解更新到种群集合对应路径数的种群中。经过 EAMA 和 SISR 过程之后，算法将搜索到的最好解更新到 $S^{best}$ （第 7 行）。之后算法会对种群中的每个个体进行扰动，给已经收敛的种群增加多样性（第 8-10 行）。混合局部搜索算法执行结束之后返回找到的最好解 $S^{best}$ （第 11 行）。

## 4.8 算法实现及单元测试

### 4.8.1 数据结构设计

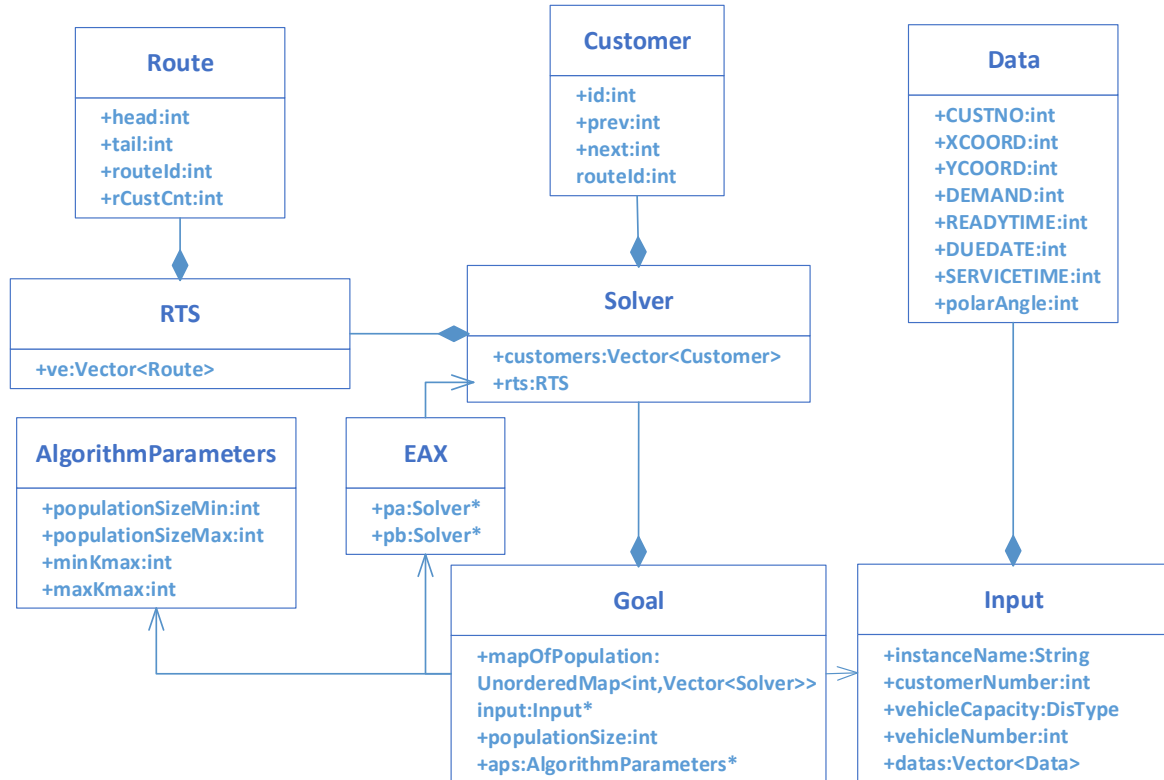


图 4.3 数据结构类图

算法使用 C++11 实现，图 4.3 是算法设计的数据结构类图。*Goal*类是算法的入口类，负责组装所有的算法组件。*Input*类在*Goal*中被引用，*Input*将读入的算例包装为算法的输入。*Goal*的成员*mapOfPopulation*保存所有的种群，对应于 4.2 节的种群集合初始化，初始化生成的所有种群保存在*mapOfPopulation*中。每个种群由*Solver*对象组成。以下具体介绍每个类的功能和具体的成员。

(1) *Data*类是一种基本数据类型，用于保存输入算例的顾客的数据，包括顾客的坐标、需求量、服务时间等。(2) *Input*类中包含有一个*Data*类型的数组，用来存储所有顾客的信息。*Input*类中还包含车辆的信息，包括车辆总数和每辆车的容量限制。

(3) *Customer*类用于记录搜索过程中顾客的状态。成员*id*是唯一标识路线的整数，*next*和*prev*成员记录同一路线上相邻位置前后的顾客*id*。(4) *Solver*类是算法的重要组成部分，它包含简单的局部搜索等重要功能。*Solver*类中的*customers*成员是*Customer*类型的数组。算法中将顾客的在*customers*数组中的索引与顾客的*id*直接对应，使用*next*和*prev*就可以将路线用双链接列表表示，每个顾客都保存了路线上前后相邻顾客的*id*。(5) *Route*类保存路线信息。成员*head*和*tail*保存路线开始和结束顾客的*id*，根据*id*可以找到路径的开头和结尾，结合双链表就可以遍历整条路径。

(6) *EAX*类是 EAX 交叉算子的实现，在种群进化目标中用于生成新的子代。(7) *AlgorithmParameter*类用于保存算法的参数。*Goal*类用于组织整个算法框架。成员*mapOfPopulation*是一个种群映射关系，它保存了算法要搜索的种群集合。

## 4.8.2 单元测试

为了增强算法的可信度，使用专门的测试项目对该算法进行了单元测试。测试程序使用了开源的测试框架*GoogleTest*提供的测试接口进行测试<sup>1</sup>。在测试程序中，编写了大量的测试样例对算法中使用到的核心组件进行了充分测试。包括 EAX 交叉算子、简单局部搜索和路径加权的修复算法中每种邻域动作、弹射池等重要组件。算法通过了所有设置的测试用例。

---

<sup>1</sup> <https://github.com/google/googletest>



## 4.9 结果分析

### 4.9.1 标准算例集

路径总长度最小化算法在两个公开的基准算例集上测试：（1）Gehring and Homberger 算例集中的 300 个算例。（2）Solomon 算例集中的 56 个算例。DIMACS 算法竞赛使用了全部的 356 个算例，对所有的求解器上进行了测试。比赛的第二阶段使用新的 300 个算例，新算例集是使用 Gehring and Homberger 算例集生成，通过仅改变每个算例的仓库的位置来获得新的算例。

### 4.9.2 参数以及运行环境

表 4.1 HLS 算法运行的环境配置

软硬件项目	版本和参数
CPU	Core (TM) i5-9300H CPU @ 2.40GHz
CPU 单线程基准分数	2367
操作系统	Ubuntu 18.04.5 LTS
编程语言	C++ 11

表 4.1 中的配置是 HLS 算法在 DIMACS 竞赛中的测试评估分为两个阶段，第一阶段在参赛队员自己的服务器环境中运行，HLS 算法第一阶段的运行环境与表 3.1 中 AWLS 的运行环境相同。每支队伍将测试结果文件提交，测试结果文件详细记录了每一个算例的详细搜索过程。第二阶段中使用的运行环境配置。求解器的第二阶段评估是在组织者统一提供的服务器进行的。

HLS 算法的参数设置如下：在 4.2 节中种群的大小  $popSize$  设置为 40，在 4.6 节中闪烁频率  $\gamma$  设置为 60。其他的参数与 EAMA 算法和 SISR 算法中的参数保持一致。

