基于传统 TSP 的机器人路径规划问题优化

摘要

传统 TSP(Traveling Salesman Problem)是一个经典的组合优化问题，本文研究的是基于传统 TSP 的变种问题 TRPP(Traveling Repairman Problem with Profit)，即一个修理工访问给定的多个城市点收集利润，利润值随时间的推移线性下降，只要求获得最大利润而不要求遍历访问所有节点。该问题在现实生活中的应用包括物流机器人的调度与路径规划，包裹配送的任务分配问题以及灾后救援的指派调度问题等。由于 TRPP 是 NP-hard 问题，难以精确求解，因此本文提出一种基于强化学习的变邻域启发式算法框架，该框架对现有常用局部搜索算符进行合理排序组合，同时提出一种新的 k-opt 搜索算符，该算符在局部搜索环节能够进行自我学习与记忆，用于反向调节改进搜索与扰动机制，在给定算例测试集上表现突出，在可接受的时间内全面超越了目前已有算法的最佳结果，并通过多组实验验证了算法较强的搜索能力与较高的搜索速度。

关键词：强化学习；局部搜索；组合优化

Optimization of Robot Path Planning Problem Based on Traditional TSP

Abstract

The traditional TSP (Traveling Salesman Problem) is a classic combinatorial optimization problem. This paper studies the TRPP (Traveling Repairman Problem with Profit) based on the traditional TSP, that is, a repairman visits a given number of cities to collect profits and the profit declines linearly with time. Its objective function is to obtain maximum profit without requiring all nodes to be visited. The application of this problem in real life includes the scheduling and path planning of logistics robots, the task distribution problem of package delivery and the problem of assignment and scheduling of post-disaster rescue. Since TRPP is an NP-hard problem, it is difficult to solve it accurately, this paper proposes a variable neighborhood heuristic algorithm framework based on reinforcement learning. The framework rationally sorts and combines existing local search operators. In addition, it uses a new k-opt search operator, which can perform self-learning and memory in the local search procedure, and the knowledge learned is used for back propagation to improve the search and disturbance mechanism. It has outstanding performance on the test bench and surpassed the best-known results in reasonable time, which verified the strong search ability and impressive search speed of the algorithm.

Keywords：Reinforcement Learning; Local Search; Combinatorial Optimization

目录

1. 绪论

1.1 课题背景与研究意义

TSP (Traveling Salesman Problem)问题是经典的组合优化问题，在实际生活中具有广泛应用，包括交通运输，物流调度，生物迁徙以及电路设计等。传统 TSP 问题可以描述为给定若干个城市以及两两城市间的距离，问题的目标是从一个随机的城市出发找到能够遍历所有城市并回到该出发城市的最短闭合路径，其中除出发城市以外每个城市的访问次数有且只有一次。TSP 问题已经被证明是 NP-hard 问题，因此从理论计算机层面难以解决。

在实际应用中，为了更贴切地描述某些特定场合下特定的数学模型，传统 TSP 问题衍生出许多变种问题，其中 TRP (Traveling Repairman Problem)问题是将 TSP 问题中两点间的路径长度表示为两点间相互对称的行程时间，并满足三角不等式法则，TRP 问题求解的是一条遍历所有城市并使得总延迟时间$\sum_{i=1}^{n} l(i)$最小的哈密尔顿回路，$l(i)$表示到达城市i的时刻，因此 TRP 问题也称为旅行商配送问题 TDP (Traveling Deliveryman Problem)或最小延迟问题 MLP (Minimum Latency Problem)，这类问题被证明是 NP-complete 问题；同时，TSP 问题具有一种特殊变种 TSPP (Traveling Salesman Problems with Profits)问题，该问题与 TSP 的区别在于旅行商不需要遍历所有城市，且每个城市附带特定的利润值，问题的目标是寻找能够获得最大利润前提下的最短路径，这是一种双重目标且相互对立的 TSP 问题，旅行商需要在付出路程代价换取利润最大化的同时兼顾路程的最小化，这类问题的实际应用包括任务的调度分配以及货物配送运输等。

然而随着实际应用场景的复杂化，单纯的 TSP、TRP 或 TSPP 问题都无法满足理论与实际层面更加特殊化与专门化的需求，因此引文[1]将 TRP 与 TSPP 问题相结合提出了 TRPP (Traveling Repairman Problem with Profits)问题。在 TRPP 问题中，每一个城市i依然附带特定的利润值p_i，将出发点指定为仓库，其利润$p_0 = 0$，修理工从仓库出发，在城市i能够获得的利润为$p_i - l(i)$，$l(i)$同样表示修理工到达该城市的时刻，每个城市至多只能被访问一次，修理工不需要访问所有城市且不需要返回仓库，即当对于某些尚未访问的城市i，修理工到达该城市的$l(i)$大于或等于其自身附带的利润值p_i时，意味着修理工将不能在该城市获得正向利润，因此该点将不会被访问。TRPP 问题的目标是寻找能够获得最大利润的城市序列，当每个城市的利润趋于无穷时，修理工不论花费多长时间到达都能够获得正向利润，则问题简化为 TRP 问题。

由引文[1]可知在计算复杂度上 TRPP 问题也属于 NP-hard 问题，但其除了具有作为典

型组合优化问题的研究作用之外，更大的意义在于贴切描述了许多实际生活的问题。常见的一个应用场景是灾害发生后的紧急救援调度工作，例如一片由多个村庄组成的区域遭遇地震袭击后，幸存者急需人道主义援助，而一支救援队到达某村庄的时刻越早，该村庄获救的人数就越多。假设村庄i具有p_i位幸存者等待救援，救援队到达该村庄的时刻是l_i，且每消耗一个单位时间会有一位幸存者失去生命，则这支救援队的目的是让$\sum_i[p_i - l_i]$取得最大值。

1.2 国内外研究现状

1.2.1 相关节点路径问题（Node Routing Problem）研究现状

目前国内外已有若干研究机构或个人对与 TRPP 相关的节点路径问题进行了不同程度与不同方向的研究，并提出诸多理论方法，其中主要分为精确算法与启发式算法两大类。

引文[2]中作者介绍了一种不确定行程耗时下的 TRPP 随机变种问题，将其表示为非线性整数模型，并通过波束搜索进行启发式求解，其与 TRPP 的唯一区别是行程耗时的不确定性；引文[3]中这种随机变种问题被延伸为灾后救援中的车队调度问题，该文作者假设车队中所有车辆的容载量无限，并采用迭代贪心启发式算法求出近似最优解。另外，上文提到的 TRP 问题近年来得到了大量研究，由于其属于时间相关的路径问题，因此与 TRPP 具有高度相似性。目前已有的针对 TRP 问题的求解算法包括近似法[4]、分支定界法[5]、动态规划法[6]、贪心自适应随机搜索过程/变邻域搜索(GRASP/VNS)算法[7]、遗传算法[8]、以及将迭代局部搜索(ILS)与 GRASP 和 VNS 相结合的一种混合算法[9]。

在已有的路径利润问题(Routing Problems with Profits)中有两大类问题与 TRPP 具有高度相似性，即 PTP(Profitable Tour Problem)[10]与 OP(Orienteering Problem)[11]。PTP 问题的目标函数是找到能够使总利润与总行程耗时的差值最大化的路径，PTP 与 TRPP 的主要区别是其目标函数中使用的是行程时间而非到达时刻，引文[12]中作者针对 PTP 问题提出一种风险规避构想，即假设所有行程时间都具有不确定性，并基于此提出一种结合禁忌搜索的遗传算法以求解问题。在 OP 问题中，旅行商的任务是确定一条路径，使其在给定路径长度的约束下访问城市并获得尽可能多的利润，TRPP 与之不同的是利润的时变性且在路径长度上没有约束。已有的求解 OP 问题的方法有分支切割法[13]、结合路径重连接的 GRASP 算法[14]、文基因算法[15]及粒子群优化算法[16]等。

另一个与 TRPP 高度相关的是 MCPTDR (Maximum Collection Problem with Time-dependent Rewards)问题[17]。在此问题中，节点上的收益是随时间推移而线性下降的函数，

其目标依然是使利润收益最大化。MCPTDR 与 TRPP 的主要区别是利润的下降率，因为 TRPP 假设每一个节点的下降率都是 1。引文[17]中采用分支定界算法求解小规模算例的最优解，并对大规模算例采用基于惩罚的贪心启发式算法。引文[5]又对 MCPTDR 问题进行延伸，使用多代理处理需求的方法并提出两段式启发算法在合理时间内求出足够优的解。

1.2.2 TRPP 问题研究现状

根据目前已有的文献资料，研究 TRPP 问题的文献有引文[1]，[18]和[19]。

引文[1]首次提出对此问题的描述与研究，该文作者介绍了 TRPP 的数学模型，并基于 TSPLIB 算例集制作了 6 种不同规模大小的 TRPP 算例测试集，算例中城市点数量$n \in \{10,20,50,100,200,500\}$，作者对规模较小的算例($n \leq 20$)进行数学建模并在固定访问点数量的前提下使用 CPLEX 求解数学模型计算最优解，对较大规模算例($n \geq 50$)采用多邻域禁忌搜索算法求得近似最优解。论文提出一种构造非零解的方法，该非零解随后通过禁忌搜索算法进行系统性改进。具体方式是首先将仅有一个仓库点的解作为初始零解，并提出一种访问点比率用于表征某个点被添加进入当前解的概率大小，即某个城市点的比率越大表示该点作为下一个访问点被添加进入当前解后实现改进的可能性更大，则该点会有更大的概率被选中，从当前初始零解开始每次添加一个访问点获得一个新解，此过程称为构造阶段。作者通过对比贪心算法与这种基于插入思想的初始解构造方法，并通过试验证明基于此方法产生的初始解进行局部搜索后能够获得均值更优的解。随后该算法将当前解作为输入并进行两组局部搜索，第一组局部搜索过程通过修改访问点的集合来获得不同邻域结构，包括删除、添加和替换三种算符；第二组局部搜索过程包括两点交换，相邻交换、前移访问点、后移访问点、删除并重新插入访问点、2-opt 以及广义 Or-opt 七种算符，通过调整访问点顺序以获得不同的邻域集合。邻域结构搜索顺序为：首先调整访问点顺序，然后改变访问点的点集，每一个禁忌搜索迭代周期都是一次完整的变邻域下降过程，此过程结束后应用每一个在邻域结构中找到的最佳改进方案。

与采用 CPLEX 求解器时需要将访问点数量作为输入的引文[1]不同，引文[18]直接采用 GRASP 与 ILS 相结合的多起点混合搜索算法，即每一个迭代过程包含两个步骤，步骤一是解的构造，步骤二是解的改进。在构造解阶段，论文采用的是一种贪心随机方法，具体步骤是基于当前修理工所在的城市点位置，找出若干个距离当前位置最近的城市点，在其中进行随机选择并加入当前解中，再基于该选择的点的位置继续选择下一个城市点，如此迭代直至所有可访问的城市点均已被添加则视为初始解构造完成。进入第二阶段后，将当前构造所得的解作为输入，通过迭代局部搜索过程完成进一步搜索实现对初始解的改进，论

文中的迭代局部搜索方法集成了禁忌增强的变邻域下降算法和自适应扰动机制，变邻域下降过程用于将此次迭代过程中找到的当前解的最佳邻域解保存在短期禁忌表中，若此邻域解优于当前解，则用其替代更新当前解并基于这个新解重启一次完整的局部搜索，否则基于原解继续进行下一个局部搜索过程，此方法能够防止搜索陷入局部循环，作者也通过实验证明该方法能够有效提升算法搜索过程的效率；自适应扰动机制也是迭代局部搜索过程的关键环节，该机制能够根据当前的搜索空间调整扰动操作以实现定向扰动，这也是保证程序不易陷入局部最优的有效手段。作者通过一系列实验证明了其提出的混合算法对比引文[1]单纯禁忌搜索的优越性，该方法使得搜索结果获得显著提升，在给定的 120 个算例中改进了 46 个已知最优结果。

