适用于可充电车辆的多车场乡村邮差问题多趟算法

Eashwar Sathyamurthy[a]、Jeffrey W. Herrmann[b]和 Shapour Azarm[a]

[a]美国马里兰大学机械工程系[b]美国马里兰大学系统研究所、机械工程系

arXiv:2303.03481v1 [cs.MA] 6 Mar 2023

摘要

本文研究了乡村邮差问题的一种扩展，即具有多个仓库、可充电且可重复使用的车辆，并且车辆可在容量约束下执行多次行程。本文提出了一种新的混合整数线性规划（MILP）公式，以求解该问题的最优解。本文还针对该问题提出了一种称为多趟算法的新启发式方法，并通过在基准实例和根据道路地图生成的真实世界实例上进行测试，将其解与文献中启发式算法的解以及通过MILP公式获得的最优解进行比较。结果表明，所提出的启发式方法能够求解所有实例，并且在总计 39 个实例中的 37 个实例上产生了优于文献中启发式算法的解。由于对内存和计算能力的需求较高，用于求解 MILP公式的 Gurobi 优化器虽然能够产生最优解，但只能求解基准实例，无法求解真实世界实例。

关键词- 启发式方法、多趟算法、乡村邮差问题、容量约束、可充电和可重复使用的车辆。

1 引言

本研究探讨带有可充电和可重复使用车辆的多仓库农村邮递员问题（MD-RPP-RV），其动机源于一个现实问题：如何为无人机（UAVs）规划路线。在该问题中，无人机需要从位于关注区域（AOI）不同位置的基地飞往可能结冰的道路路段。有关道路路段可能结冰的信息，是通过分析关注区域内道路状况的历史数据并结合专家意见获得的。但是，这些道路路段需要由无人机进行目视检查。

车辆的任务是利用其传感器收集可能结冰道路的数据并确认这些道路确实结冰，以便向驾驶员发出危险路况警告

。车辆必须在巡视结冰道路时沿道路行驶，并返回其基地。此外，由于电池中的能量有限，车辆的行程范围（容量）也受到限制，因此必须在电池耗尽之前前往基地充电。感兴趣区域中的道路地图表示为一个图，其中基地是称为仓库的节点子集，而可能结冰的道路是称为必需边的边子集，车辆必须 traversed。

本文将 MD-RPP-RV 建模为混合整数线性规划（MILP），提出一种基于启发式的算法来寻找可行解，并介绍了一项研究的结果，该研究将其性能与使用 Gurobi 优化器求解 MILP公式所获得的最优解进行了比较 [1]。

MD-RPP-RV 与容量约束弧路径问题（CARP）的变体相关 [6]。中国邮差问题（CPP）和乡村邮差问题（RPP）是众所周知的 CARP 变体，二者都旨在确定单个邮差的最短环游路线。CPP [2] 要求所有边 E至少被 traversed 一次，而 RPP [3] 要求遍历 E 的一个子集。尽管 CPP 和 RPP 之间存在许多相似之处，但它们的计算复杂性不同。CPP 可以利用匹配算法 [4], 在多项式时间内求得最优解，而 RPP 是 NP难问题 [5]。通过仅考虑一个仓库并将车辆限制为单次行程，可以将 MD-RPP-RV 归约为 RPP，因此 MD-RPP-RV 的求解也是 NP难的。

MD-RPP-RV 是多趟 [11, 21] 和多车场 [12–14]型 CARP 的组合。在多行程车辆路径问题中，每辆车辆都能够执行多次行程，以遍历所有必需边。在多车场 CARP 中，车辆可以从多个仓库出发。MD-RPP-RV 与多车场乡村邮差问题（MD-RPP）最为相似；在该问题中，位于多个仓库且容量有限的车辆必须遍历边的一个子集。在 MD-RPP [15], 中，车辆从同一仓库出发并返回该仓库，且仅限执行单次行程，而 MD-RPP-RV 则并非如此。MD-RPP-RV 可以表示许多现实世界中的问题，例如为扫雪作业优化车辆路径，或规划无人机路径以检查泄漏的燃气管线或变形的铁路轨道。

在相关文献中，求解弧路径问题通常有三种方法。这些方法包括启发式方法、元启发式方法和精确方法。基于启发式方法的方法通常会生成次优解，但无法保证解的质量。Frederickson 等 [16] 提出了第一个用于最小-最大 CPP 的构造启发式方法。Ahr 等 [17] 提出了增广-合并和聚类启发式方法，在大多数实例中找到了更优的解。尽管这些解可能是次优的，但这些启发式算法能够快速生成可行解，这在处理大规模实例（即包含许多车辆和弧的实例）时非常有用。