### 4.9.3 求解器分数计算规则

12-th DIMACS VRP 实现挑战赛中对于 VRP 问题的目标函数值进行了重新的定

义<sup>1</sup>，不是单纯比较算法求解输出解的目标函数值。使用收敛曲线积分值来衡量求解器的表现<sup>[45]</sup>。积分值的计算是基于求解器搜索到的最优解与给定算例的已知最优解的值之间的差异计算的。将  $BKS$  定义为给定算例的已知最优解的路径总长度的值，将  $v(0)$  定义为  $1.1 \times BKS$ 。同时，为了限制求解器的运行时间，设  $T$  为给定算例的最大运行时间，单位为秒。

当求解器依次找到比  $v(0)$  更好的  $n$  个解时，这些解随着时间的变化值在递减。对于每个解  $i = 1, \dots, n$ ，设  $v(i)$  是它的路径总长度的值，设  $t(i)$  为它被搜索到的时间单位为毫秒。特别地， $t(0)$  被定义为 0。公式 (4.1) 中定义了归一化的积分  $PI$ ：

$$PI = 100 \times \left( \frac{\sum_{i=1}^n v(i-1) \cdot (t(i) - t(i-1)) + v(n) \cdot (T - t(n))}{T \times BKS} - 1 \right) \quad (4.1)$$

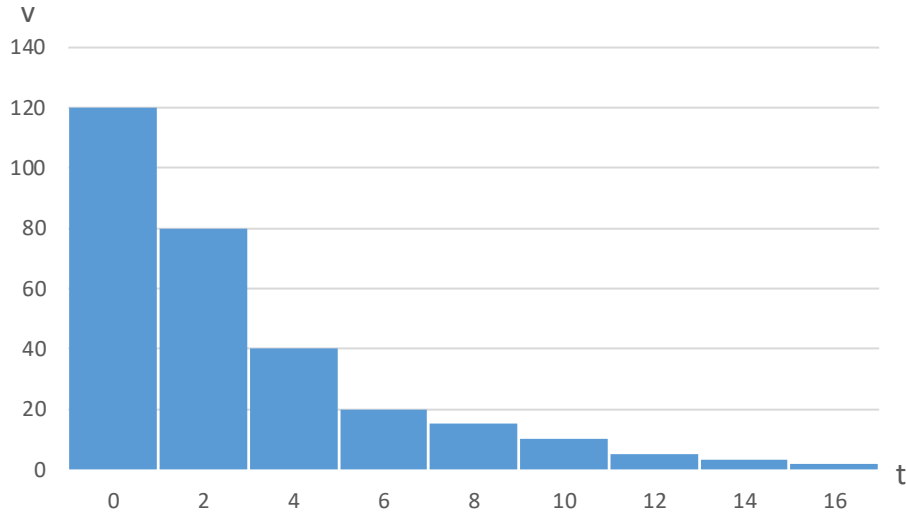


图 4.4 积分值计算示例

如图 4.4，是对于积分值计算的一个示例图。在  $t = 0$  时刻，求解器求的目标函数值是 1.1 倍的  $BKS$ ，经过两个时间单位，求解器发现了新的最优解，路径总长度的值为 80，在之后的搜索过程中依次得到的最优解的路径总长度是  $\{40, 20, 15, \dots, 2\}$ ，那么积分值就是图中带颜色的主题的面积。之后经过归一化，将这个积分的值统一处理到范围  $[1, 10]$  之间。如果一个求解器找到的最优解都在当前已知最优解的 1.1 倍以上，

<sup>1</sup> <http://dimacs.rutgers.edu/programs/challenge/vrp/vrptw/>

那么求解器对应的 $PI$ 值都是 10。如果求解器改进了当前已知的最优解，那么 $PI$ 值可能为负数。所有的参赛队伍在全部的 356 个测试实例上都用 $PI$ 值进行比较， $PI$ 值越小，排名越靠前，每一个算例都设置了不同的分数 (Score)，排名第一的求解器获得 10 分，第二的队伍拿 8 分，以此类推。求解器最终的排名根据 Score 的大小进行排序，Score 越高排名越靠前。

使用归一化的积分值去衡量求解器的求解效果，可以不止反映算法求解的最终求解优度，也可以反映算法的收敛速度，这样就做到了兼顾这两个指标，更加的科学合理。

#### 4.9.4 HLS 收敛性分析

本小节的数据来自 12-th DIMACS 实现挑战第一阶段结果，详细的全部结果可以在 DIMACS 的结果上查看<sup>1</sup>，本小节只针对 HLS 算法的收敛性进行分析，根据输入算例的特征，从顾客分布可以分为 3 类，顾客呈现聚类特征的 C 类型算例，顾客随机分布的 R 类型算例，以及聚类特征和随机特征同时存在的 RC 类型算例。按照车辆的容量约束，可以分为两类，1 类型算例是车辆的容量较小的算例，并且时间窗的限制比较严格。2 类型是车辆容量较大的算例，并且时间窗的限制比较宽松。其中 2 类型算例中，因为其车辆容量更大，因此一条路径中容纳的顾客更多，所以这个问题更加接近 TSP 问题，EAMA 算法中的 EAX 交叉算子更加适合求解这类算例。算例在顾客分布以及车辆的容量约束特征不同的影响下呈现出不同的特征。

如图 4.5 中是算例 C1\_8\_2 的收敛过程，横轴是算法的运行时间，单位秒，纵轴是路径的总长度。算例属于 C1 类别，即顾客呈现聚类分布并且车辆的容量较小的算例。在图中可以看到，在 1.88 秒之前，算法的收敛速度很快，在 1.88-7.20 秒之间，算法的收敛速度逐渐放缓，在 7.20 秒之后，算法基本收敛，在 156.11 之后有微小改进。结合 C1\_8\_2 类型算例的特点，这类型的算例因为单条路径内的顾客较少，又因为时间窗的限制比较严格，可行解的区域小很多，因此算法收敛较快。

---

<sup>1</sup> <http://dimacs.rutgers.edu/programs/challenge/vrp/vrptw/vrptw-competition/>

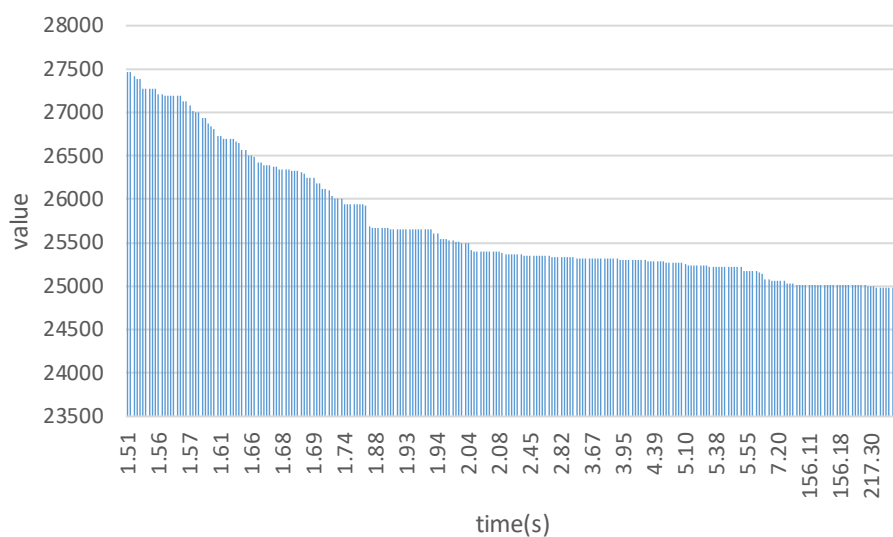


图 4.5 算例 C1\_8\_2 计算收敛过程

图 4.6 中是 C2\_8\_2 算例的计算收敛过程, 可以看到 C2 类型的算例在收敛性上, 与 C1 类型算例有明显的不同。算法在 5.28 秒之前属于下降速度较快的时间区间, 在 5.28-94.43 秒虽然下降的速度有所放缓, 但是和 C1 类型的算例相比, 目标函数的下降的速度依然很快。在 102.36 秒之后, 算法的搜索基本收敛。C2 类型的算例的车辆容量约束比 C1 更大, 并且时间窗的宽度更宽, 因此可行解的区域比 C1 更大, 可以改进的位置也更多。因此收敛的速度比 C1 类型更慢, 与实验结果相吻合。