引文[19]在已有算法基础上进一步提出一种基于种群的混合进化搜索框架，该框架在通用文基因搜索框架的基础上加以改进，结合基于种群搜索方法和局部优化使原框架的搜索能力获得较大提升。框架主要分为三个部分：初始种群的生成同样采用贪心随机算法；专用可变邻域搜索用于种群的进一步改进；最后迭代进行种群的交配重组和种群局部进化过程，交叉算子结合两个随机选择得到的亲代解进行特定的交叉遗传操作获得一个子代解。亲代的交叉遗传方式也是论文的核心内容之一，作者提出两种新的交叉算子，能够在保留两个亲代优秀基因的同时对亲代解产生扰动进而保证搜索的广度，并使用自适应规则对两种交叉算子进行选择，即算子被选中执行操作的概率随其产生改进效果的增大而提高，遗传后的局部优化过程同样采用变邻域搜索算法实现对重组生成的子代解的改进，并基于改进后的子代解进行一次种群更新，整个算法流程采用时间作为终止条件，这种方法对三种较大规模共 39 个算例($n \geq 100$)的已知最优结果做出了改进。

尽管已有的三篇 TRPP 文献提出三种不同的启发式搜索方法，但在搜索过程中均采用了多种邻域结构且均通过算符进行随机扰动以跳出局部最优，因此可以将这三种方法都视为广义变邻域搜索的变种算法。

1.3 本文的主要研究内容

本文的主要工作首先是设计了一种用于求解 TRPP 问题的基于强化学习的启发式算法框架，该框架从构造初始解出发，进行可变邻域结构的局部搜索和定向扰动，在局部搜索环节提出一种新的基于蒙特卡洛搜索树的邻域结构，同时对搜索过程中获得的所有改进解的解结构进行学习和记忆，这种方法对于后续搜索与扰动操作具有重要的指导作用，能够大幅提高搜索效率与搜索能力。同时在已有算例测试集上进行算法框架的测试实验，实验

表明本算法在较小规模($n \leq 50$)算例上能够快速获得最优解，在较大规模($n \geq 100$)算例上的表现均不同程度优于现有针对相同问题的传统局部搜索算法和遗传算法框架，在给定的算例中改进了 29 个已知最优结果且其余算例全部与已知最优解相平，本文内容安排如下：

第一章主要介绍 TRPP 问题的研究背景及研究意义，并概述了国内外对相关节点路径问题与 TRPP 的研究现状，同时对本文的主要研究内容进行简要概括；

第二章对算法的完整流程及框架要点进行整体介绍；

第三章重点介绍了本文提出的一个新的局部搜索算符 k-opt 以及强化学习在本工作中的体现；

第四章通过三个实验证明算法框架与细节的有效性与鲁棒性以及局部搜索算符组合顺序的合理性；

第五章对全文工作进行总结，并对未来再进一步深入开展研究工作进行思考与展望。

2. 算法框架概述

2.1 整体算法流程

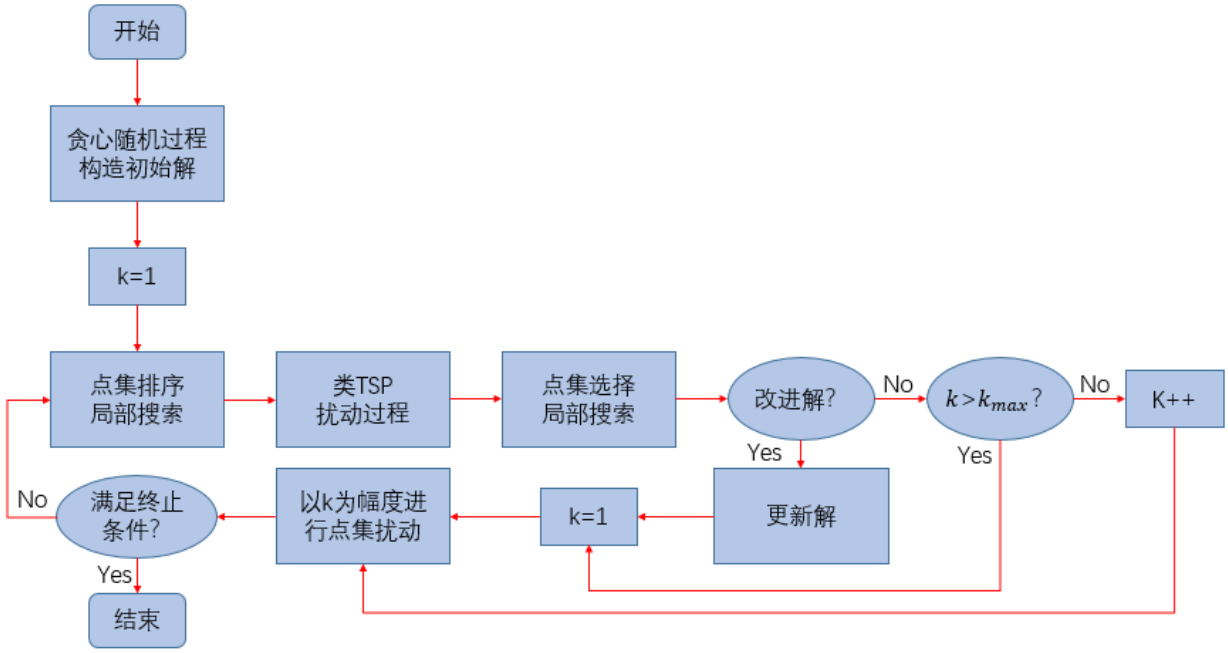

图 1：完整算法流程图

由于 TRPP 问题的目标是在保证所有城市点至多被访问一次的前提下获得最大利润值，且不要求遍历所有城市点，因此求解此问题的算法具有两个层次的任务：选择访问点以及调整访问点的顺序。针对这两个具有较低相关程度的任务，本文提出一种基于强化学习思想的多邻域迭代搜索算法框架。算法指定修理工(Repairman)从初始点即仓库出发，基于当前所在位置以贪心随机策略迭代选择下一个目标点至遍历所有城市点时视为初始解构造

完成；此时所有给定的城市点一同组成访问点集，在此基础上开始点集排序局部搜索过程，此过程类似传统 TSP 问题的局部搜索，但二者目标函数具有本质上的区别，传统 TSP 只需考虑调整访问点顺序以实现访问点序列中两两之间路径长度的叠加之和最小，即$\sum_{i=0}^{n} d_{i\to(i+1)}$，其中$d_{i\to(i+1)}$表示当前解序列中第$i$个点到第$i+1$个点之间的距离；而 TRPP 问题在此过程计算的是访问点序列中修理工从仓库出发到达当前点所经过的累积距离叠加之和，即$\sum_{i=0}^{n}(\sum_{j=0}^{i} d_{j\to(j+1)})$，修理工每到达城市点$i$时都需要将他从仓库出发到当前点所经过的所有路径长度累加，而代价计算公式的区别也造成了传统 TSP 与 TRPP 在点集排序局部搜索中策略的差异；完成一轮点集排序搜索后，为了防止搜索陷入局部最优，算法采用定向扰动策略，同时采用概率机制接收较差的解，使解序列获得更多的可能性从而逐渐向全局最优区域靠近；当点集排序搜索过程与定向扰动进行到一定程度时，认为已基本获得当前访问点序列所能找到的最佳情况，随后进入调整访问点的搜索过程，通过增删或替换等操作调整修理工所访问的城市点，避免出现付出路程代价后没有利润回报的情况；至此视为完成一轮完整搜索过程，若此时较上一轮搜索获得的解有所改进，则进行一次调整点集的扰动，否则当前有几轮搜索未改进就进行几次调整点集扰动，这种根据解的改进情况进行扰动的方式既能够保证不遗漏某个局部区域可能的更优解，同时又能及时跳出局部最优防止搜索机制陷入死循环；最后当算法满足设定的终止条件(本算法中设置的终止条件为搜索时长)时，结束程序并将搜索获得的最佳结果输出保存。

2.2 构造初始解

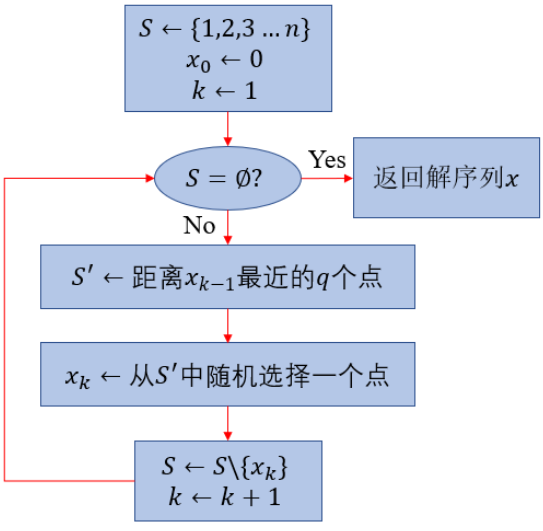

图 2：初始解构造流程图

本文采用贪心随机算法构造初始解，具体方法是以仓库点为初始零解，记为当前点x_0，初始化$k=1$，所有可访问的城市点构成集合S，修理工从仓库出发基于贪心规则选择距离

x_{k-1}最近的q个点作为候选点集，再从点集中以随机方式选择一个点x_k加入解序列，同时在点集S中删除x_k，随后另k自加并基于x_k继续以相同规则选择下一个目标，循环迭代直至点集S为空时完成初始解的构造，获得一个完整且符合问题约束的解序列。

2.3 点集排序搜索过程

在给定访问点集的局部搜索环节，本文采用 2-opt，Insert，Or-opt 与 Swap 四个搜索算符的结合方案，同时进行多组实验测试不同算符执行顺序对搜索性能与搜索速度的影响，根据实验结果以最佳算符执行顺序进行点集排序的局部搜索过程。

每一个算符的搜索过程为：在程序初始化时预先根据贪心规则基于距离计算好每一个点的近邻并根据距离的远近排序后存储，在后续的搜索过程中，每一个算符都将遍历所有当前访问点并针对每一个访问点遍历其近邻，原因是两点之间的相对距离与路径总长度的目标函数具有高度相关性，而路径的长短又在一定程度上影响最终获得的利润值。在两层遍历过程中将获取到的每个城市点及其近邻进行算符模拟执行，将模拟后的总利润值减去模拟前的总利润值作为模拟输出结果 Delta 值，Delta>0 表明该算符对于特定城市点的操作能够改进当前解，则真实执行该操作且在该算符最终遍历完成后检测这一轮搜索是否确实获得全局更优解，若是则将解保存并更新全局最优解以供后续比较。