元启发式方法 [18–23] 是通用方法，通过调整超参数使其适用于特定问题，从而能够解决各种不同的问题。通常，这些方法以基于启发式的方法生成的初始解为起点，探索解空间，并收敛到一个新解。Hertz 等人 [24] 提出了一种名为 CARPET 的禁忌搜索启发式算法，其结果优于基于启发式的方法 [7]，并在大多数基准实例中得到了最优解。Lacomme 等人 [25] 提出了第一个用于 CARP 的遗传算法（GA），结果表明，在计算时间较短的基准实例中，GA 得到的解优于 CARPET 得到的解

。然而，调整超参数可能是一个耗时且需要反复试验的过程；此外，这些方法需要一个初始解，其计算时间可能因初始解的质量和求解过程的随机性而异。

精确方法 [26–29] 通过将问题建模为混合整数线性规划，并采用分支定界技术求解，得到最优解 [30]。Fern´andez 等人[10] 研究了无向图上的 MD-RPP，并提出了具有两种不同目标函数的混合整数线性规划公式：(i) 最小化总路径成本；(ii) 旨在最小化最长路径长度的极小极大目标函数。他们提出了一种分支切割算法，可针对这两种目标生成均衡路径。由于寻找并验证最优解需要检查（至少隐式地检查）所有可行解，因此，这些方法在处理大型实例时，可能因计算和内存需求而变得不切实际。

开发多趟算法的主要原因在于自动驾驶车辆日益普及，以及它们执行监视操作的能力 [31–33]。车辆路径规划需要考虑电池限制、频繁充电和重复使用，以有效完成任务。所提出的多趟算法通过纳入电池和充电约束来应对这些限制。

本文的主要贡献如下：

1. 本文提出了一类新的弧路径问题，称为 MD-RPP-RV。在该问题中，车辆可充电且可重复使用，从多个仓库出发执行多趟行程，并且最终可以返回不同的仓库。尽管已有文献从节点路径问题 [34–38], 的角度研究了使用可充电、可重复使用且能够执行多趟行程的车辆，以及在弧路径问题中使用可重复使用车辆（如多行程车辆路径问题 CARP） [11, 21], ，但据我们所知，文献尚未研究将车辆的可充电性和可重复使用性相结合，通过多趟行程解决多车场弧路径问题的思路。

2. 本文将该问题表述为混合整数线性规划公式，可用于求解最优路线。

3. 所提出的多趟算法能够快速求解实例，包括基于现实道路地图的实例（图6）。对多趟算法、现有 CARP启发式算法 [7–9], 以及 Gurobi 优化器进行测试的结果表明，在39个实例中的37个实例上，多趟算法找到的解优于其他 CARP启发式算法。

本文其余部分的组织如下：第2节介绍所考虑的假设，并为 MD-RPP-RV 提出一个混合整数线性规划公式。第3 节简要介绍所提出的多趟算法。第4 节给出在 MD-RPP-RV 实例上测试多趟算法和其他 CARP 算法所获得的结果。第5 节通过总结全文作出结论。

2 问题建模与假设

本节介绍所考虑的假设，以帮助将车辆路径规划以收集道路信息的问题转化为组合优化问题。

2.1 假设

对于 MD-RPP-RV，我们作出以下假设：

1. 所有车辆均从各自的仓库出发，且电池容量充满。2. 车辆出发、停止和充电的仓库均位于道路交叉路口。3. 如果某条边为必经边，则至少有一辆车辆必须经过该边。4. 所有车辆的容量（充满电时的运行时间）相同。5. 所有车辆的速度相同。6. 在所有仓库为车辆电池充电所需的时间相同。无法进行部分充电。7. 表示感兴趣区域的图是连通的。8. 经过一条边所需的时间仅取决于该边的长度和车辆速度。

以下问题建模纳入了这些假设。

2.2 问题建模

令 $G=(N,E,T)$ 表示对感兴趣区域进行建模的无向图，其中 N 是节点（位置）的集合， E 是连接节点（$i,j \in N$）的边（$(i,j) \in E$）的集合，而 T 是边权重（$t(i,j) \in T$）的集合。一条边表示一段道路路段；令 $l(i,j)$ 表示该道路路段的长度，单位为米。令 S 表示所有车辆行驶道路路段时的恒定速度，单位为米每秒。令 $t(i,j)$ 表示边权重，即车辆通过边（i,j）所需的时间： $t(i,j) = l(i,j)/S$。