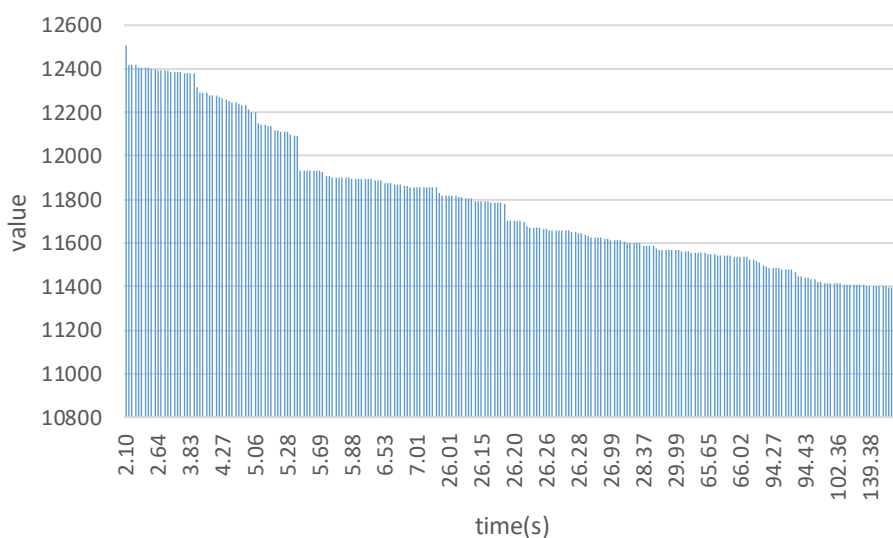


图 4.6 算例 C2\_8\_2 计算收敛过程

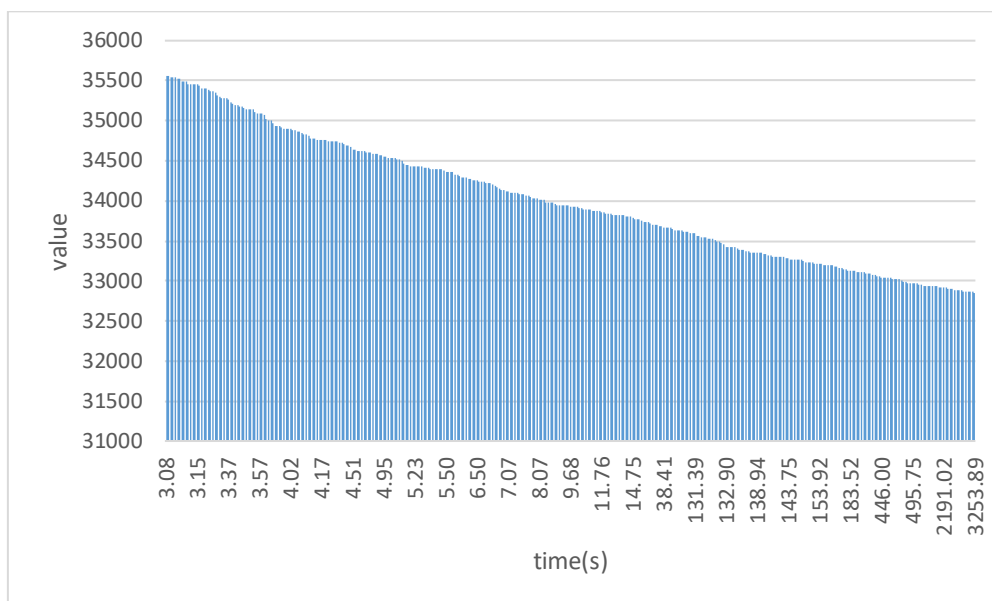


图 4.7 算例 R1\_8\_2 计算收敛过程

如图 4.7 中是算例 R1\_8\_2 的收敛过程，R1 类型算例，与 C1 类型的区别只是顾客分布规律不同，顾客的时间窗的宽度较窄。与 C1\_8\_2 算例的收敛过程类似，算法在 14.75 秒之后，收敛速度放缓。与 C1\_8\_2 类型算例不同的是，算法在搜索后期没有出现明显的收敛特征，但是发现新的最优解的时间间隔越来越长。图 4.8 是 RC1\_8\_2 算例的收敛曲线。从收敛曲线来看，收敛过程与 R1 算例比较相似。

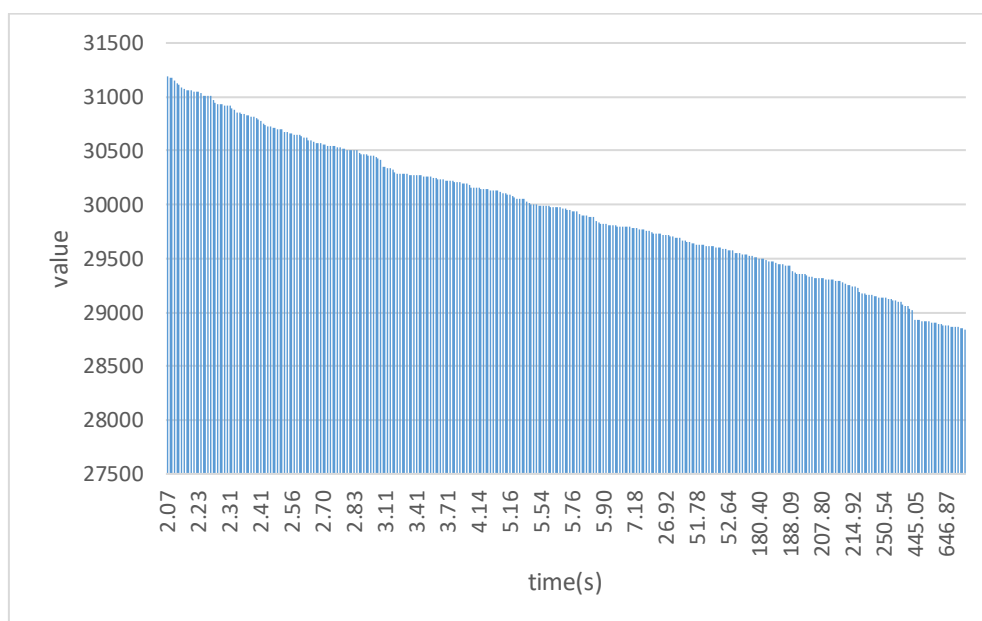


图 4.8 算例 RC1\_8\_2 计算收敛过程

#### 4.9.5 HLS 在不同算例规模上的求解结果

表 4.2 AWLS 算法在各规模算例上的结果

值类别	数量	Router	HustSmart	FHDSolver	LKHSP	ALNS++	ILS-SP	GA+TBD
Score	All	2542	2161	2033	1483	1277	988	916
	200	502	445	400	230	203	214	286
	400	528	424	412	264	265	189	198
	600	492	432	413	310	298	181	154
	800	516	416	406	329	275	199	139
	1000	504	444	402	350	236	205	139
Average PI	All	0.07020	0.32656	0.44080	0.72556	1.03041	1.78362	2.12630
	200	0.01313	0.10211	0.08968	0.38058	0.33785	0.58617	0.18352
	400	0.05555	0.31936	0.27834	0.75235	0.62820	1.45071	1.05234
	600	0.07720	0.37719	0.48839	0.74805	0.81829	1.82886	2.19463
	800	0.09944	0.45702	0.60787	0.97131	1.30853	2.54337	3.36540
	1000	0.10566	0.37711	0.73972	0.77552	2.05916	2.50901	3.83561
Median PI	All	0.05342	0.20048	0.12109	0.56101	0.64619	1.54419	1.82687
	200	0.00870	0.01478	0.02392	0.25214	0.20794	0.43803	0.05708
	400	0.04267	0.20570	0.15010	0.71887	0.54213	1.33414	1.00447
	600	0.07654	0.24886	0.15199	0.54629	0.63661	1.61230	2.14736
	800	0.11161	0.34086	0.26075	0.90434	1.20379	2.19378	3.47286
	1000	0.10544	0.38369	0.41791	0.68447	1.95502	2.35089	4.21311