2.3.1 2-opt 算符

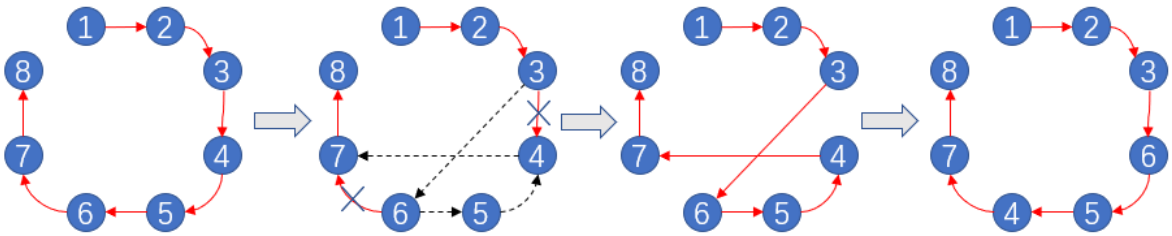

图 3：2-opt 算符示意图

2-opt 算符作为组合优化问题中一种常用的局部搜索算符，它的核心思路是断开当前解中的某两条边并重新连接构成一个新解。具体原理如上图所示：(1)点为修理工出发的起始点，沿箭头指向逐个访问城市点最终到达城市点(8)完成任务，这八个点序列按既定顺序组成当前解，假设此时搜索过程进行到城市点(3)及其邻域点(6)，记城市点(3)为 First_City，城市点(6)为 Second_City，依据当前解序列记(3)的下一点(4)为 First_City.Next，(6)的下一点(7)为 Second_City.Next，切断 First_City→First_City.Next 与 Second_City→Second_City.Next 两两之间的边，并重新连接 First_City→Second_City，First_City.Next→Second_City.Next，同时由于整个解结构应该是一个单向的序列且每个点被访问次数至多为一次，因此还需翻转 First_City.Next 到 Second_City 之间的点序列的箭头指向，也即修理工在该段的移动方

向，其余各部分序列保持不变，至此完成一次 2-opt 操作，执行一次完整 2-opt 搜索的时间复杂度为O(n)。

2.3.2 Insert 算符

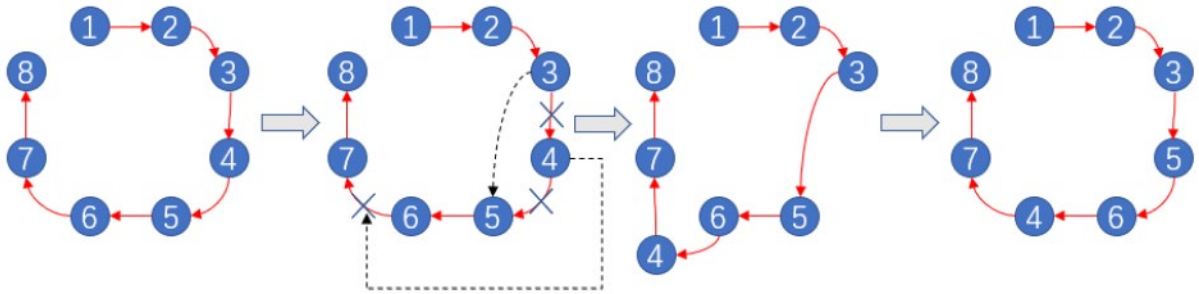

图 4：Insert 算符示意图

Insert 算符的核心思想是将某个城市点插入在特定的位置构成一个新解，可用于对较成熟的解结构进行微调以获得改进解。具体原理如上图所示：假设此时算符搜索过程进行到城市点(4)，将该点作为待插入城市 Insert_City，同时检索到城市点(4)的某一个近邻城市点(6)作为待插入位置 Insert_Position，则有待插入城市点的上一点 Insert_City.Pre 为城市点(3)，待插入城市点的下一点 Insert_City.Next 为城市点(5)，待插入位置的下一点 Insert_Position.Next 为城市点(7)，此时断开当前解的三条边 Insert_City.Pre→Insert_City，Insert_City→Insert_City.Next，Insert_Position→Insert_Position.Next，随后将待插入城市置于待插入位置点之后，同时将待插入城市的原前后点连接，即重新连接 Insert_City.Pre→ Insert_City.Next，Insert_Position→ Insert_City 和 Insert_City→ Insert_Position.Next 三条边构成新解，其余各部分序列保持不变，至此完成一次 Insert 操作。

上述 Insert 执行方式只涉及将城市点插入于位置点之后的操作，实际情况中由于待插入城市与待插入位置的前后关系也会造成结果的差异，因此在每一轮 Insert 算符搜索中会分别执行正向插入与反向插入，即分别搜索城市与插入点正向连接和反向连接是否带来更优的改进解，其各自的运算的时间复杂度均为O(n)。

2.3.3 Or-opt 算符

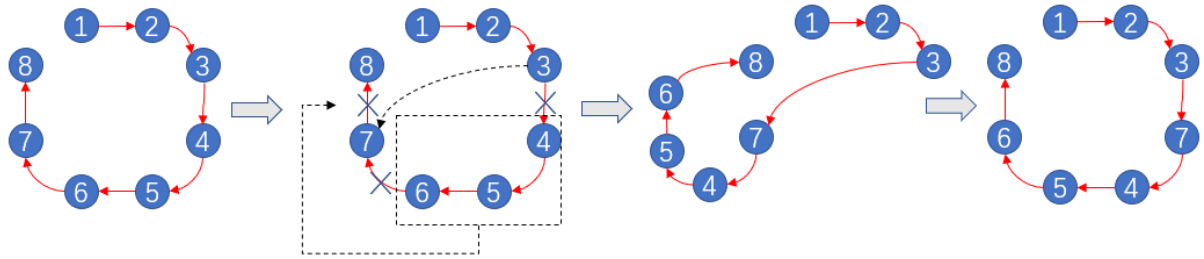

图 5：Or-opt 算符示意图

Or-opt 算符与 Insert 算符具有相似的思路和连接方式，但其主要作用是保持局部整体在搜索过程中的稳定性进而防止某一段较优的点序列在搜索过程中被打散，因此 Or-opt 算符与 Insert 的唯一区别是其插入的不只是一个点，而是 2 或 3 个连续的点序列。具体原理如上图所示：假设此时算符搜索过程进行到城市点(4)，将该点作为待插入序列的首端城市 Or_City，同时检索到城市点(4)的某一个近邻城市点(7)作为待插入位置 Or_Position，且此处假设经过计算需要保留的整体是三个连续点序列，则有待插入序列的首端城市点的上一点 Or_City.Pre 为城市点(3)，待插入序列的尾端城市点 Tail_City 为城市点(6)，尾端城市点的下一点 Tail_City.Next 为城市点(7)，待插入位置的下一点 Or_Position.Next 为城市点(8)，此时断开当前解得三条边 Or_City.Pre→Or_City，Tail_City→Tail_City.Next，Or_Position→Or_Position.Next，由 Or_City 到 Tail_City 之间的整段点序列就构成了算符即将操作的序列对象，随后将待插入序列置于待插入位置点之后，同时将该序列的原前后点连接，即重新连接 Or_City.Pre→ Tail_City.Next，Or_Position→ Or_City 和 Tail_City→ Or_Position.Next 三条边构成新解，其余各部分序列保持不变，至此完成一次 Or-opt 操作。

由于 Or-opt 算符操作的重点在于在保留一段点序列的完整性前提下移动找到更优的序列位置，因此不考虑插入序列与插入位置的前后相对位置问题，且经过实验表明在 Or-opt 算符中考虑插入的方向问题对于搜索结果只能提高约 0.3%，然而由于算符本身在搜索时需要对比选择操作序列的长度造成了运算量的提高，其运算时间复杂度虽然也为O(n)却是 Insert 算符的常数倍数，因此为了权衡兼顾算法的优度与速度，此算符中只考虑插入序列相对于位置正向插入的方式。

2.3.4 Swap 算符

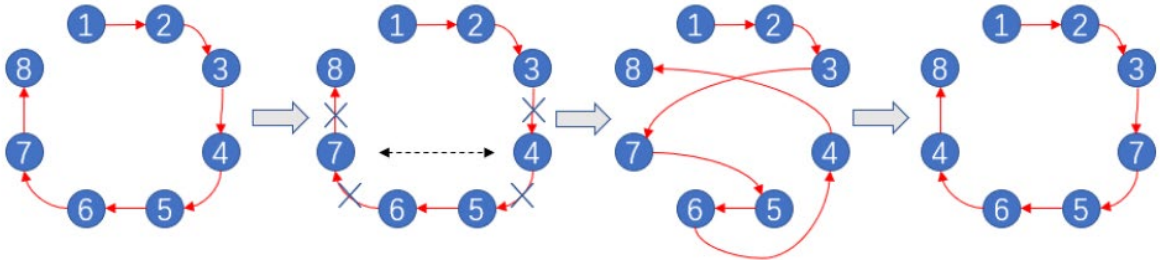

图 6：普通 Swap 算符示意图

Swap 算符的核心思想是切断当前解中的四条边并交换两个节点的位置，以通过对访问点顺序的微调来获得改进解。具体原理如上图所示：假设此时算符搜索过程进行到城市点(4)，将该点作为执行交换的 A 城市点，记为 A_City，同时检索到城市点(4)的某一个近邻为城市点(8)，则该点的前一个点(7)可作为执行交换的 B 城市点，记为 B_City，则有当前解序列中位于 A 城市上一个点的 A_City.Pre 为城市点(3)，位于 A 城市下一个点的 A_City.Next 为城市点(5)，位于 B 城市上一个点的 B_City.Pre 为城市点(6)，位于 B 城市下一个点的 B_City.Next 为城市点(8)，此时断开当前解的四条边 A_City.Pre→A_City，A_City→A_City.Next，B_City.Pre→B_City，B_City→B_City.Next，随后交换 A 与 B 的位置，即重新连接四条边 A_City.Pre→B_City，B_City→A_City.Next，B_City.Pre→A_City，B_City→A_City.Next，同时保持其他各部分序列不变，至此完成一次 Swap 操作。由于相邻城市点之间的交换与上述原理有所不同，只需要断开并重新连接三条边，因此在本算符中需要进行独立搜索，具体原理如下图所示：

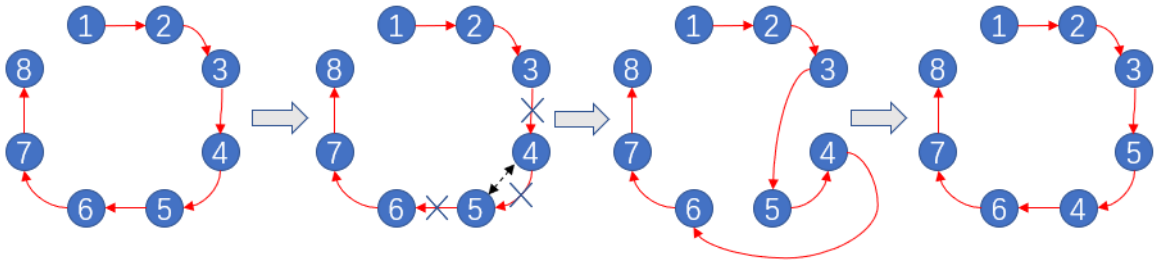

图 7：相邻 Swap 算符示意图

此图中交换的是相邻的城市点(4)和城市点(5)，因此依据普通 Swap 原理有 A_City 为城市点(4)，B_City 为城市点(5)，A_City.Pre 为城市点(3)以及 B_City.Next 为城市点(6)，但由于(4)和(5)相邻，因此在执行时只需断开三条边：A_City.Pre→A_City，A_City→B_City，B_City→B_City.Next，再重新连接三条边：A_City.Pre→B_City，B_City→A_City，A_City→B_City.Next 即可，保持其余部分点序列不变，则完成一次相邻节点的交换操作。

2.4 定向扰动

完成一定程度的点集排序搜索后，为了跳出当前局部最优，本文采用的是定向扰动方式。具体实现方法是：对于当前搜索过程中已经搜索过的解空间内所有获得的改进解进行解结构的学习和分析，将每一个解视为一个完全无向图，图中的所有访问城市点构成顶点集合 V，按照解序列连接两两访问点之间的所有边共同构成边集合 E，E 中的每一个元素e_i用其连接着的两个顶点i_l和i_r表示，每次定向扰动过程开始前统计边集合中每条边在已有的解容器中出现过的次数，并采用基于轮盘赌的思想对其进行选择，由于出现次数越少的边越有可能将搜索过程带入一个新的解空间中，因此带来更优解的可能性与潜能更大，具体的实现方式是统计结果中出现次数越少的边有更大的概率得到保留而出现次数越多的边越可能被切断并重新随机连接，这种基于轮盘赌的相对定向的扰动方式既能保留已有最佳解的一部分解结构用于遗传给子代解，又能够明显将搜索带入更大的解空间，实验表明这种方法在速度上明显优于随机扰动方式，又比单纯贪心算法能够获得更高的优度。