令 $E_u \subseteq E$ 表示需要访问的必需边集合。令 N_d 表示车辆可以出发、停靠或补充燃料（充电）的仓库集合。 $N_d \subseteq N$。令 T_{max} 表示车辆充电后可运行的最长时间。令 K 表示车辆数。令 R_T 表示车辆充电所需的时间。令 F 表示通过迭代过程（第 4.3.1节）确定的车辆最多可执行的行程数。

MD-RPP-RV 可表述为以下混合整数线性规划：

最小化 β

$$\text{such that} \sum_{(B(k),j)\in E} x(k,1,B(k),j) = z(k,1), \quad k = 1,...,K \tag{1}$$

$$z(k,f) - z(k,f+1) \geq 0, \quad k = 1,...,K, f = 1,...,F-1 \tag{2}$$

$$\sum_{(i,d)\in E, d\in N_d} x(k,f,i,d) = y(k,f,d), \quad k = 1,...,K, f = 1,...,F \tag{3}$$

$$y(k,f-1,d) \geq \sum_{(d,j)\in E} x(k,f,d,j), \quad k = 1,...,K, f = 2,...,F, d \in N_d \tag{4}$$

$$z(k,f) - \sum_{d\in N_d} y(k,f,d) = 0, \quad k = 1,...,K, f = 1,...,F \tag{5}$$

$$\sum_{f=1}^{F} \sum_{(i.j)\in E} x(k,f,i,j) \times t(i,j) + (\sum_{f=1}^{F} z(k,f) - 1) \times R_T \leq \beta, \quad k = 1,...,K \tag{6}$$

$$\sum_{(i,j)\in E} x(k,\ f,\ i,\ j) \times t(i,\ j) \leq T_{max}, \quad k = 1,\ ...,\ K,\ f = 1,\ ...,\ F \tag{7}$$

$$\sum_{(i,j)\in E, i\in N_d} x(k,\ f,\ i,\ j) - \sum_{(i,\ j)\in E,\ j\in N_d} x(k,f,i,j) = 0, \quad k = 1,\ .....K、\ f = 1,\ .....F \tag{8}$$

$$\sum_{j\in N} x(k,f,i,j) - \sum_{j\in N} x(k,f,j,i) = 0, \quad k = 1,...,K, f = 1,...,F, i \in N/\{N_d\} \tag{9}$$

$$\sum_{k=1}^{K}\sum_{f=1}^{F} x(k,f,i,j) + \sum_{k=1}^{K}\sum_{f=1}^{F} x(k,f,j,i) \geq 1, \quad \forall (i,j),(j,i) \in E_u \tag{10}$$

$$\sum_{(i,\ j)\in E} x(k,f,i,j) \leq z(k,f) \times M, \quad k = 1,...,K, f = 1,...,F \tag{11}$$

$$\sum_{(i,\ j)\in\delta(S)} x(k,f,i,j) \geq 2 \times x(k,f,p,q) \quad k = 1,...,K, f = 1,...,F, \forall S \subseteq N/\{N_d\}, (p,q) \in E(S) \tag{12}$$

$$x(k,f,i,j) \in [0,1], \quad k = 1,...,K, f = 1,...,F, \forall (i,j) \in E \tag{13}$$

$$y(k,f,d) \in [0,1], \quad k = 1,...,K, f = 1,...,F, d \in N_d \tag{14}$$

$$z(k,\ f) \in [0,1],\ k = 1,...,K, f = 1,...,F \tag{15}$$

$$\beta \in R^{+},\ M >> |E| \tag{16}$$

该模型中有三组二进制决策变量。 $x(k,f,i,j) = 1$若车辆 k 在其第 f次行程中沿边（i，j）从节点 i traverses 到节点 j，则为1。 $x(k,f,j,i)$ =若车辆 k 在其第 f次行程中沿边（i，j）从节点 j到节点 i，则为1。注意，车辆从仓库开始一次行程，沿图中的路径行驶，并在仓库结束该行程，以便充电。 $y(k,f,d)$用于记录每辆车结束行程时所在的车场节点。若车辆 k 在车场节点$d \in N_d$结束其第 f次行程，则该变量为1，否则为0。 $z(k,f)$若车辆 k使用其第 f次行程，则为1，否则为0。对于$k = 1$，.....， K， $B(k)$是 N_d中的一个节点，使得车辆 k初始位于仓库 B（k）。 $B(k)$的值是给定的。