本小节中介绍的结果数据来自 12-th DIMACS VRP 实现挑战赛第二阶段发布的测试结果，HLS 算法的求解器名称为 HustSmart。表 4.2 中的数据来自比赛官方发布的在所有算例上的测试结果。测试是在新生成的 300 个算例上进行的，值类别列中包括三行不同的数据，Score 行的数值计算是 4.9.3 节中介绍的竞赛中求解器的打分规则进行的。Average PI 行是求解器在所有算例上的 PI 值的平均数，Median PI 行是求解器在所有算例上的 PI 值的中位数。对于以上三行类别的数值，数量列表示算例的顾客数量，表中接下来的列是七个不同的求解器的测试结果。在表中展示了在不同算例规模下所有求解器的运行结果。All 行表示在所有算例上的表现，200 表示顾客规模大小是 200 个的算例，以此类推。通过对比可以看到 HustSmart 求解器一共获得了 2161 分，排名第二，与排名第一的求解器 Router 的 2542 差距为 380。

图 4.9 中的圆环图表示了在所有算例上，每个求解器的得分占有所有求解器总分的百分比。可以看到 Router 求解器占据 22%，求解效果最好，其次是 HustSmart。与

第三名FHDSolver的差距为1%。前三名的求解器得分总数之和占有求解器的59%。从求解器的第四名开始，差距开始变得显著。

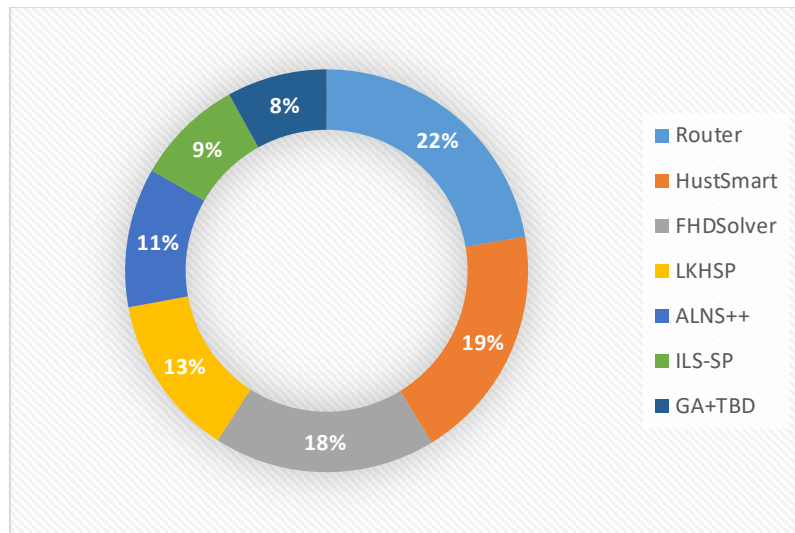


图 4.9 各个求解器得分占比圆环图

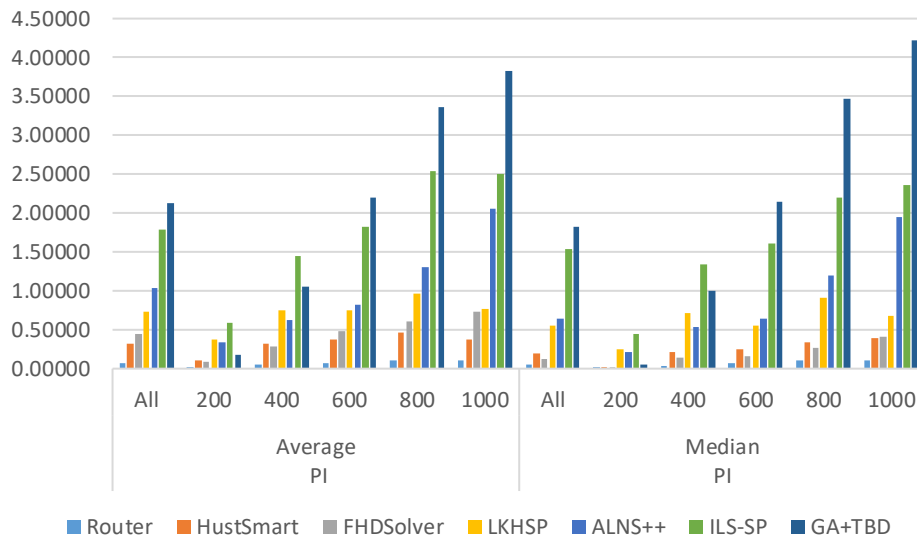


图 4.10 不同算例规模下 PI 值的平均数和中位数柱状图

在图 4.10 中的柱状图统计的是求解器测试结果的 PI 值的平均数（Average PI）和中位数（Median PI）。PI 值的定义在 4.9.3 中进行了介绍。测试结果基于所有的 300 个新算例，在主办方提供的计算机上进行的测试，测试环境的详细参数在 4.9.2 节中

列出。

从均值来看, HustSmart 求解器显著差于 Router 求解器, 优于 FHDSolver。并且在较小算例规模 200 顾客和 400 顾客规模算例上, 排名为第三, 但是差距较小。在 600、800 和 1000 顾客规模算例上排名第二并且显著优于 FHDSolver。从中位数来看, HustSmart 求解器 PI 值的中位数排名为第三, 差于 Router 和 FHDSolver。在 200 顾客规模算例和 1000 顾客规模算例上的求解效果排名第二, 其余规模算例上排名第三。

综合来看, HustSmart 求解器在较大规模算例上表现更好, 对于小规模算例求解的效果略差。集合算法开发过程中的特征可以对算法进行相应修改。分析原因主要如下: 对于小规模算例上算法收敛都很快, 很多算例在一秒钟之内就可以得到使用精确算法确定的下界。所以算法的求解速度和初始化部分的耗时有很大的关系。可以在初始化部分对算法进行进一步加速优化。

#### 4.9.6 HLS 在不同类型算例上的结果分析

表 4.3 HLS 算法在不同顾客分布算例上的结果

值类别	CR	Router	HustSmart	FHDSolver	LKHSP	ALNS++	ILS-SP	GA+TBD
Score	All	2542	2161	2033	1483	1277	988	916
	C	730	<b>782</b>	756	435	402	363	332
	R	893	705	605	558	442	318	279
	RC	919	674	672	490	433	307	305
Average PI	All	0.07020	0.32656	0.44080	0.72556	1.03041	1.78362	2.12630
	C	0.05545	0.12913	0.00825	0.81507	0.75173	1.54691	1.79532
	R	0.09685	0.39439	0.73466	0.60367	1.18453	1.92727	2.47317
	RC	0.05829	0.45614	0.57949	0.75795	1.15496	1.87669	2.11041
Median PI	All	0.05342	0.20048	0.12109	0.56101	0.64619	1.54419	1.82687
	C	0.02344	0.00943	0.00987	0.41438	0.18580	0.57950	0.82758
	R	0.09453	0.30272	0.42968	0.55659	0.89506	1.85919	2.53729
	RC	0.06386	0.40031	0.19810	0.68447	0.71751	1.87275	1.94747