2.5 点集调整搜索过程

由于 TRPP 问题不要求修理工遍历访问所有给定城市点，因此需要进行访问点集的筛选与调整，本文采用的调整策略是按既定顺序进行 Add，Drop，与 InterChange 三种算符的搜索，此过程需要计算类似于点集排序搜索过程中的 Delta 值用于表征特定算符对于特定城市点操作后对当前解的改进情况，而在本过程中还需分别将三个算符搜索过程中所获得的所有 Delta 值进行存储以供后续点集调整扰动的概率选择使用。

2.5.1 Add 算符

Add 算符的核心思想是将未访问的城市点中可能为当前解带来改进的节点加入到合适的位置中，由于添加节点在带来更多利润的同时会带入更大的路程代价，因此本算符的关键是选择较优的位置添加节点避免新节点引入的利润与路程代价之差为负。具体实现方式是：假设当前解序列为$X = \{x_0, x_1, x_2 \dots x_{i-1}, x_i, x_{i+1} \dots x_m, x_{m+1} \dots x_{j-1}, x_j, x_{j+1} \dots x_n\}$，其中 m 为当前访问点的数量，n 为可访问点的总数，遍历(n-m)个当前不访问的城市点及其所有近邻，且必须保证该操作近邻点属于$\{x_0 \sim x_m\}$，否则视为非法操作，当检测到将x_j添加在x_i点之后能够带来解的改进时，执行该添加操作，即将当前解序列修改为$X = \{x_0, x_1, x_2 \dots x_{i-1}, x_i, x_j, x_{i+1} \dots x_m, x_{m+1} \dots x_{j-1}, x_{j+1} \dots x_n\}$，注意到此时访问点数量在原解基础上增加 1，因此每执行一次 Add 操作需要对 m 进行一次自增运算。

2.5.2 Drop 算符

Drop 算符的核心思想是将当前解中产生的路程代价大于为修理工带来的利润值的节点删去，即该点可能造成修理工在访问自身或其他节点时付出了一定的路程代价，但由于 TRPP 问题中节点利润值随时间推移而线性减少的约束，最终并没有在该点获得正向利润，这样的点即为无用节点，会为当前解带来多余的路程代价，因此将其从解中删去以带来改进。具体实现方式是：假设当前解序列表示为$X = \{x_0, x_1, x_2 \dots x_{i-1}, x_i, x_{i+1} \dots x_m, x_{m+1} \dots x_n\}$，其中 m、n 分别为当前访问点的数量及可访问点的总数，遍历 m 个当前访问的城市点，当检测到将x_i删除能够带来解的改进时，执行该删除操作，即将当前解序列修改为$X = \{x_0, x_1, x_2 \dots x_{i-1}, x_{i+1} \dots x_m, x_{m+1} \dots x_n\}$，注意到此时访问点数量在原解基础上减少 1，因此每执行一次 Drop 操作需要对 m 进行一次自减运算。

2.5.3 InterChange 算符

InterChange 算符的核心思想是 Add 与 Drop 算符的一种折中方法，当 Add 算符搜索到一个值得为其付出路程代价换取利润的节点时执行 Add，当 Drop 算符搜索到一个对当前解带来非正利润的节点时执行 Drop，但面对单一的删除点或添加点不能直接对当前解带来改进的情况时，InterChange 算符采用交换点的方式，将当前解中访问价值不大的点取下，同时在该点的原位置添加一个对解更有改进可能性的城市点。具体实现方式是：假设当前解序列表示为$X = \{x_0, x_1, x_2 \dots x_{i-1}, x_i, x_{i+1} \dots x_m, x_{m+1} \dots x_{j-1}, x_j, x_{j+1} \dots x_n\}$，其中 m、n 分别为当前访问点的数量及可访问点的总数，遍历(n-m)个当前不访问的城市点及其所有近邻，且必须保证该操作近邻点属于$\{x_0 \sim x_m\}$，否则视为非法操作，当检测到将x_j添加到其近邻点x_i之后同时删除x_i的下一个点x_{i+1}，即交换x_j与x_{i+1}能够带来解的改进时，将当前解序列修改为$X = \{x_0, x_1, x_2 \dots x_{i-1}, x_i, x_j, x_{i+2} \dots x_m, x_{m+1} \dots x_{j-1}, x_{i+1}, x_{j+1} \dots x_n\}$，此时访问点数量在原解基础上保持不变。

2.6 基于轮盘赌的点集调整扰动

在点集调整搜索过程中，如果有发现相对于当前解有改进的操作，则算法不需要进行此环节，直接进入下一轮点集排序局部搜索，但如果三个点集调整搜索算符均未找到改进解，则将此过程中计算出并存储的所有 Delta 值进行排序，由于扰动操作的目的是带来更多的解结构，扩充当前的解空间，因此本文采用基于轮盘赌的机制按概率接受较差解，具体实现方式是：对于排序后的 Delta 值序列，每一个 Delta 值对应三种点集调整算符中的一

种及其操作对象，每一个 Delta 值及其操作对象被选中执行的概率与 Delta 值呈线性关系，即 Delta 值越大对应的操作被选中执行的概率越大，若被选中的操作为 Add 或 InterChange 时，同样依据轮盘赌机制按相对近邻距离的大小进行概率选择插入位置或交换城市点，采用这种扰动方式在一定程度上能够有更大可能性获得改进解且由于点集调整后可能需要一定的点集排序局部搜索才能获得更优解，这样的扰动机制适当接受较差解也能带来更大的搜索灵活性。

2.7 本章小结

本章主要介绍了整体算法流程，从采用贪心随机算法构造初始解到 2-opt、Insert、Or-opt 和 Swap 四种局部搜索算符进行点集排序搜索；搜索到达一定程度后通过点集排序扰动跳出局部最优；再结合 TRPP 问题中不要求遍历所有访问点且每个城市点至多只能被访问一次的约束条件，通过 Add、Drop 和 InterChange 三个算符进行点集调整局部搜索；最后采用基于轮盘赌思想的点集扰动，按概率接受一定程度上的较差解，循环迭代这五个步骤直到搜索满足程序终止条件时跳出循环并将搜索过程中的最佳结果输出保存。

3. 强化学习与蒙特卡洛搜索树算符

3.1 蒙特卡洛树搜索算符

3.1.1 k-opt 搜索过程

在点集排序搜索过程中，本算法除了使用上述四种常见的局部搜索算符以外，根据已有的 2-opt，3-opt，4-opt 等经典搜索算符提出一种新的基于蒙特卡洛搜索框架的 k-opt 算符。常见的 q-opt 算符(包括 2-opt，3-opt，4-opt 等)的核心思路是断开当前解中的 q 条边并通过策略选择后重新连接 q 条新的边构成一个新解以寻求改进，且重连接时可选的边的条数随着 q 的增大以阶乘速度增长。本文在此基础上提出一种多状态的搜索方法，每一个状态对应一个解，表示为$X = \{x_0, x_1, x_2 \dots x_{i-1}, x_i, x_{i+1}, \dots x_m, x_{m+1} \dots x_{j-1}, x_j, x_{j+1} \dots x_n\}$，n 为所有可访问点的数量，每一个改进操作都能够将当前状态X转变为X^*，每一个可行解都包含 m 条边，本文提出一种不定 q 的搜索算符称为 k-opt($2 \le k \le m$)操作，该操作通过删除当前解的 k 条边随后重新连接 k 条新边获得一个新的解序列。每一个操作可视为 2k 个子决策过程(即删减哪 k 条边与添加哪 k 条边)，但同时进行 2k 个决策不仅会造成计算复杂度的升高且可能导致操作后无法获得可行解，对此本文采用逐步决策的方法，且只需完成 k 个决策，具体操作方法如下图所示：