目标函数 β 表示任何车辆完成所有行程及行程间充电所需的总时间中的最大值。约束 (1) 确保如果车辆 k 已从仓库节点 $B(k)$ 开始其首次行程，则使用 $z(k,\ 1)$ 对其进行跟踪。约束 (2) 通过确保只有在车辆 k使用行程 f 时才使用行程 $f+1$，来限制不必要的行程。约束 (3) 使用 $y(k,\ f,\ d)$ 跟踪车辆 k 的每个行程 f结束时所在的仓库节点 $d \in N_d$。约束 (4) 在必要时强制车辆 k 从与其上一行程 $f-1$ 结束时相同的仓库节点 $d \in N_d$ 开始行程 f。约束 (5) 确保车辆 k若开始行程 f，则必须在任一仓库节点 $d \in N_d$结束。

约束 (6) 为所有车辆的最大行程时间设定上界，并考虑充电时间。请注意，约束 (6) 仅对已使用的行程添加充电时间$f > 1$，该行程通过 $z(k,\ f)$ 确定。约束 (7) 确保每辆车执行的所有行程 k 均满足电池约束。约束 (8) 确保每辆车执行的所有行程进入或离开仓库节点的次数相同。约束 (9) 确保每辆车的所有行程均形成连续流。约束 (10) 确保所有车辆执行的所有行程至少遍历每条必需边一次。约束 (11) 确保车辆在未使用的行程中不遍历任何边。约束 (12) 是一种子回路消除约束，其依据的是 Chen等人 [15]提出的一种约束。集合 S 是一个不包含仓库的节点集合，$E(S)$ 是 S中节点之间必需边的集合，而 $\delta(S)$ 是仅有一个节点属于集合 S的边的集合。该约束确保，如果车辆遍历一条必需边 $(p,\ q) \in E(S)$，那么它也会遍历进入和离开 S的边。

遗憾的是，该混合整数线性规划的规模会随节点数呈指数增长。仅约束 (12) 就有 $2^{|N/N_d|}$ 个子集 S，因此该模型总共需要 $2^{|N/N_d|} \times K \times F$ 个此类约束。下一节将介绍我们的多趟算法，该算法解决了计算复杂度问题，使其适用于求解大型实例。

3 求解方法

本节首先通过一个示例介绍算法用来遍历必需边的多行程路线。然后逐步讲解该算法的伪代码，以帮助理解算法的工作原理。

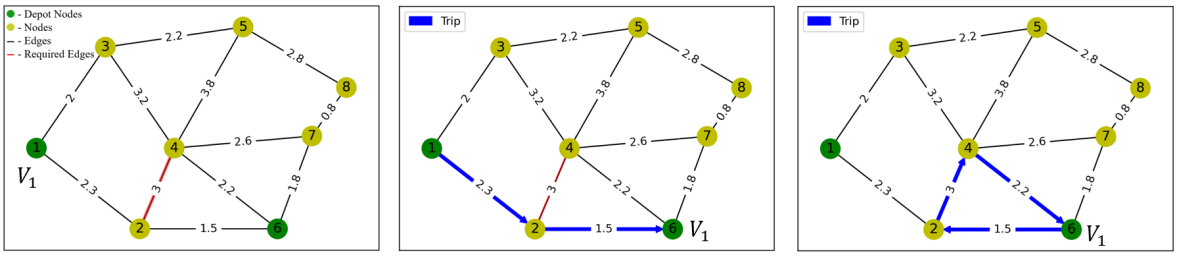
3.1 多行程路线

一个可行解为每辆车指定一条路线。一条路线包含一个或多个行程；每个行程始于并终于同一个或不同的仓库。行程长度受容量约束限制。虽然车辆的任务是覆盖必需边，但为了接近一条或多条必需边，车辆可能需要执行不包含任何必需边的行程。

考虑一个由无向图 G 构成的问题实例，该图有 8 个节点、12 条边、2 个仓库节点和 1 条必需边，如图 1(a)所示。只有一辆车辆（V_1）；其容量为 7 个时间单位，从仓库节点1出发，充电时间为 1.1 个时间单位。

由于容量约束，车辆无法在一次行程中直接通过必需边（2，4）。覆盖这条边的最优行程为 {1-2-4-6}，但这需要 7.5 个时间单位，超过车辆 7 个时间单位的容量，因此不可行。

在一条可行的多行程路线中，车辆从其当前仓库行驶至另一仓库（节点6），以更接近必需边（图1(b)），在该仓库充电，然后完成一趟包含该必需边的行程（图1(c)）。



(a) (b) (c)