本小节中介绍的结果数据来自挑战赛第二阶段发布的测试结果。在表 4.3 中是将测试的 300 个算例依据顾客的集散程度进行分类。CR 列表示顾客的分布规律类别, 可以分为三种, 在 3.8.1 节中对于三种分类进行了详细的介绍。与表 4.2 中一样, 包括求解器的得分 (Score), PI 值的平均数 (Average PI), PI 值的中位数 (Median PI)。可以看到在顾客分布呈现聚类特征时, HustSmart 获得了 782 分, 是所有求解器中得



分最高的。HustSmart 求解器更适合去求解这类算例。

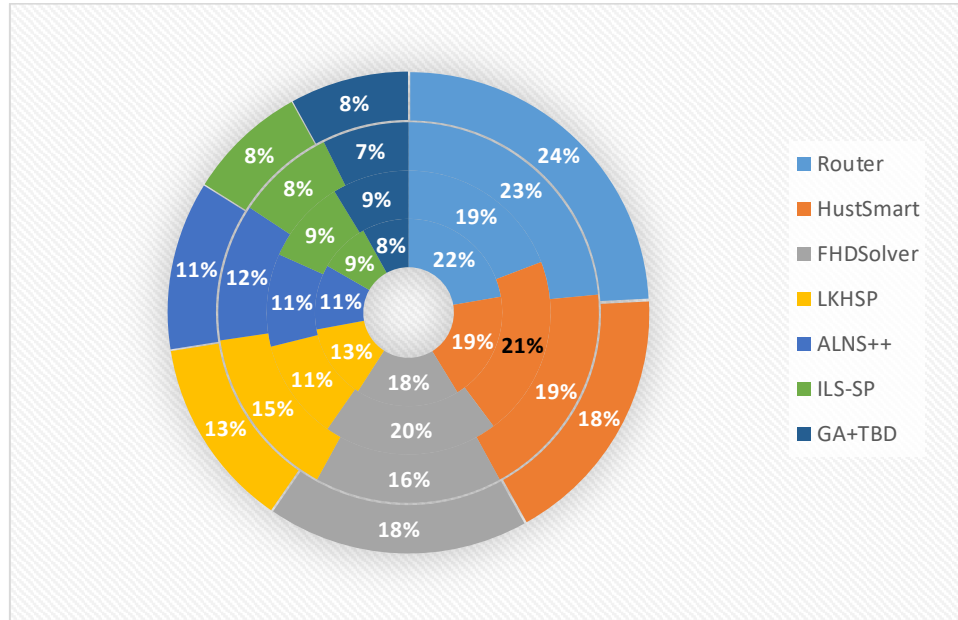


图 4.11 各个求解器在不同顾客分布算例得分占比圆环图

在图 4.11 中的圆环图中展示了所有求解器得分所占分数总和的比例。圆环图的圆环从内到外依次表示的分数比例为 All, C, R, RC。从图中可以看到 HustSmart 求解器在 C 类型算例，即顾客呈现聚类分布的算例中，得分占总分数的 21%。比例高于在所有实例上的得分占比 19%。但是在 RC 类型算例，即分布内既有聚类特征的顾客也有随机分布的顾客类型的算例中，得分占比 18%，差于在所有算例上的得分占比。而在 R，即顾客呈现随机分布的算例中，表现与所有算例上得分占比的 19%持平。从这个图中可以看出，HustSmart 求解器在单独求解 C 类型算例和 R 类型算例表现良好，在求解 RC 类型算例表现较差。

在图 4.12 中将算例依照顾客地理位置的分布进行分类比较，分别从 PI 值的平均数（Average PI）和 PI 值的中位数（Median PI）两个维度进行比较。从均值来看，在所有算例上，HustSmart 排名第二，在 C 类型算例上排名第三，R 类型和 RC 类型算例上排名第二。也就是说，与 FHDSolver 求解器比较来看，在所有算例中的 PI 值的和要大于 FHDSolver，但是在 C 类型顾客的总分却比 FHDSolver 高，即 HustSmart 在 C 类型的某些算例上得到了较高的分数，结合 4.9.4 节中，HustSmart 适合求解较大规模算例的结论，可以推出 HustSmart 求解器在 C 类型中较大规模算例上求解表

现好的结论。

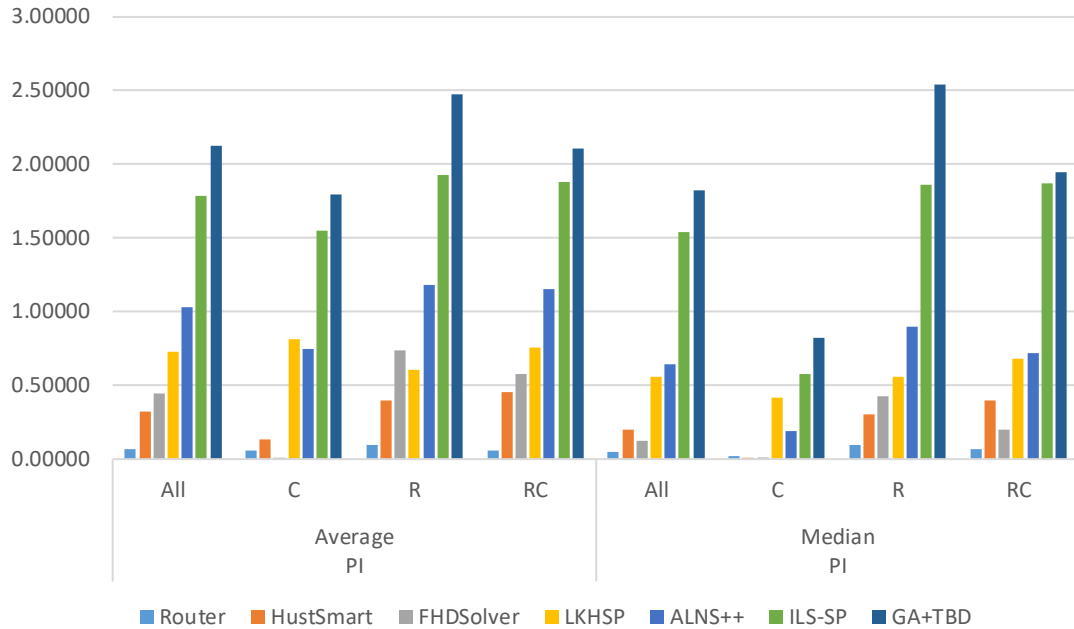


图 4.12 按顾客分布分类 PI 值的平均数和中位数柱状图

表 4.4 中的结果，是依据顾客的时间窗特征对算例进行了分类整理的数据。TW 列是算例顾客的时间窗的宽度类别。从表中可以看出 HustSmart 求解器在时间窗较窄（Narrow）类型算例上比较有优势，与 Router 的差距较小，并且优于 FHDSolver。但是在时间窗较宽松（Wide）的算例上较差，与 Router 差距较大，并且得分少于 FHDSolver。可以从下面图 4.13 中的圆环图中更加直观地看到这一特点。