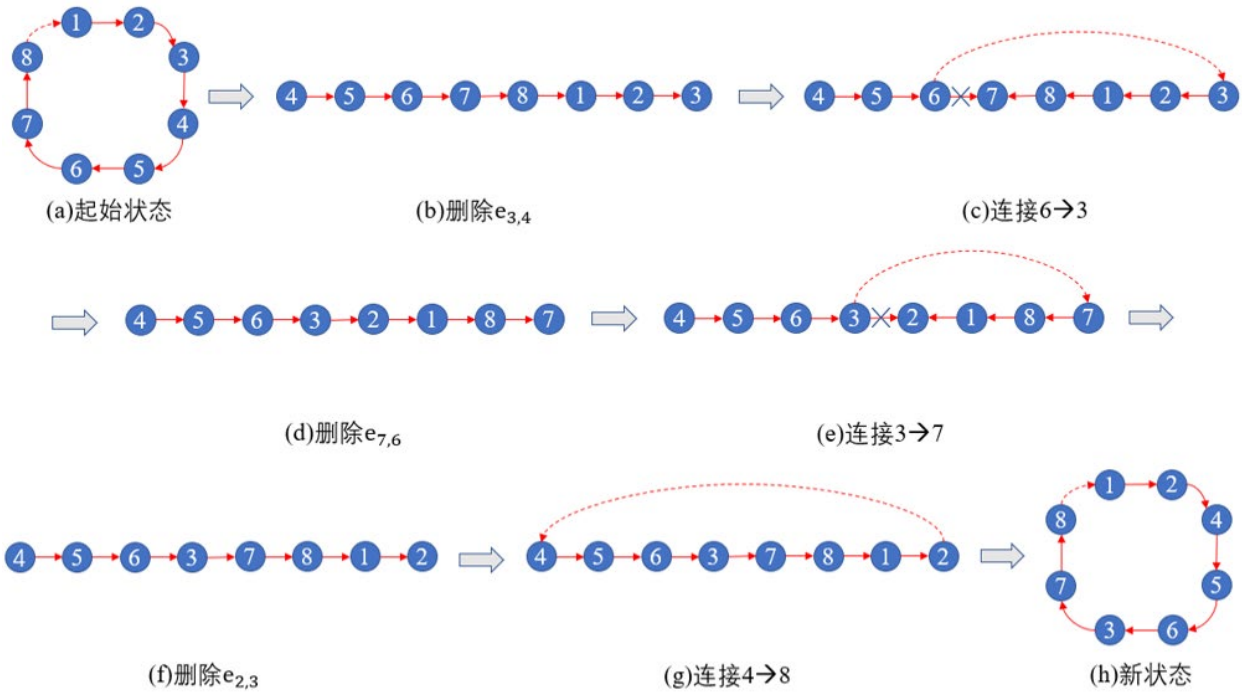

图 8：k-opt 算符示意图

假设当前解如步骤(a)所示，原 TRPP 问题不要求修理工完成任务后返回仓库，此处为了算

符描述的方便将城市点(8)与(1)(即仓库点)相连作为 k-opt 算符搜索前的起始状态，首先算法进行第一次决策确定第一条删除的边，此处假设删除的是连接城市点(3)与(4)的边，表示为 $e_{3,4}$，删除 $e_{3,4}$ 后将一个环状解序列转化为链状解序列如步骤(b)所示，我们将城市点(4)称为本轮搜索的 Begin City，其在原解序列中的上一个点(3)称为 Current City，随后进行第二次决策，在 Current City 的近邻中寻找一个与之相连后可能对解产生改进的节点，此处假设为城市点(6)，如果此时将(6)与(3)相连但(6)原本又连接着城市点(7)和城市点(5)，注意到每一个合法的可行解结构中除了仓库点与末尾点以外的所有点都有且只有两条边与之相连，则此时直接将(6)与(3)相连会导致出现一个不合法的解结构，因此连接 6→3 之前需要将(6)与(7)或(6)与(5)之间的边断开，其中(5)作为(6)的上游点而(7)是(6)的下游点，如果断开 $e_{5,6}$ 则城市点(4)和(5)将成为两个孤立节点，显然操作不合法，因此必须断开 $e_{6,7}$ 才能保持一个完整的链状解结构，我们将城市点(6)称为 Next City to Connect，城市点(7)称为 Next City to Disconnect，且 Next City to Disconnect 一定是 Next City to Connect 为了获得可行解而断开的一个下游点，即 Next City to Disconnect=Next City to Connect.Next，随后的每一步连接新边的操作都会附带一个自动删除对应边的操作以保证最终能够形成一个可行解，由于搜索开始前将末尾点和仓库点相连形成了一个单向闭合的环状解，因此还需要翻转 Next City to Disconnect 到 Current City 之间的节点连接方向以保证整个解中节点访问方向的一致性，至此完成步骤(c)获得形如步骤(d)所示的一个链状解，注意到此时只有 Begin City(4)和 Next City to Disconnect(7)两个节点都仅一条边与之相连，所以应该更新 Current City 的值，将 Next City to Disconnect 赋值给 Current City 后进行下一次决策，假设决策机制又发现一个 Current City 的近邻中可能带来改进解的城市点为(3)，类似的，将 Next City to Connect 更新为(3)且 Next City to Disconnect 更新为(3)的下游点(2)，随后连接 $e_{3,7}$ 并自动断开 $e_{3,2}$ 同时翻转 Next City to Disconnect 到 Current City(即(2)到(7))之间的节点连接方向，完成步骤(e)后同样更新 Current City 并进入步骤(f)的第四次决策过程，假设此时决策机制在 Current City 的近邻中搜索不到可能带来改进解的城市点或搜索过程返回的 Next City to Connect 恰好就是 Begin City 即城市点(4)，那么创建新边 $e_{2,4}$，将当前的 Current City 与 Begin City 相连重新形成单向闭环解，进入步骤(h)跳转到新状态，至此本轮 k-opt 搜索结束，由于整个过程中共切断三条边又重新连接三条边，因此 k=3。

从数学表达上每一轮算符搜索中执行的操作集可表示为$a = (a_1, b_1, \dots, a_k, b_k, a_{k+1})$，其中 k 为一个没有预先设定的变量，且第一个操作的城市点与最后一个操作城市点必须相同，即要求$a_1 = a_{k+1}$，每一组操作集对应一轮 k-opt 算符搜索，搜索过程中将会删除 k 条边即

$e_{a_i,b_i}(1 \le i \le k)$，又添加 k 条新边即$e_{b_i,a_{i+1}}(1 \le i \le k)$以获得一个新解进入新状态，注意到并不是操作集中的所有元素都由决策机制产生，一旦a_i确定则b_i自动由a_i的下游点构成，因此一轮 k-opt 搜索过程只需要进行 k 次子决策，这种逐点决策的机制不仅带来更少的决策次数，且能够保证搜索完成后获得的新解是一个合法解。

3.1.2 蒙特卡洛树搜索算符完整流程

蒙特卡洛树搜索算符的操作方法如上所述，虽然采用 k-opt 的方法能够大幅减少决策过程，但对于较大规模的算例，如何兼顾速度与优度更合理地在近邻中做出决策，选择更合适的下一个操作点，对搜索结果的影响依旧很大，因此本文提出一种基于强化学习的方法，通过对已有解空间中的所有解的结构进行分析学习，逐渐筛选出更有可能构成最优解序列的边供决策时使用，所采用的学习框架是蒙特卡洛树搜索框架，框架包含初始化、模拟、选择与反向传播四部分，具体流程如下图所示：

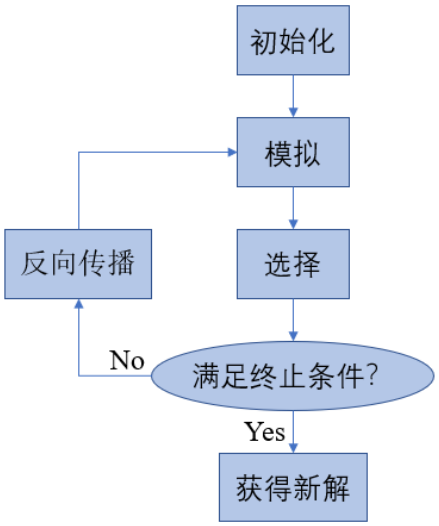

图 9：蒙特卡洛搜索树算符流程图

3.1.2.1 初始化

定义两个 n×n 的对称矩阵，n 为所有可访问点的数量，其中一个为权重矩阵 W，其元素$W_{i,j}$ (初始化为 1)表示将 j 点连接在 i 点之后的概率大小；另一个为访问矩阵 Q，其元素$Q_{i,j}$ (初始化为 0)表示边$e_{i,j}$在整个模拟过程种被选中过的次数。另外有一个变量 M(初始化为 0)用于记录模拟次数，初始化过程只需在整个蒙特卡洛树搜索过程开始前执行一次即可。

3.1.2.2 模拟

给定当前解 X，本文通过模拟过程以 3.1.1 节所述方式生成一系列操作，一个操作集可表示为$a = (a_1, b_1, \dots, a_k, b_k, a_{k+1})$，包含一系列子决策$a_i$，其中 k 为变量，1≤i≤k 且$a_{k+1} =$

a_1，而当a_i确定时每一个b_i也随之确定，随后搜索与b_i相连可能获得改进解的下一个操作城市点，对于b_i与城市点j相连的每条边$e_{b_i,j}(b_i \neq j)$都使用以下公式评估其被选中加入新解的理论潜在可能$Z_{b_i,j}$ ($Z_{b_i,j}$越大则$e_{b_i,j}$被选中的机会越大)：

$$Z_{b_i,j} = \frac{W_{b_i,j}}{\omega_{b_i,j}} + \alpha\sqrt{\frac{\ln(M+1)}{Q_{b_i,j}+1}}$$

其中$\omega_{b_i,j} = \frac{\sum_{j\neq b_i} W_{b_i,j}}{\sum_{j\neq b_i} 1}$是对所有可能与$b_i$点相连的城市点的$W_{b_i,j}$求均值，公式中左半部分$\frac{W_{b_i,j}}{\omega_{b_i,j}}$用于突出附带高权重值的边的重要性(用于保证决策搜索的集中性即搜索深度)，右半部分$\sqrt{\frac{\ln(M+1)}{Q_{b_i,j}+1}}$则更倾向于对低权重的边的选择(用于保证决策搜索的分散性即搜索广度)，而参数α的作用是调整达到搜索深度与广度的平衡，注意到公式中“+1”部分是为了避免分子为负以及分母为零的情况出现。

在进行决策过程中，首先根据访问矩阵 Q 按概率随机选择一条$Q_{i,j}$值较小的边，表示该边在既有的解中出现次数较少，对最优解的重要性相对较低，将该边切断即可获得与边相连的上游点a_1与下游点b_1。在尚未获得完整的操作集时，a_i与b_i一旦确定，算法采用以下两个规则确定a_{i+1}：（1）如果此时关闭搜索过程，将b_i连接回到a_1能够获得改进解，或者搜索次数已达设定的搜索深度上限，则令$a_{i+1} = a_1$；（2）否则在b_i的近邻点集合中进行搜索，将其中$Z_{b_i,l} \geq 1$的所有点作为候选点集共同构成集合 N，注意 N 中不包括a_1及其他已经与b_i相连的城市点，随后对 N 中的所有点采用如下公式进行概率赋值：

$$p_j = \frac{Z_{b_i,j}}{\sum_{l\in N} Z_{b_i,l}}$$

p_j表示城市点b_i的近邻中城市点 j 被选中作为a_{i+1}的概率，一旦$a_{i+1} = a_1$就立即终止搜索过程返回完整的操作集。算法在模拟过程中生成多组操作集共同组成操作采样池直到发现一个能够对已有最优解做出改进的动作或操作集数量达到设定上限时才停止模拟。

3.1.2.3 选择

如果在模拟过程中发现一个改进操作，则该操作集立刻被选中应用使得当前状态 X 提升到改进状态 X^*，否则如果操作采样池中的操作集数量达到上限时依旧没有在模拟过程中发现改进操作，表示当前的搜索区域较难发现新的改进解，此时算法调用 2.4 节定向扰动机制使当前状态跳转到一个新状态 X^{new} 并重新开始新一轮局部搜索。

3.1.2.4 反向传播

模拟次数 M、权重矩阵 W、访问矩阵 Q 的元素值均在反向传播环节进行更新。其中，每当操作采样池中增加一组改进操作时，M 增加 1；当b_i与a_{i+1}相连的边在改进操作中出现一次时，访问矩阵 Q 中的元素值$Q_{b_i,a_{i+1}}$增加 1；当一个解从状态 X 经过应用操作$a=(a_1,b_1,a_2,b_2,\dots,a_k,b_k,a_{k+1})$(1≤i≤k)后跳转到新状态 X*时，其中的每一条边$e_{b_i,a_{i+1}}$对应的每一个$W_{b_i,a_{i+1}}$值按照如下公式进行值的更新：

$$W_{b_i,a_{i+1}} \leftarrow W_{b_i,a_{i+1}} + \beta[\exp\left(\frac{L(X^*)-L(X)}{L(X)}\right)-1]$$

其中β是用于控制$W_{b_i,a_{i+1}}$增长速率的变量，$L(X)$表示状态 X 下的目标函数值。注意到只有当解获得改进时才进行$W_{b_i,a_{i+1}}$的更新，这样的反向传播机制能够提高较优边被选中的概率，使得模拟采样环节随着循环迭代次数的增加愈加有针对性。

3.2 强化学习思想在本算法中的体现

本文工作大量融合了强化学习思想，即由算法根据整个搜索过程中所有的解结构进行分析与记忆，反向作用于搜索过程包括局部搜索与定向扰动等环节。具体的体现包括三部分：（1）蒙特卡洛树搜索过程中的 k-opt 算符需要进行a_1的选择，第一次迭代时由于访问矩阵 Q 中的元素尚未获得任何更新，因此采用随机选择方式；但之后的每一次 k-opt 算符搜索都会根据访问矩阵的元素信息进行更加有针对性的选择，由于被选作a_1的城市点会作为上游点切断与之相连的边，且我们认为大致上某条边越有可能参与构成最优解时该边在已有解集中的出现次数应该越大，因此需要根据矩阵 Q 按概率随机方式选出$Q_{i,j}$相对较小的边；（2）同样是蒙特卡洛搜索过程中的模拟环节，决策机制需要选择更有可能形成改进解的a_{i+1}点与b_i相连，此过程参考的是权重矩阵的$W_{b_i,a_{i+1}}$值，权重越大表示该边构成改进解的可能性越大；（3）在点集排序的扰动环节，为了扰动过程更加具有针对性，算法也通过访问矩阵的$Q_{i,j}$值进行扰动指导，由于扰动的目的是扩大搜索广度，因此需要更加倾向于切断在已有解集中出现次数较多的边而连接此前出现的较少的新边，这样的定向操作能够迅速将当前解拉出局部最优跳转到新的搜索区域。

本文的算法框架在上述三个部分中运用已有知识进行操作的导向，而在所有局部搜索环节，一旦获得改进解就依据 3.1.2.4 节中的反向传播机制进行学习与记忆，形成了前后相