图1：（a）MD-RPP-RV的一个示例实例，（b）车辆 V_1 接近必需边。（c）车辆 V_1 遍历必需边。

第一趟行程 {1-2-6} 需要车辆用时3.8个时间单位完成。随后，车辆在节点6充电，需要1.1个时间单位。最后，车辆进行第二趟行程 {6-2-4-6}，完成该行程需要6.7个时间单位。每趟行程均可行，因为其持续时间小于7个时间单位的容量。车辆完成该路线并遍历必需边共需11.6个时间单位，即两趟行程时间与充电时间之和。

3.2 多趟算法

表 1 列出了所提出算法伪代码中使用的符号。

表1：符号说明

符号	Type	解释
$G=(N,\ E,\ T)$	输入	由节点、边和边权重构成的图
N_d	输入	仓库集合 $N \subseteq N_d$
E_u	输入	必需边集合
K	输入	车辆数
$t_k,\ k=1,\ \ldots\ldots,\ K$	输入	车辆 k 可用的时间
$P_k,\ k=1,\ \ldots\ldots,\ K$	输入	车辆 k 截至时间 t_k的路径
$n_k,\ k=1,\ \ldots\ldots,\ K$	输入	车辆 k 在时间 t_k的位置
$u_k,\ k=1,\ \ldots\ldots,\ K$	输入	车辆使用时长 k （自上次充电以来经过的时间）
$y_k,\ k=1,\ \ldots\ldots,\ K$	输入	车辆 k 在 P_k 中最近一次到达仓库的时间
R_T	输入	车辆在仓库充电所需的时间
C	输入	车辆容量（时间）
$P_k,\ k=1,\ \ldots\ldots,\ K$	输出	车辆 k 的路径
$y_k,\ k=1,\ \ldots\ldots,\ K$	输出	车辆 k 到达 P_k 中最后一个仓库的时间

多趟算法综合考虑车辆的位置（n_k）、利用率（u_k）和可用性（t_k），以确定能够遍历必需边的可行且高质量的行程。

```
y_k, P_k = MultiTripAlgorithm(G, N_d, K, t_k, P_k, n_k, u_k, y_k, E_u, R_T, C)

Procedure
1. while E_u is not empty
2.     k = argmin {t_1, …, t_K}                          # find first available vehicle
3.     Q*, T*, e*, T_e = closestfeasibleedge(G, E_u, N_d, u_k, n_k, C)
4.     if e* > 0
5.         add Q* to P_k                                 # vehicle k traverses edge e*
6.         add T* to u_k
7.         add T* to t_k
8.         n_k = end of Q*
9.         if n_k ∈ N_d                                  # recharge at a depot
10.            y_k = t_k                                 # arrival time at depot
11.            t_k = y_k + R_T                           # vehicle k is available after recharging
12.            u_k = 0
13.        remove e* from E_u
14.    else                                              # there is no feasible edge
15.        e* = argmin {T_e: e ∈ E_u }                   # find closest edge
16.        d*, V_d* = closestfeasibledepot(G, T_e, N_d, n_k, u_k, C, e*)
17.        if V_d* < infinity
18.            Q_1 = shortestpath(G, n_k, d*)            # shortest path to depot d*
19.            add Q_1 to P_k                            # vehicle k goes to depot d*
20.            y_k = t_k + time(Q_1)                     # arrival time at depot
21.            t_k = y_k + R_T                           # vehicle k is available after recharging
22.            n_k = d*
23.            u_k = 0
24.        else
25.            t_k = infinity                            # this vehicle cannot feasibly move
26. for k = 1, …, K
27.    if n_k ∉ N_d                                      # vehicle k is not at a depot at end
28.        Q_1 = bestdepot(G, n_k, N_d)                  # shortest path to a depot
29.        add Q_1 to P_k                                # vehicle k goes to depot d* at end
30.        y_k = t_k + time(Q_1)                         # arrival time at depot
31.        n_k = end of Q_1
32. return y_k, P_k
```

图2：多趟算法的伪代码

多趟算法（图2）会不断迭代，直到遍历完所有必需边（第 1 行）。每次迭代中，算法会确定接下来哪辆车辆可用（第 2 行），以及该车辆单次行程能够抵达的最近边。随后，算法使用 $closestfeasibleedge()$子程序确定遍历该边的路径，该子程序采用 Dijkstra算法 [39]，并更新车辆的位置（n_k）、利用率（u_k）、可用性（t_k）以及必需边（E_u）（第 4-13 行）。