表 4.4 HLS 算法在不同时间窗宽度算例上的结果

值类别	TW	Router	HustSmart	FHDSolver	LKHSP	ALNS++	ILS-SP	GA+TBD
Score	All	2542	2161	2033	1483	1277	988	916
	Narrow	1253	<b>1112</b>	956	719	683	568	409
	Wide	1289	1049	1077	764	594	420	507
Average PI	All	0.07020	0.32656	0.44080	0.72556	1.03041	1.78362	2.12630
	Narrow	0.07369	0.31954	0.62210	0.64327	0.94871	1.32837	2.35755
	Wide	0.06670	0.33357	0.25950	0.80785	1.11211	2.23888	1.89505
Median PI	All	0.05342	0.20048	0.12109	0.56101	0.64619	1.54419	1.82687
	Narrow	0.05338	0.24886	0.15903	0.65409	0.48644	1.28761	2.18771
	Wide	0.05342	0.18150	0.11226	0.47952	0.92201	1.75382	1.70899

在图 4.13 中依据求解器得分的占比进行对比。圆环图的圆环从内到外是表示不

同的算例集合分类,依次为在所有算例(All),较窄(Narrow)时间窗,和较宽(Wide)时间窗三个类别。在较窄时间窗类型算例的得分占比为 19%,优于 FHDSolver,但是在较宽的时间窗类型算例上,差于 FHDSolver,与排名第一的 Router 求解器的差距较大。

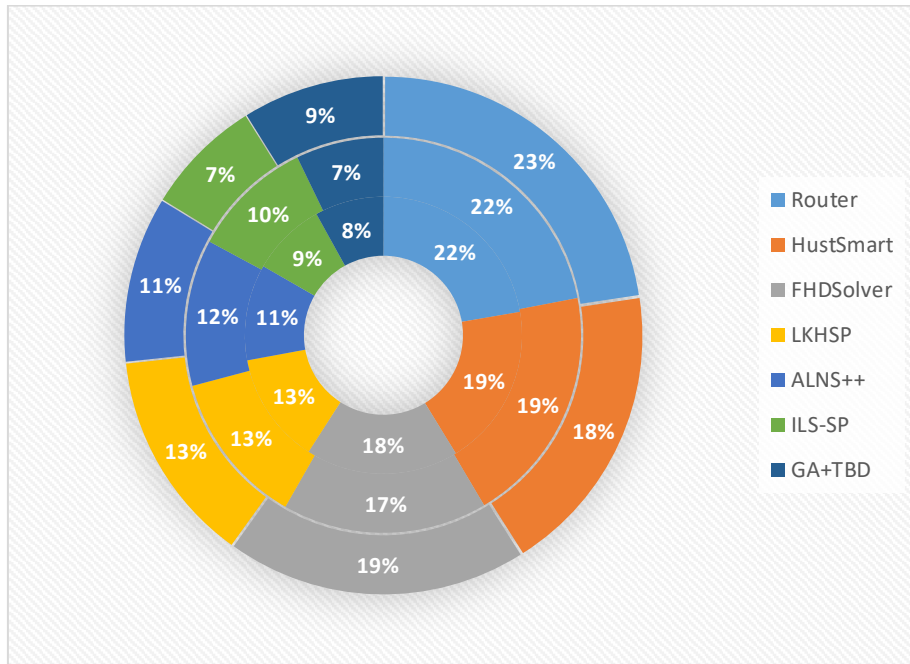


图 4.13 各个求解器在不同时间窗宽度算例得分占比圆环图

从上述结果比较结果可以得出结论, HustSmart 求解器在时间窗较窄的算例上表现更好。而时间窗较窄的算例在求解过程中往往更加容易违反时间窗约束,因此可以看出 HustSmart 在时间窗合法性的处理上更有优势。以上对求解器得分的占比有了整体的认识,图 4.14 中对求解算例的均值和中位数进行了更加详细的分析。

图 4.14 中的柱状图的数据来自表 4.4 中的 PI 值的均值(Average PI)和 PI 值的中位数(Median PI)。从均值来看,在所有的测试算例上, HustSmart 排名第二,PI 的均值低于 0.5。其中在顾客时间窗较紧张的算例上, HustSmart 求解器和 FHDSolver 的均值都超过了 0.5,显著差于 Router。在顾客时间窗比较宽松的算例上,均值低于 0.5。结果差于 FHDSolver。从中位数来看, HustSmart 求解器 PI 值的中位数均高于 FHDSolver,但是在所有算例集上的结果优于 FHDSolver,可以推出 HustSmart 在 PI

值高于中位数的一半算例中，获得了较好的分数。

综合 4.9.4 节和 4.9.6 节的数据分析，可以初步得出结论，HustSmart 在顾客分布呈现聚类特征，顾客时间窗较窄，算例规模较大的算例上表现良好。而在顾客时间窗宽松，顾客分布中有聚类加随机混合特征，算例规模较小的算例上表现较差，可以针对这部分算例做针对性优化与改进。

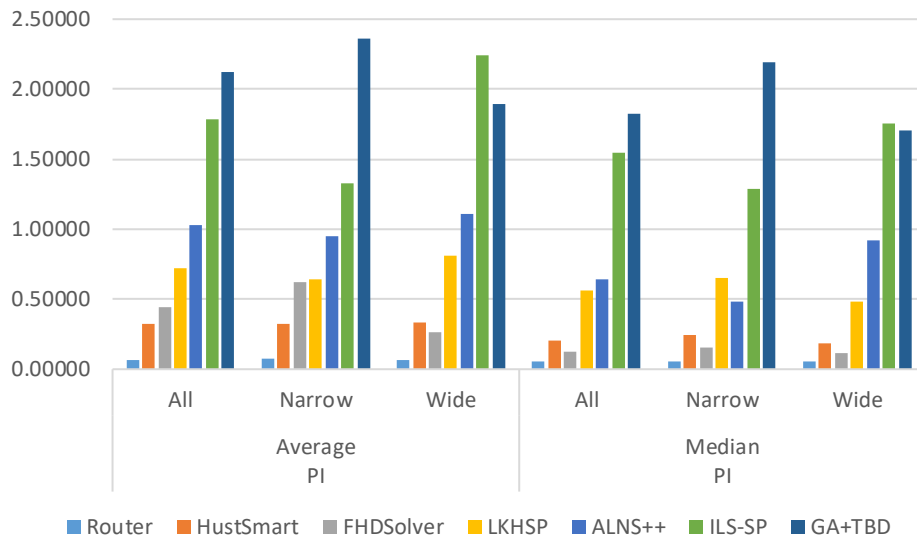


图 4.14 不同时间窗宽度分类中 PI 值的平均数和中位数柱状图

## 4.10 本章小结

本章介绍的算法优化目标是最小化所有路径的总长度，该算法在两个已经发表的 SISR 算法和 EAMA 算法的基础上进行改进。算法使用不同的路径数将解空间进行划分用以提高搜索的效率，并且满足了 EAMA 中交叉算子的要求。因此，在算法的遗传框架中设计了多个种群，每个种群中维护路径数相等的解。算法流程为可以概括为：首先使用 4.2 节中的种群集合初始化算法生成多个种群，使用 4.3 节中的简单局部搜索对种群集合中的解进行提高。最后对于每一个种群，都调用 4.7 节中的混合局部搜索算法进行优化。首先调用 4.4 节中的算法找到要搜索的种群，再调用 4.5 节中改进的 SISR 算法和 4.6 节中改进的 EAMA 算法对每一个种群进行优化。在 4.8 节中对实现的算法的数据结构进行了详细说明，并且对算法进行了单元测试。

## 5 总结与展望

### 5.1 主要内容以及结论

带有时间窗的车辆路径问题是物流配送行业的核心难题之一，深入地研究 VRPTW，能够为物流行业提供更高效率的、更优质的配送服务。不仅如此，对于提升行业运转效率和客户体验，研究该问题也有很大的现实意义。在我国碳中和发展转型的背景下，研究 VRPTW 可以为减少化石能源的消耗、优化物流行业从业者的工作体验提供有益的帮助与支持，有助于计算机学科与传统运输行业的交叉研究。此外，将求解 VRPTW 的算法进行精益求精的提升，能够推动启发式优化算法的发展，对求解 NP-hard 类问题的方法提出新思路。