互辅助的完整的强化学习反馈机制。

3.3 本章小结

本章主要介绍了算法框架中进行局部搜索的重要算符之一：蒙特卡洛搜索树算符，该算符的 k-opt 搜索机制是对已有 q-opt 搜索算符原理的总结与改进，通过逐点决策方式大幅提高了搜索速度同时保证搜索结果的可行性；同时完整介绍了整个蒙特卡洛搜索树框架在本算法中的应用流程；最后总结了强化学习思想在本工作中的三处体现。

4. 实验数据及对比结论

4.1 各算符局部搜索性能与速度实验

本文通过如下实验测试了点集排序搜索过程与点集调整搜索过程中各个算符的搜索能力与搜索速度，同时为了体现算符间的优度与速度差距并兼顾实验的普遍性与实验时间，具体方案如下：采用算例规模为 n=100 的算例测试集，固定随机种子使算法通过贪心随机过程生成的初始解固定，同时为了体现 Add 与 InterChange 算符的搜索能力，实验时提前随机指定了一个初始访问城市点的数量 k，k 值是(0.7n~0.9n)之间随机生成的整数，基于此对给定的 20 个算例进行测试，每个算例完成 10 个周期迭代的局部搜索过程，各组实验采用不同的局部搜索算符进行搜索，统计后对 20 个算例的计算结果与用时进行累加并求均值。实验数据如下表所示：

算符名称	平均结果	平均改进幅度(%)	平均用时(s)
2-opt	208459.85	138.45	0.0197
Or-opt	207851.25	137.75	0.0166
Swap	139823.05	59.94	0.0006
Insert	208909.40	138.96	0.0145
k-opt	210447.05	140.72	0.5513
Drop	140414.15	60.61	0.0009
Add	114971.80	31.51	0.0024
InterChange	169103.35	93.43	0.0071

表 1：各算符搜索性能与速度表

由实验数据可知每一个算符单独搜索时都能够将贪心随机构造的初始解改进 30%以上，部分搜索能力较强的排序搜索算符具有接近 150%的改进能力；且大部分算符对于 n=100 规模的算例完成一次搜索所需的时间都低于$\frac{1}{50}$秒，搜索能力最强的 k-opt 算符也只需要约$\frac{1}{2}$秒的时间，由此可见每一个局部搜索算符都具有较强且较为稳定的搜索能力，各个算符的搜索速度也较为可观，都是能够进行高效搜索的邻域结构，并且由于本实验是单个算符的迭代搜索测试，而实际算法框架是多个算符相互辅助依次进行局部搜索，因此实际运行时的每个算符的搜索时间将更短于本实验的实验结果。

4.2 点集排序搜索与点集调整搜索中的算符排序实验

由于局部搜索过程中算符的搜索顺序对搜索结果与搜索速度都具有较大影响，因此本实验通过测试几种主要的算符排列顺序以找到最合适的局部搜索顺序。

4.2.1 点集排序搜索中的算符排序实验

在点集排序搜索过程中本算法采用了 2-opt、Or-opt、Swap、Insert 与 k-opt 五种搜索算符，并排列出以下六种搜索顺序的组合方式：

OCM-1	2-opt → Or-opt → Swap → Insert → k-opt
OCM-2	Swap → 2-opt → Insert → Or-opt → k-opt
OCM-3	Insert → Swap → Or-opt → 2-opt → k-opt
OCM-4	2-opt → k-opt → Or-opt → Swap → Insert
OCM-5	Or-opt → 2-opt → Swap → k-opt → Insert
OCM-6	2-opt → Or-opt → k-opt → Swap → Insert

表 2：点集排序搜索组合实验编号对照表

表格中 OCM 表示算符组合方式(Operation Combination Mode)，实验同样采用算例规模为 n=100 的算例测试集进行测试，并且通过固定随机种子来固定搜索的初始解进而固定起始搜索区域，基于 n=100 规模的 20 个算例进行测试，除了上述 6 种算符组合方式，另外进行了一个随机组合的对照组，即通过 5 个算符的随机排序进行局部搜索，且两两迭代周期之间的排序方式也不一定相同，该对照组用 OCM-R 表示，每个算例完成 10 个周期迭代的局部搜索过程，各组实验采用不同的算符组合方式进行搜索，统计后对 20 个算例的计算结果与用时进行累加并求均值。实验数据如下表所示：

算符组合	平均结果	平均改进幅度(%)	平均用时(s)
OCM-1	211035.35	141.39	0.166
OCM-2	211140.30	141.51	0.152
OCM-3	210816.35	141.14	0.182
OCM-4	211096.75	141.46	0.305
OCM-5	211131.60	141.50	0.198
OCM-6	211063.40	141.42	0.252
OCM-R	210992.15	141.34	0.291

表 3：点集排序搜索组合实验结果

由实验数据可知，第二种算符组合方式 OCM-2，即(Swap → 2-opt → Insert → Or-opt → k-opt)的搜索顺序能够获得最大的改进幅度，具有最大的搜索能力且耗时最短，因此该组合方式具有最大的搜索效率，下节的完整算法实验将采用该种组合方式。

4.2.2 点集调整搜索中的算符排序实验

在点集调整搜索过程中本算法采用了 Add、Drop 和 InterChange 三种调整点集算符，共

有以下六种排列组合方式：

ASC-1	Add → InterChange → Drop
ASC-2	Add → Drop → InterChange
ASC-3	Drop → Add → InterChange
ASC-4	Drop → InterChange → Add
ASC-5	InterChange → Add → Drop
ASC-6	InterChange → Drop → Add

表 4：点集调整搜索组合实验编号对照表

表格中 ASC 表示调整点集组合(Adjust Set Combination)，具体实验方式类似 4.2.1 节所述，同样有一组对照实验采用算符随机组合方式，该组用 ASC-R 表示。实验数据如下表所示：

算符组合	平均结果	平均改进幅度(%)	平均用时(s)
ASC-1	180673.30	106.66	0.153
ASC-2	184520.10	111.06	0.183
ASC-3	182421.35	108.66	0.132
ASC-4	185130.10	111.76	0.174
ASC-5	181972.50	108.15	0.177
ASC-6	182816.15	109.11	0.152
ASC-R	183884.95	110.34	0.149

表 5：点集调整搜索组合实验结果

由实验数据可知，第四种算符组合方式 ASC-4，即(Drop → InterChange → Add)的搜索顺序能够获得最佳搜索效果，虽然 ASC-4 耗时相对较长，但为了保证整个调整点集环节的搜索能力，下节的完整算法实验将采用该种组合方式。

4.3 基于给定算例的完整搜索实验及现有结果对比 [1]

由于 TRPP 问题具有不同问题规模的多种给定算例测试集供算法性能的测试使用，且现有研究也已基于该测试集对其算法做出测试实验，因此本文也在相同测试集上进行算法性能的测试并与结果对比。其中，引文[1]采用的是禁忌搜索算法，结果对比表格中用 TS(Tabu Search)表示，引文[18]采用贪心自适应随机搜索过程与迭代局部搜索相结合的方式，用 GRS (Greedy Randomized Search)表示，引文[19]采用混合进化搜索算法，用 HES(Hybrid Evolutionary Search)表示，而本文提出的是基于强化学习的启发式算法框架，

[1] 实验对比的数据说明：由于相关的 TRPP 论文都没有给出程序源码且受时间限制没有实现其原算法重新再同等平台上进行公平的测试实验，因此本工作对比的算法执行时间与结果均为原论文所给出的数据。

用 RLS(Reinforcement Learning-based Search)表示，另外由于引文[1]已给出较小规模(n≤20)算例的精确最优解，因此也用于结果对比，用 Exact 表示；表格中的 Gap 项表示本工作对已知最优解的改进比例，其计算公式为$\mathrm{Gap}(\%) = 100 \times \frac{(Best-RLS)}{RLS}$，其中 Best 为目前已知所有 TRPP 问题相关文献给出的最优解，完整实验数据及软硬件平台数据如下各表所示：

算法	编程语言	处理器型号	CPU 频率	内存大小
TS	C++	Intel(R) Core(TM) 2 Duo	3.00 GHz	4.00 GB
GRS	Matlab	Intel Core Duo 2 T 7500	2.20 GHz	未提供
HES	C++	Intel Xeon(R) CPU E5-2695 v4	2.10 GHz	2.00 GB
RLS	C++	Intel(R) Core(TM) i5-6200U	2.30 GHZ	4.00 GB

表 6：各算法所运行的软硬件平台数据表

	n=10					n=20				
	Exact	TS	GRS	HES	RLS	Exact	TS	GRS	HES	RLS
1	2520	2520	2520	2520	2520	8772	8772	8772	8772	8772
2	1770	1770	1770	1770	1770	10174	10174	10174	10174	10174
3	1737	1737	1737	1737	1737	7917	7917	7917	7917	7917
4	2247	2247	2247	2247	2247	7967	7967	7967	7967	7967
5	2396	2396	2396	2396	2396	7985	7985	7985	7985	7985
6	1872	1872	1872	1872	1872	7500	7500	7500	7500	7500
7	1360	1360	1360	1360	1360	9439	9439	9439	9439	9439
8	1696	1696	1696	1696	1696	7999	7999	7999	7999	7999
9	1465	1465	1465	1465	1465	6952	6952	6952	6952	6952
10	1014	1014	1014	1014	1014	8582	8582	8582	8582	8582
11	1355	1355	1355	1355	1355	7257	7257	7257	7257	7257
12	1817	1817	1817	1817	1817	6857	6857	6857	6857	6857
13	1585	1585	1585	1585	1585	7043	7043	7043	7043	7043
14	2122	2122	2122	2122	2122	6964	6964	6964	6964	6964
15	1747	1747	1747	1747	1747	6270	6270	6270	6270	6270
16	1635	1635	1635	1635	1635	8143	8143	8143	8143	8143

17	2025	2025	2025	2025	2025	10226	10226	10226	10226	10226
18	1783	1783	1783	1783	1783	7625	7625	7625	7625	7625
19	1797	1797	1797	1797	1797	7982	7982	7982	7982	7982
20	1771	1771	1771	1771	1771	7662	7662	7662	7662	7662

表 7：城市点数量 n=10 & n=20 算例实验结果

	TS		**GRS**		**HES**		**RLS**		
	f_{best}	t(s)	f_{best}	t(s)	f_{best}	t(s)	f_{best}	t(s)	Gap(%)
1	50921	14.1	50921	11.5	50921	11.5	50921	9.5	0.0
2	52594	12.6	52594	8.4	52594	8.4	52594	6.9	0.0
3	52144	12.5	52144	9.4	52144	9.4	52144	6.7	0.0
4	45465	12.4	45465	7.3	45465	7.3	45465	5.6	0.0
5	45489	11.2	45489	8.4	45489	8.4	45489	6.2	0.0
6	55630	12.8	55630	7.3	55630	7.3	55630	5.1	0.0
7	44300	9.2	44302	9.4	44302	9.4	44302	8.4	0.0
8	55753	12.3	55801	10.5	55801	10.5	55801	9.4	0.0
9	44964	11.3	44964	10.5	44964	10.5	44964	8.5	0.0
10	47071	9.8	47071	9.4	47071	9.4	47071	7.3	0.0
11	51912	10.3	51912	9.4	51912	9.4	51912	7.6	0.0
12	53567	14.5	53567	8.4	53567	8.4	53567	5.9	0.0
13	46830	12.6	46830	9.4	46830	9.4	46830	8.3	0.0
14	52665	53.1	52665	8.4	52665	8.4	52665	6.8	0.0
15	58856	14	58856	10.5	58856	10.5	58856	9.0	0.0
16	49754	12.1	49754	12.6	49754	12.6	49754	10.8	0.0
17	42525	9.6	42525	8.4	42525	8.4	42525	7.5	0.0
18	40536	11.9	40536	9.4	40536	9.4	40536	7.0	0.0
19	55346	15	55346	10.5	55346	10.5	55346	8.5	0.0
20	61286	13.4	61286	9.4	61286	9.4	61286	8.2	0.0