如果需要多行程路线（第 15-23 行），算法会先确定距离车辆位置最近的必需边（e^*），然后使用 $closestfeasibledepot()$子程序将车辆移至更靠近 e^* 的仓库。算法对单趟和多趟情形采用不同的处理方式：在单趟情形下，将第一辆可用车辆分配给一条必需边；而在多趟情形下，则将最近的必需边分配给该车辆。

如果车辆无法遍历必需边或移动到另一仓库（由于容量约束），算法会使该车辆无法移动（第 17 行）。（当 G 断开连接时，可能会发生这种情况。）遍历完 E_u 中的所有必需边后，算法会补全未在车场节点结束的部分车辆路径（第 26—31 行）。最后，对于每辆车辆，算法都会返回 k 的 P_k车辆路径，以及 y_k车辆到达 P_k中最后一个仓库的时间。下一小节将通过一个示例演示多趟算法的运行过程。

3.3 演示示例

考虑一个具有 9 个节点和 13 条边的无向图实例。如图 3所示，该实例包含一条必需边（8，9）、三个仓库（节点 1、5 和 6）以及两辆车辆，它们分别从节点 1 和 6 出发。车辆容量为 6 个时间单位，充电时间为 9 个时间单位。在图 3中，车辆位置（n_k）、车辆利用率（u_k）和车辆可用性（t_k）均包含两个元素（$\{\text{veh}_1, \text{veh}_2\}$）。

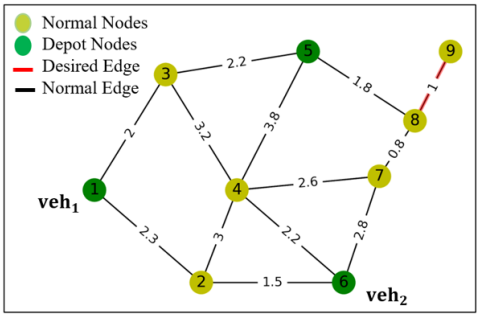
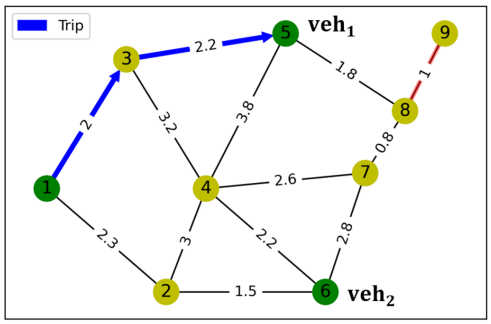


图 3：示例实例（n_k = 车辆位置，u_k = 车辆利用率，t_k =车辆可用性，C = 车辆容量，R_T = 充电时间，E_u = 必需边）

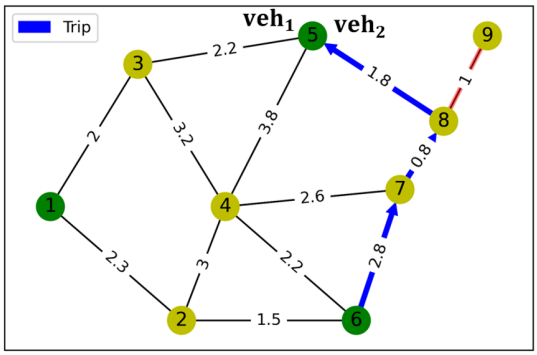
由于两辆车都无法在单次行程中通过这条必需边，算法将车辆1从节点1移动到更靠近该必需边的节点5。图4展示了此次移动后的更新状态。



$n_k = \{1,6\}$
$u_k = \{0,0\}$
$t_k = \{0,0\}$
$C = 6$
$R_T = 9$
$E_u = \{8-9\}$
$y_k = \{4.2,0\}$
$P_k = \{\{\{1-3-5\}\},\{\}\}$

图4：车辆1更新后的位置（n_k = 车辆位置， u_k = 车辆利用率， t_k = 车辆可用性， C = 车辆容量， R_T = 充电时间， E_u= 必需边）， P_k = 车辆路径， y_k = 车辆最后一次到达仓库）

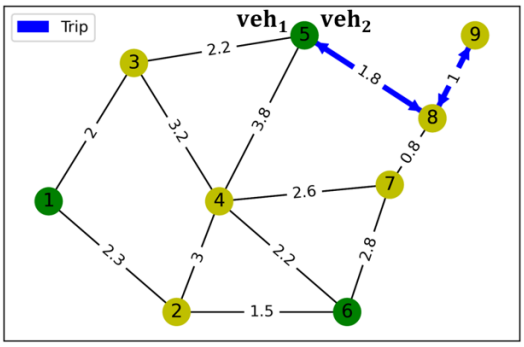
由于车辆1到达仓库节点5后需要充电，因此在下一次迭代中，车辆2是第一辆可用的车辆。算法将车辆2从节点6经由一次单次行程移动到节点5。图5 展示了此次移动后的更新状态。