求解 VRPTW 的主要难点来自该问题的强约束，包括时间窗约束和容量约束两个方面。在路径长度最小化问题中，两种约束的违反量都比较集中。随着路径数越来越少，单条路径上的顾客越来越多，导致容量约束很容易违反，特别是很多算例中，求解到的最小路径数已经到达理论的下界，这类型算例优化路径数就变得格外困难。最重要的是，单条路径上的顾客增多的情况下，路径上违反时间窗的概率也大大增加，对于时间窗惩罚的修复带来巨大困难。

本文重点研究了求解 VRPTW 启发式算法，针对 VRPTW 的两个目标，分别设计了两个启发式算法。

### 5.2 主要创新点

为提升解决求解 VRPTW 问题中路径数量最小化和路径总长度最小化的问题的启发式算法的求解效率与优度，本文中提出了以下新的方法与思路：

(1) 在路径数量最小化问题中，本文设计了一种顾客加权和路径加权相结合的局部搜索算法。通过定义顾客权重和路径权重的初始值以及增长时机，能够更好地衡量顾客的互斥关系以及路径中时间窗惩罚消除的难易程度。在此基础上进行局部搜索具有良好的疏散性，是 VRPTW 问题的强约束难点对应的解决策略。实验结果表明，该算法显著提高了 VRPTW 中路径数量最小化问题的求解效果。

(2) 在 VRPTW 路径数量最小化算法的修复算法中, 本文提出了对不同邻域动作的缩减邻域范围的方法, 这些方法对于提高邻域评估部分的效率有很大帮助。采用本文中的方法进行邻域缩减。可以帮助算法尽可能地减少探索无用的解空间, 从而提升了搜索的效率。

(3) 在探索路径数量最大下界过程中, 本文将 VRPTW 松弛之后转化为最大团问题, 并找到了一些通过松弛时间窗约束难以确定的路径数的最大下界。求得的下界可以知道路径数量最小化算法的运算, 并且是 VRPTW 问题向其他问题转化, 进而求解其他问题的一个思路, 部分问题通过转化可能会获得良好的求解效果。

(4) 在 VRPTW 的路径总长度最小化问题中, 本文将 SISR 算法和 EAMA 算法进行结合, 设计了混合局部搜索算法。并且在遗传算法部分设计了多个种群, 将解空间进行划分, 既提高了搜索的效率, 又能满足 EAMA 对于参与交叉的两个解的路径数量相等的需要。这种方法能够更全面地探索解空间, 一定程度上避免重复搜索重叠的解空间。

## 5.3 未来工作展望

关于 VRPTW 问题研究的未来工作方向主要有以下几点:

(1) 在局部搜索方面的改进可以向着探索更多的不可行解区域去探索。因为 VRPTW 问题的强约束, 导致不管是路径长度最小化还是路径总长度最小化, 都时刻被解的可行性所约束, 与不可行的解空间相比, 可行解的空间范围要小得多, 但是因为邻域动作步幅较小的原因, 不能跨过这些不可能解区域, 所以增强算法对不可行解的探索可能可以显著提升算法的优度。

(2) 研究更大规模问题的 VRPTW, 例如将顾客规模达到 10000 甚至更大的规模。随着我国物流行业的迅猛发展, 物流公司的规模不仅仅局限于一个仓库, 1000 个顾客规模, 所以需要加大 VRPTW 问题研究的规模, 以满足工业界的需求。

(3) 在经典的 VRPTW 问题, 只关注路径长度, 从而忽略车辆的代价是不合理的, 这样的设计是不符合现实需求的, 一个物流运输的公司不可能不考虑车辆的成本, 比如车辆的维护成本, 司机的薪资等, 所以需要设计一个科学的优化函数, 将路

径长度和车辆数量的优化统一起来。

(4) 在容量约束部分, 考虑物体的三维体积的约束, 而不仅仅考虑物体的容量这一个维度的量, 这与实际问题不符合。其次在装卸货物的顺序上, 也需要进行优化, 例如根据配送顾客的顺序调整货物的装车顺序, 或者根据货物的密度安排不同的装货顺序。此外还可以考虑车辆的重心, 装在车上的货物的质心不能让货物和货车处于一个危险的状态, 要考虑装货的平衡性。

(5) 在算法工程化部分, 使用并行加速技术提升算法的搜索效率。例如使用 CPU 多线程, 将搜索任务或者邻域评估任务分配到不同的线程中, 同时搜索或者探索多个解空间, 提高搜索效率。或者利用 GPU 并行加速等手段, 进一步提高并行度, 但是由于搜索算法操作复杂程度比一般的显卡运算任务更加复杂, 使用该技术获得的算法收益有待商榷。

## 致 谢

时间总是在不经意间流逝,转眼本科毕业已经三年,当我再次写到论文致谢的时候,就意味着我的研究生学业也即将画上句号,同时也是我二十余载求学之路的句号。硕士论文即将完成,我要向所有帮助过我的人表示感谢。

首先,感谢我的导师吕志鹏教授,他带领我进入运筹优化领域,让我在领略算法魅力之美的同时修炼了内功。吕老师深耕启发式优化算法领域,对多种 NP-hard 问题求解有独到的见解。在我研究 VRPTW 期间,正值疫情,每次实验有了新的进展,都要与吕老师进行长时间通话以交流求解思路。吕老师思维活跃,并且经验丰富,每次通话都有新的收获,在此过程中我对于算法研究的兴趣也与日俱增。

我要感谢实验室的同门师兄,感谢苏宙行师兄在我科研道路上的支持,他对于算法实现和 C++语言有着丰富的经验,许多技术是通过阅读师兄的代码学到的。感谢丁俊文师兄,在论文修改期间,丁师兄认真查看我的论文,提出了若干修改意见,让我的论文更加完善。还有已经毕业的罗茂师兄给我介绍了很多 SAT 问题的知识,方圆师兄指导我做图着色基础训练,本科和我同院的李研师兄介绍我加入实验室,黄施豪师兄和我一起做华为的项目、一起参加 DIMACS 竞赛,张庆雲学姐指导我做 p-center, 和李春根师兄一起做华为的项目,还有宋迪师姐、袁楷文师姐、金琦师兄在我找工作期间提供了有力的帮助。


我要感谢和我一起从大工来华科的李课和宋良才,同期一起实习的张顺根,我们是最早来实验室实习的四人,一起在华科租房实习,让我初来华科也不觉得孤单。还有罗灿辉,孙致波、高岩和李思源,无论是一起学习、生活还是搞科研、找工作,他们都是最好的伙伴。我要感谢实验室的学弟学妹们,他们的加入让实验室更加的丰富多彩,充满活力。也要感谢张壬泓同学对我学习和生活上的照顾。