表 8：城市点数量 n=50 算例实验结果

	TS	**GRS**	**HES**	**RLS**

	f_{best}	t(s)	f_{best}	t(s)	f_{best}	t(s)	f_{best}	t(s)	Gap(%)
1	209952	119.7	209952	92.2	209952	92.2	209952	50.67	0.0
2	196318	101.3	196318	135.1	196318	135.2	196318	50.44	0.0
3	211937	126.7	211937	107.9	211937	108	211937	50.01	0.0
4	217685	112.1	217685	89	217685	89.1	217685	50.13	0.0
5	215119	169.2	215119	131	215119	131	215119	50.96	0.0
6	228687	144	228687	102.7	228687	102.8	228687	50.05	0.0
7	200060	347.3	200064	128.9	200064	129	200064	50.69	0.0
8	205760	117.8	205760	119.4	205760	119.5	205760	50.43	0.0
9	226240	93.7	226240	112.1	226240	112.1	226240	50.33	0.0
10	218202	162.5	218202	88	218202	88	218202	50.08	0.0
11	212503	126.8	212503	125.7	212503	125.8	212503	50.78	0.0
12	222249	148	222249	98.5	222249	98.6	222249	50.71	0.0
13	206878	145.2	206957	89	206957	89.1	206957	50.92	0.0
14	215690	126.8	215690	85.9	215690	86	215690	50.83	0.0
15	213758	157.7	213811	89	214041	89.1	214041	50.01	0.0
16	214036	152	214036	89	214036	89	214036	50.71	0.0
17	223636	136.6	223636	118.4	223636	118.5	**223642**	50.89	**-0.0027**
18	192849	122.5	192849	97.4	192849	97.4	192849	50.31	0.0
19	206755	174.9	206755	128.9	206755	128.9	206755	50.17	0.0
20	198842	213.4	198908	104.8	198908	104.8	198908	50.20	0.0

表 9：城市点数量 n=100 算例实验结果

	TS		**GRS**		**HES**		**RLS**		
	f_{best}	t(s)	f_{best}	t(s)	f_{best}	t(s)	f_{best}	t(s)	Gap(%)
1	877523	1454.2	877530	1097.9	877610	1098.9	877610	650.86	0.0
2	901419	1434.4	901559	1047.6	901898	1048.2	**901928**	650.84	**-0.0033**
3	888082	1431.7	888393	1143	888393	1144.1	888393	650.66	0.0
4	873530	1440.1	873840	1065.4	873910	1068.8	873910	650.98	0.0
5	849205	1432.7	849231	1110.5	849358	1111.7	849358	650.35	0.0

6	816452	1434.7	816910	1108.4	816923	1109.0	**816928**	650.55	**-0.0006**
7	783367	1448.7	783594	1082.2	784120	1082.4	784120	650.88	0.0
8	837908	1432.7	837938	1114.7	838023	1115.4	**838093**	650.64	**-0.0084**
9	891035	1439.7	891071	1043.4	891203	1046.0	891203	650.65	0.0
10	846662	1430.2	847235	1062.3	847303	1065.4	**847308**	650.45	**-0.0006**
11	803605	1430	803671	1082.2	804851	1082.4	804851	650.85	0.0
12	808313	1428.8	808686	1108.4	808966	1108.6	808966	650.84	0.0
13	861375	1433.5	861604	1072.8	861729	1075.5	**861749**	650.60	**-0.0023**
14	849942	1429.4	850118	1102.1	850507	1106.5	**850621**	650.62	**-0.0134**
15	847181	1455.8	847755	1088.5	848006	1092.0	848006	650.02	0.0
16	853315	1445	854075	1121	854075	1122.2	854075	650.52	0.0
17	860638	1429.4	861175	1040.3	861747	1042.4	861747	650.63	0.0
18	841922	1433	842283	1138.8	842953	1142.2	842953	650.25	0.0
19	822308	1432.6	822397	1063.3	822855	1065.6	**822881**	650.28	**-0.0032**
20	902948	1428.1	903772	1041.3	904326	1042.0	**904338**	650.21	**-0.0013**

表 10：城市点数量 n=200 算例实验结果

	TS		**GRS**		**HES**		**RLS**		
	f_{best}	t(s)	f_{best}	t(s)	f_{best}	t(s)	f_{best}	t(s)	Gap(%)
1	6596427	2857.1	6610423	1905.6	6630382	1911.1	**6635698**	1809.9	**-0.0801**
2	6382793	2856	6409937	1995.7	6418393	2001.5	**6419428**	1805.7	**-0.0161**
3	6331860	2856	6336705	1981	6383933	1985.6	**6385273**	1800.0	**-0.0210**
4	6589154	2856.7	6620316	1992.6	6628111	1996.4	**6635526**	1809.4	**-0.1117**
5	6751588	2857.2	6778323	2075.3	6780912	2079.7	**6781445**	1805.6	**-0.0079**
6	6104076	2857.2	6123548	1908.8	6128506	1911.0	**6132826**	1802.2	**-0.0704**
7	6694671	2856.9	6778295	2024	6778911	2026.1	**6781455**	1803.4	**-0.0375**
8	6506056	2857	6518013	1870	6534078	1871.2	**6537932**	1804.4	**-0.0589**
9	6530832	2857.1	6560088	1931.8	6581036	1934.6	**6590915**	1809.5	**-0.1499**
10	6686395	2856.6	6713796	1917.1	6714813	1922.9	**6715589**	1801.7	**-0.0116**
11	6689340	2856.1	6719607	2082.7	6739761	2087.1	**6745809**	1800.1	**-0.0897**

12	6466482	2855.9	6505037	2005.1	6507506	2011.1	**6539658**	1807.5	**-0.4916**
13	6341393	2856.5	6385898	1926.6	6386100	1931.1	**6388217**	1804.4	**-0.0331**
14	6718962	2856.4	6720088	1907.7	6758997	1910.5	**6766121**	1805.0	**-0.1053**
15	6719534	2856.6	6731903	1853.2	6759142	1856.2	**6766902**	1806.7	**-0.1147**
16	6548694	2855.9	6606948	1954.9	6610001	1955.5	**6633873**	1800.7	**-0.3599**
17	6575817	2856.8	6606835	2090	6608356	2095.5	**6608762**	1804.1	**-0.0061**
18	6556239	2855.9	6563180	1977.9	6581089	1983.6	**6586074**	1808.7	**-0.0757**
19	6586219	2857	6614261	2012.5	6617187	2017.4	**6618374**	1809.9	**-0.0179**
20	6616931	2840	6636568	1848	6638626	1853.6	**6657391**	1809.3	**-0.2819**

表 11：城市点数量 n=500 算例实验结果

结果对比总表如下所示，其中 Av.R 表示该算法下对应规模算例的平均计算结果(Average Result)，Av.T 表示该算法下对应规模算例的平均用时(Average Time)，Ip.G 表示该算法与本文算法在对应规模下的算例所求结果之间的改进比例(Improve Gap)，其计算公式为 $\mathrm{Ip.G}(\%) = 100 \times \frac{(Av.R_{Algrithom} - Av.R_{RLS})}{Av.R_{RLS}}$：

	TS			**GRS**		
	Av.R	Av.T(s)	Ip.G	Av.R	Av.T(s)	Ip.G
10	1785.7	<1	0.000%	1785.7	<1	0.000%
20	7965.8	<1	0.000%	7965.8	<1	0.000%
50	50380.4	14.2	-0.005%	50382.9	9.4	0.000%
100	211857.8	149.9	-0.0104%	211867.9	106.6	-0.0055%
200	850836.5	1436.2	-0.0728%	851141.9	1086.7	-0.0369%
500	6549673.0	2855.8	-0.7125%	6576988.5	1963.0	-0.2949%

表 12：与 TS & GRS 结果对比汇总表

	HES			**RLS**		
	Av.R	Av.T(s)	Ip.G	Av.R	Av.T(s)	Ip.G
10	1785.7	<1	0.000%	1785.7	<1	
20	7965.8	<1	0.000%	7965.8	<1	
50	50382.9	9.4	0.000%	50382.9	7.7	
100	211879.4	106.7	-0.0001%	211879.7	50.5	

200	851437.8	1088.5	-0.0017%	851451.9	650.6	
500	6589292.0	1967.1	-0.1073%	6596363.4	1805.4	

表 13：与 HES 结果对比汇总表

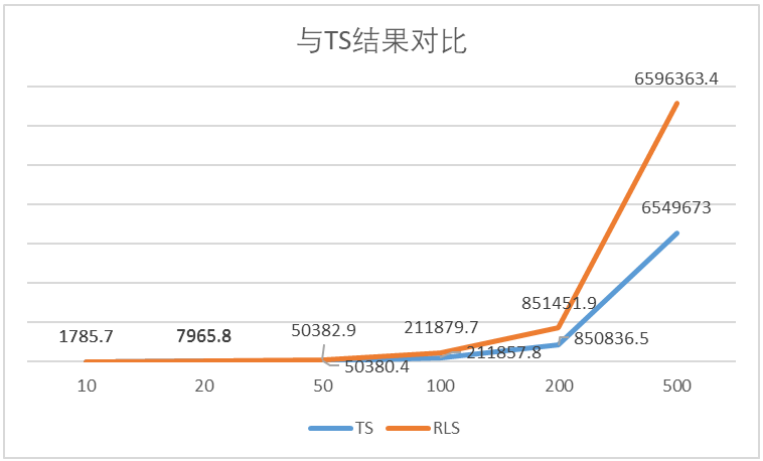

图 10：与 TS 结果对比堆积折线图

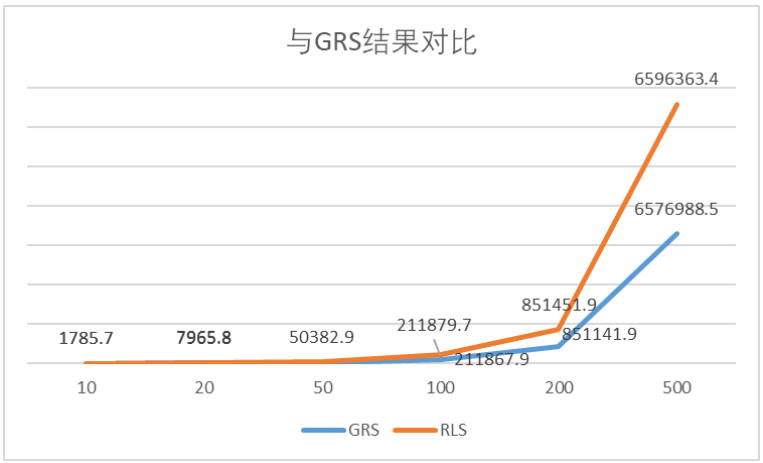

图 11：与 GRS 结果对比堆积折线图

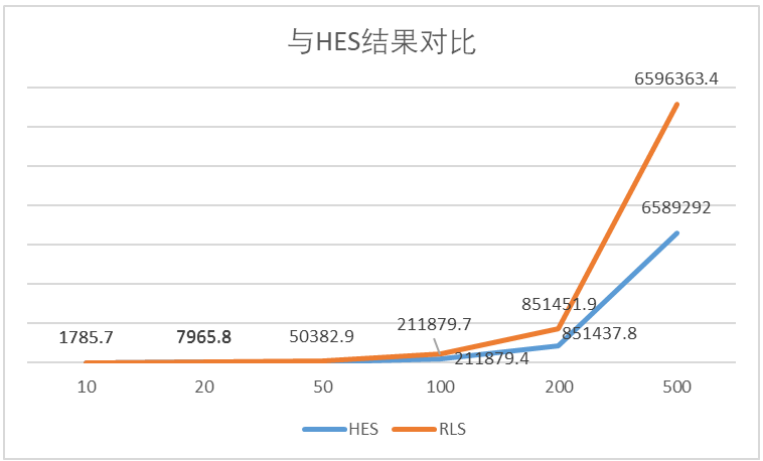

图 12：与 HES 结果对比堆积折线图

由实验数据可知，相对于已有的基于 TRPP 问题的三种算法 TS、GRS 和 HES 所给出的最

佳结果，本文提出的 RLS 算法在小规模算例(n≤20)上能够快速计算出精确最优解，在较大规模算例(n≥50)上对于各个算法都有不同程度的改进，共改进了 29 个现有算法的最佳结果，而在搜索速度上，本算法也在大部分算例上均有明显改进，体现了 RLS 算法较高的搜索效率与搜索能力。