$n_k = \{5,5\}$
$u_k = \{0,0\}$
$t_k = \{13.2,14.4\}$
$C = 6$
$R_T = 9$
$E_u = \{8-9\}$
$y_k = \{4.2,5.4\}$
$P_k = \{\{\{1-3-5\}\},\{6-7-8-5\}\}$

图5：车辆2更新后的位置（n_k = 车辆位置， u_k = 车辆利用率， t_k = 车辆可用性， C = 车辆容量， R_T = 充电时间， E_u= 必需边）， P_k = 车辆路径， y_k = 车辆最后一次到达仓库）

在下一次迭代中，车辆1是第一辆可用车辆，算法延长其路径，使其在一次行程中经过必需边；该行程从节点5的仓库出发，并返回该仓库。图6 展示了这次行程以及行程结束后的状态。最大行程时间等于 $\max\{18.8, 5.4\} = 18.8$ 个时间单位。



$n_k = \{5,5\}$
$u_k = \{2.8,0\}$
$t_k = \{16,14.4\}$
$C = 6$
$R_T = 9$
$E_u = \{\}$
$y_k = \{18.8,5.4\}$
$P_k = \begin{Bmatrix}\{\{1-3-5\},\{5-8-9-8-5\}\},\\ \{6-7-8-5\}\end{Bmatrix}$

图6：必经边遍历（n_k = 车辆位置， u_k = 车辆利用率，t_k = 车辆可用性， C = 车辆容量， R_T = 充电时间， E_u = 必需边）， P_k = 车辆路径， y_k = 车辆最后一次到达仓库）

此示例突显了这样一个事实：车辆可能需要多次行程才能经过必需边。下一节将讨论多趟算法和现有启发式算法得到的结果，并将这些解与通过求解混合整数线性规划得到的最优解进行比较。

4 结果与讨论

本节简要介绍了用于求解 CARP 的一些启发式算法，并将其作为多趟算法的基准方法。本节还详细介绍了实验中使用的实例，并说明了使用 Gurobi 优化器获得最优结果时所遵循的一些重要步骤。最后，本节讨论所选启发式算法、多趟算法和混合整数线性规划的结果，并分析多趟算法的计算工作量。

4.1 现有 CARP 启发式方法

本节简要回顾以下 CARP 方法：(i) 路径扫描算法，(ii) 增广合并算法，以及 (iii) 构造-打击算法。

4.1.1 路径扫描（PS）算法

路径扫描算法 [7] 依据短视优化准则，每次添加一条边，从而构造满足 CARP 约束的车辆路径。Golden等人 [7] 提出了五种优化准则，并选择能够生成最佳路径的准则作为该算法的输出。尽管优化准则的短视特性有助于算法更快地生成解，但由于采用贪心方法，这些解的质量可能较差 [8]。

4.1.2 AugmentMerge（AM）算法

增广合并算法 [7] 由两个阶段组成。增广阶段通过寻找从仓库到最近需求弧的最短路径，以及从该需求弧返回仓库的最短路径来构建路线，使每条解路线仅经过一条需求弧，并满足 CARP 约束。合并阶段按照路径成本降序排列解路线。随后，仅当满足 CARP 约束时，该阶段才会将最长路线与较短路线中的一条合并。最后，该阶段选择总路径成本最小的组合作为算法的输出。由于该算法在合并过程中只考虑两条路线，因此，与路径扫描算法相比，随着解时间的增加，其所得解的质量可能有所提高；但由于合并步骤仅尝试合并两条路线，生成的解不一定总是最优解 [8]。

4.1.3 ConstructStrike（CS）算法

构造-打击算法 [9] 反复执行三个步骤，直到所有必需边都被 traversed。第一步使用路径扫描算法 [7] 寻找满足 CARP 约束的解。第二步从图中移除该解中的所有边。第三步确定被移除的边是否使图不连通。如果仓库与剩余图之间不存在连接边，则图是不连通的。若图变得不连通，则添加连接仓库与剩余图的人工边，并将包含节点和未 traversed 必需边的图传递给第一步。尽管构造-打击算法通过在构建解路线后从图中移除该路线中的所有边，避免了对必需边的冗余 traversing，但随着迭代进行，由于图中的边数减少，构建解路线也变得更加困难，从而增加了路径成本。因此，所得解的质量可能较差。在某些情况下，该算法可能无法生成 traverses 所有必需边的解 [8]。