我要感谢我的好朋友好兄弟,杨心远,武小东,孙虎,白鑫,从初中毕业之后的十几年间,我们这帮人不仅是朋友,也像亲人。

最后我要感谢我的父母和家人,在我的求学路上,父母给与了我无私的付出与支持,无论在物质上还是精神上,他们永远是我坚强的后盾。



## 参考文献

- [1]. George B. Dantzig, John H. Ramser. The truck dispatching problem. *Management science*, 1959, 6(1): 80–91
- [2]. Michel W. P. Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 1985, 4(1): 285–305
- [3]. Jiu X. Zhao, Min J. Mao, Xi Zhao, Jian H. Zou. A hybrid of deep reinforcement learning and local search for the vehicle routing problems. *IEEE Transactions on Intelligent Transportation Systems*, 2020, 22(11): 7208–7218
- [4]. Martin Desrochers, Jacques Desrosiers, Marius Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations research*, 1992, 40(2):342–354
- [5]. Leonardo Lozano, Daniel Duque, Andrés L. Medaglia. An exact algorithm for the elementary shortest path problem with resource constraints. *Transportation Science*, 2015, 50(1): 348–357
- [6]. Brian Kallehauge, Jesper Larsen, Oli B.G. Madsen, Marius M. Solomon. Vehicle routing problem with time windows. *Column Generation*, 2005, 5: 67–98
- [7]. Niklas Kohl, Oli B. G. Madsen. An optimization algorithm for the vehicle routing problem with time windows based on Lagrangian relaxation. *Operations research*, 1997, 45(3): 395–406
- [8]. Ruslan Sadykov, Eduardo Uchoa, Artur Pessoa. A  **Bucket Graph-Based Labeling Algorithm with Application to Vehicle Routing**. *Transportation Science*, 2021, 55(1):4–28
- [9]. Diego Pecin, Artur Pessoa, Marcus Poggi, Eduardo Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, 9(1):61–100,2017
- [10]. Diego Pecin, Claudio Contardo, Guy Desaulniers, Eduardo Uchoa. New enhancements for the exact solution of the vehicle routing problem with time windows. *Inform Journal on Computing*, 2017, 29(3): 489–502
- [11]. Thangiah, Sam Rabindranath, Kendall E. Nygard, Paul L. Juell. Gideon: A genetic

- algorithm system for vehicle routing with time windows. In: Proceedings The Seventh IEEE Conference on Artificial Intelligence Application(AIAPP 1991), Miami Beach, Florida, USA, February 24-28, 1991: 322-328
- [12]. Yuichi Nagata, Olli Bräysy. A powerful route minimization heuristic for the vehicle routing problem with time windows. *Operations Research Letters*, 2009, 37(5): 333-338
- [13]. Yuichi Nagata, Olli Bräysy, Wout Dullaert. A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. *Computers & operations research*, 2010, 37(4): 724-737
- [14]. Martin M. Solomon. 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research* 35(2) 254–265
- [15]. Billy E. Gillett, Leland R. Miller. A heuristic algorithm for the vehicle-dispatch problem. *Operations research*, 1974, 22(2): 340-349
- [16]. John E. Beasley. Route first—cluster second methods for vehicle routing. *Omega*, 1983, 11(4): 403-408
- [17]. Thibaut Vidal. Split algorithm in  $O(n)$  for the capacitated vehicle routing problem. *Computers & Operations Research*, 2016, 69: 40-47
- [18]. Gerard A. P. Kindervater, Martin W. P. Savelsbergh. Vehicle routing: handling edge exchanges. *Local search in combinatorial optimization*, 1997: 337-360
- [19]. Jean Y. Potvin, Jean M. Rousseau. An exchange heuristic for routeing problems with time windows. *Journal of the Operational Research Society*, 1995, 46(12): 1433-1446
- [20]. Vinícius R. Máximo, Mariá C.V. Nascimento. A hybrid adaptive iterated local search with diversification control to the capacitated vehicle routing problem. *European Journal of Operational Research*, 2021, 294(3): 1108-1119
- [21]. Stefan Ropke, David Pisinger. A unified heuristic for a large class of vehicle routing problems with backhauls. *European Journal of Operational Research*, 2006, 171(3): 750-775
- [22]. Jan Christiaens, Greet Vanden Berghe. Slack induction by string removals for vehicle routing problems. *Transportation Science*, 2020, 54(2): 417-433
- [23]. Fred Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 1996, 65(1-3): 223-253
-

- [24]. Thibaut Vidal. Hybrid genetic search for the CVRP: Open-source implementation and SWAP\* neighborhood. *Computers & Operations Research*, 2022, 140: 105643
- [25]. Yiannis A. Koskosidis, Warren B. Powell, Marius M. Solomon. An optimization-based heuristic for vehicle routing and scheduling with soft time window constraints. *Transportation science*, 1992, 26(2): 69-85
- [26]. Florian Arnold, Kenneth Sörensen. Knowledge-guided local search for the vehicle routing problem. *Computers & Operations Research*, 2019, 105: 32-46
- [27]. Florian Arnold, Ítalo Santana, Kenneth Sörensen, Thibaut Vidal. PILS: Exploring high-order neighborhoods by pattern mining and injection. *Pattern Recognition*, 2021, 116: 107957
- [28]. Pablo Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 1989, 826(1989): 37
- [29]. Hui Z. Zhang, Qin W. Zhang, Liang Ma, Zi Y. Zhang, Yun Liu. A hybrid ant colony optimization algorithm for a multi-objective vehicle routing problem with flexible time windows. *Information Sciences*, 2019, 490: 166-190
- [30]. Yuan Wang, Ling Wang, Zhi P. Peng, Guang C. Chen, Zhao Q. Cai, Li N. Xing. A multi ant system based hybrid heuristic algorithm for vehicle routing problem with service time customization. *Swarm and Evolutionary Computation*, 2019, 50: 100563
- [31]. 黄楠. 复杂多行程车辆路径问题的精确算法研究[博士学位论文]. 武汉: 华中科技大学, 2021
- [32]. 王建新. 面向物联网客户动态需求的车辆路径问题研究[博士学位论文]. 重庆: 重庆大学, 2021
- [33]. 苏欣欣. 带时间窗和人力资源分配的车辆路径问题研究[博士学位论文]. 武汉: 华中科技大学, 2021
- [34]. 宋亮. 面向物流行业的车辆路径问题数学建模与近似算法设计[博士学位论文]. 哈尔滨: 哈尔滨工业大学, 2018
- [35]. 宋强. 基于群体智能优化算法的多行程车辆路径问题的研究[博士学位论文]. 武汉: 武汉理工大学, 2018

- [36]. Florian Arnold, Michel Gendreau, Kenneth Sörensen. Efficiently solving very large-scale routing problems. *Computers & operations research*, 2019, 107: 32-42
- [37]. Accorsi, Luca, Daniele Vigo. A fast and scalable heuristic for the solution of large-scale capacitated vehicle routing problems. *Transportation Science*, 2021, 55(4): 832-856
- [38]. Soeffker, Ninja, Marlin W. Ulmer, and Dirk C. Mattfeld. Stochastic dynamic vehicle routing in the light of prescriptive analytics: A review. *European Journal of Operational Research*, 2022, 298(3): 801-820
- [39]. Lim, Andrew, and Xingwen Zhang. A two-stage heuristic with ejection pools and generalized ejection chains for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 2007, 19(3): 443-457
- [40]. Chu M. Li, Hua Jiang, and Felip Manyà. On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem. *Computers & Operations Research*, 2017, 84: 1-15
- [41]. Hermann Gehring, Jörg Homberger. A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In: *Proceedings of EUROGEN99(ECGA99:1999)*, Brussels, Belgium, June 30-July 2, 1999:57-64
- [42]. David Pisinger, Stefan Ropke. A general heuristic for vehicle routing problems. *Computers & operations research*, 2007, 34(8): 2403-2435
- [43]. Toshihide Ibaraki, Shinji Imahori, Koji Nonobe, Kensuke Sobue, Takeaki Uno, Mutsunori Yagiura. An iterated local search algorithm for the vehicle routing problem with convex time penalty functions. *Discrete Applied Mathematics*, 2008, 156(11): 2050-2069
- [44]. Andrew Lim, Xing W. Zhang. A two-stage heuristic with ejection pools and generalized ejection chains for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 2007, 19(3): 443-457
- [45]. Timo Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 2013, 41(6): 611-614