4.4 本章小结

本章主要介绍了三个实验，首先是各算符局部搜索性能与速度的测试实验，该实验证明了本文采用的 8 种算符都具有较强的搜索能力与较快的搜索速度，且从实验数据可知本文提出的新算符具有最优越的局部搜索能力，对贪心随机构造的初始解的改进比例接近 150%；其次进行了点集排序搜索与点集调整搜索两大环节中各个算符搜索顺序组合的实验，通过该实验确定了具有最佳搜索性能的算符组合方式，并在最终的测试集算例实验中采用了该种组合方式；最后给定各种规模(n=10,20,50,100,200,500)的算例测试集上进行了完整的算法测试实验并与既有算法的结果进行了对比，结果表明本算法在较小规模(n≤50)上能够快速计算出已有最优解，对于较大规模算例，本算法在搜索时间更短的同时做出了大幅的改进，实验结果证明了本算法突出的搜索性能。

5. 总结与展望

5.1 全文工作总结

TRPP(Traveling Repairman Problem with Profit)是基于传统 TSP(Traveling Salesman Problem)的更加切合实际的变种问题，TRPP 加入了访问城市点能够获得对应利润以及该利润值随时间线性下降的约束条件，同时不要求遍历访问所有城市点。该问题在现实生活中的应用包括包裹配送人员的任务分配，物流调度机器人的路径规划，灾后救援中的决策问题，是一个与实际生活高度相关的组合优化问题。

由于 TRPP 已被证明为 NP-hard 问题，对于较大规模的算例难以用精确算法进行求解，只能采用启发式或元启发式算法，目前已有的针对 TRPP 的启发式算法包括禁忌搜索算法、贪心随机自适应迭代搜索算法以及混合进化算法，逐步对给定的各规模算例做出了改进，本文根据上述已有成果进行总结与研究，提出一种基于强化学习思想的多邻域搜索框架，主要的研究成果如下：

（1）算法框架中包括常用的点排序局部搜索算符 2-opt、Or-opt、Swap、Insert 以及用于调整访问点集的搜索算符 Add、Drop 和 InterChange。同时提出一种能够根据现有解集进行自我学习与自我调整的 k-opt 搜索算符，该算符采用蒙特卡洛搜索树框架，具有初始化、模拟、选择和反向传播四个环节，在每一次选择过程中都根据现有知识，即对当前解空间中获得过的所有改进解的结构进行分析，随后按照轮盘赌的方式概率选择较有可能带来改进的边进行操作，这种方式能够兼顾搜索的集中性与分散性，保证搜索机制既能深入局部区域进行搜索又能不陷入某处的局部最优而最终找到全局最优解；

（2）另外，算法在局部搜索环节学习获得的知识还作用于搜索后的扰动过程，实现更有针对性的扰动，有效提升了算法的搜索速度；

（3）最后通过三组实验证明了算法框架从单个算符到局部搜索组合顺序再到整体框架流程的设计合理性与突出性能。从各组实验数据可知框架中的每一个局部搜索算符都有较强的搜索能力与较快的搜索速度；并且通过实验数据选择了更为有效的算符组合方式；在给定算例测试集上的实验结果也直接反映了本算法突出的高效性与鲁棒性。

5.2 未来研究展望与方向

本文提出的基于强化学习的变邻域启发式算法只在引文[1]给出的算例测试集上进行测

试，没有完成实体实验或仿真实验，因此如果直接应用于求解实际生活中的任务分配，物流机器人调度等 TRPP 问题时可能出现诸多现实问题，例如难以处理多机器人路径规划的冲突或任务紧急性的差异等；另外，本算法没有在 TRPP 以外的其他节点路径问题给出的测试集上进行实验，可能存在算法对问题过于拟合的情况导致算法的普遍性与可扩展性较差，甚至需要针对具体问题进行具体的大幅修改。针对以上问题，未来的研究展望与研究方向包括以下几点：

（1）首先基于 Qt 图形用户应用程序开发框架进行简单仿真环境的搭建，模拟物流配送工厂内移动机器人的配送任务场景，最初从单机器人开始进行实验，实验的具体任务是对多个包裹进行取送，仿真实验中假设机器人的容载量无限，每完成一个包裹的取或送都能够获得一定利润，通过多次实验对算法的鲁棒性进行测试；由于实际问题中灾后救援场景往往有多支救援队同时实施救援工作，因此后续还需进行多机器人合作仿真实验，将算法调整修改为多代理的协同任务框架，并仿真测试算法在多机器人协同工作时是否出现任务分配不合理或路径规划冲突等问题，相信通过仿真实验能够暴露出本算法在实际应用中可能出现的问题并及时解决，更大程度提升算法框架的实际应用能力与稳定性；

（2）未来计划将本算法框架在更多的节点路径问题(例如 PTP、OP、MCPTDR 等)上进行算例测试，尝试找到相关问题的共性并对原框架做出相应调整，同时也能够测试框架在不同问题上的稳定性与可扩展性，由于多数节点路径问题都是对现实生活中的组合优化问题的数学建模，如果本算法框架经过修改能够适应更多的相关问题，则在实际应用上也将带来人员任务分配、车辆调度或者网络资源分配效率的大幅提高。

致谢

本文即将完成，在此由衷感谢我的论文指导老师范衡教授，不论是毕业设计的完成过程还是最后的论文撰写过程都离不开范老师的悉心指导，在此向范老师表达我衷心的谢意！

另外，本工作的完成离不开我在深圳人工智能与机器人研究院实习期间的项目导师付樟华博士，从拟定毕业设计题目开始，付博士就给了我很多的指导性意见，并且在前期研究现有论文工作的阶段不断地教我科研上以及代码工程上的各种知识，这是我第一次独立地完成一项科研项目，如果没有付博士的指导将很难完成，感谢付樟华博士的悉心教导！

从大二学年我加入汕头大学人工智能与机器人实验室的 NEO 团队起，就逐步在陈文钊等师兄的指导下开始接触科研项目，师兄们给我详细讲解了关于如何查找文献、整理文献以及快速阅读文献的方法，同时在实验室开展科研的经历也锻炼了我的工程能力，在 NEO 团队中我收获了非常多知识与经验，对我本次毕业设计的完成具有极其重要的作用，在此向各位师兄和各位同学的耐心指导与包容表示真诚地感谢，也祝愿他们今后在工程领域或科研领域都能获得满意的成就！

在完成此次毕业设计与论文撰写的过程中，由于新冠肺炎疫情的爆发，我无法返回深圳实习单位或学校开展实验，因此大部分工作和全篇论文都在家完成，在这样的情况下我能够有一个安心舒适的环境开展工作，离不开我的家人的帮助、关心和鼓励，是他们为我营造了一个良好的科研环境，对本工作的完成起到了不可或缺的重要作用，衷心感谢他们！

目前中国国内的疫情已趋于稳定，相信不久之后就能够全面复工复学，我也能够回到熟悉的校园参加毕业设计答辩工作，这一切都离不开所有医护人员与行政人员的共同努力，在此向广大奋战在一线的医护人员和行政人员表示由衷的敬意和满腔的感谢！

不论是前期的文献查阅阶段还是在后期的算法设计与实现工作中，我参考了大量的论文、博客与书籍，这些资料给了我非常大的启发与指导，极大地推动了我毕业设计的工作开展，谨向这些论文，博客与书籍的作者表示衷心的感谢！

参考文献：

[1] Dewilde, T., Cattrysse, D., Coene, S., Spieksma, F. C., & Vansteenwegen, P. (2013). Heuristics for the traveling repairman problem with profits. Computers & Operations Research, 40(7), 1700-1707.

[2] Beraldi P, Bruni M E, Laganà D, et al. The risk-averse traveling repairman problem with profits[J]. Soft Computing, 2019, 23(9): 2979-2993.

[3] Bruni M E, Beraldi P, Khodaparasti S. A fast heuristic for routing in post-disaster humanitarian relief logistics[J]. Transportation research procedia, 2018, 30: 304-313.

[4] Goemans M, Kleinberg J. An improved approximation ratio for the minimum latency problem[J]. Mathematical Programming, 1998, 82(1-2): 111-124.

[5] Ban H B, Nguyen K, Ngo M C, et al. An efficient exact algorithm for Minimum Latency Problem[J]. J. PI, 2013, 10: 1-8.

[6] Wu B Y. Polynomial time algorithms for some minimum latency problems[J]. Information Processing Letters, 2000, 75(5): 225-229.

[7] Salehipour A, Sörensen K, Goos P, et al. Efficient GRASP+ VND and GRASP+ VNS metaheuristics for the traveling repairman problem[J]. 4or, 2011, 9(2): 189-209.

[8] Bang B H, Nghia N D. Improved genetic algorithm for minimum latency problem[C]//Proceedings of the 2010 Symposium on Information and Communication Technology. 2010: 9-15.

[9] Silva M M, Subramanian A, Vidal T, et al. A simple and effective metaheuristic for the minimum latency problem[J]. European Journal of Operational Research, 2012, 221(3): 513-520.

[10] Feillet D, Dejax P, Gendreau M. Traveling salesman problems with profits[J]. Transportation science, 2005, 39(2): 188-205.

[11] Vansteenwegen P, Souffriau W, Van Oudheusden D. The orienteering problem: A survey[J]. European Journal of Operational Research, 2011, 209(1): 1-10.

[12] Bruni M E, Brusco L, Ielpa G, et al. The Risk-averse Profitable Tour Problem[C]//ICORES 2019-Proceedings of the 8th International Conference on Operations Research and Enterprise Systems pp. 459–466. 2019.

[13] Fischetti M, Gonzalez J J S, Toth P. Solving the orienteering problem through branch-and-cut[J]. INFORMS Journal on Computing, 1998, 10(2): 133-148.

[14] Campos V, Martí R, Sánchez-Oro J, et al. GRASP with path relinking for the orienteering problem[J]. Journal of the Operational Research Society, 2014, 65(12): 1800-1813.

[15] Lu Y, Benlic U, Wu Q. A memetic algorithm for the Orienteering Problem with Mandatory Visits and Exclusionary Constraints[J]. European Journal of Operational Research, 2018, 268(1): 54-69.

[16] Dang D C, Guibadj R N, Moukrim A. An effective PSO-inspired algorithm for the team orienteering problem[J]. European Journal of Operational Research, 2013, 229(2): 332-344.

[17] Erkut E, Zhang J. The maximum collection problem with time‐dependent rewards[J]. Naval Research Logistics (NRL), 1996, 43(5): 749-763.

[18] Avci M, Avci M G. A GRASP with iterated local search for the traveling repairman problem with profits[J]. Computers & Industrial Engineering, 2017, 113: 323-332.

[19] Lu Y, Hao J K, Wu Q. Hybrid evolutionary search for the traveling repairman problem with profits[J]. Information Sciences, 2019, 502: 91-108.