尽管这些启发式算法最初是为单仓库CARP问题提出的，但经过修改后，它们能够适用于多个仓库的MD-RPP-RV。下一小节将详细介绍用于测试的实例生成方法。

4.2 实例生成过程

我们从CARP 基准实例 [7] 和真实道路地图中创建了实例。实例生成过程随机选择五分之一的节点作为仓库，并随机选择三分之一的边作为必需边。因此，仓库数少于需求边数。实例生成过程将车辆数设定为需求边数的一半。由此生成的实例适用于测试所提出的多趟算法，并将其与CARP启发式方法进行比较。Sathyamurthy [41] 介绍了实例生成过程的详细信息。

下一小节将介绍创建不同实例集时所考虑的标准和假设，这些实例集用于获取结果。

4.2.1 A 组实例

我们通过实例生成过程，将 23 个具有无向图的 CARP 实例 [7] 转换为多仓库容量约束 RPP 实例。在这些实例中，车辆容量设为实例中最大边权重的两倍。这样设置是为了确保车辆在最长边被选为必需边时能够通过该边。

4.2.2 B 组实例

Set B 包含八个实例，这些实例由不同寒冷地区的道路地图创建而成。我们将道路地图转换为无向图，然后使用

实例生成过程。在这些实例中，车辆容量设为 31 分钟，即 DJI Mavic Pro 2 无人机标示的容量 [40]。

4.2.3 SetC 实例

Set C 包含八个实例，这些实例是在 Set B 实例的基础上加入风对边行驶时间的影响而生成的，这需要放宽假设 8（小节2.1）。在这些实例中，每条边有两个权重（行驶时间）：一个对应一个方向，另一个对应相反方向。我们通过考虑风对车辆行驶时间的影响，确定了不同的权重。在一个方向上，风可能会提高车辆的速度（相对于地面），从而缩短行驶时间；在另一个方向上，风可能会降低速度，从而延长行驶时间。Sathyamurthy [41] 介绍了这些计算的详细方法。对于 C 组实例，由于仅将图类型从无向改为有向，我们决定让车辆数（K）、需求边数（E_u）和仓库节点数（N_d）与 B 组实例保持相同。对于必需边（E_u），由于两个方向上的边权重不同，因此随机选择边的一个方向，即（i，j）或（j，i），作为必需边。

4.3 使用 Gurobi 求解最优解

本节介绍如何确定 Gurobi 生成最优解所需的行程数（F），以及对提供给 Gurobi 优化器的实例输入所做的修改。

4.3.1 确定行程数（F）

根据 MILP公式（第 2.2 节），为了在 Gurobi 中以编程方式定义决策变量，需要预先确定车辆数（K）和行程数（F）。但是，F 无法预先确定，因为每辆车辆 traversing 必需边时可能需要不同的行程数。因此，F 需要设为 K 辆车辆行程数的上界。为了确定该上界，采用一个简单的迭代过程。

初始时，我们将 $F = 1$，并使用 Gurobi 为该实例寻找最优解。如果存在可行解，则该实例得到解决。如果 Gurobi 报告不存在可行解，我们将 F 增加 1，并再次尝试求解。该过程持续进行，直到 Gurobi 找到一个可行的最优解。

随着 F 的增加，求解该问题所需的变量数量和随机存取存储器（RAM）也会增加。从 $F = 1$ 开始，并仅在需要时增加其值，这种方法避免了求解不必要的大型实例，因此更加高效。

4.3.2 修改 Gurobi 优化器输入

由于子回路消除约束（约束 12），如果车辆 traverses 子集 S中的一条必需边，那么它还会 traverses 进入和离开 S的边。对于由节点集 N/N_d形成的所有可能子集，情况都是如此。如果一条必需边的一个或两个节点是车场节点，就会出现问题（图 7a）。在这种情况下，不存在任何可能的子集，因为子集不包含车场节点（N_d）。为避免这种情况，我们添加了虚拟节点和权重为 0 的边，以修改包含此类必需边的任何实例。例如，在图 7b中，必需边（1，3）的节点 1 是车场节点，因此插入了一个虚拟节点 9 和一条虚拟边（1，9），其边权重为 0；同时，必需边（1，3）变为（9，3），且边权重保持不变。请注意，插入虚拟节点和边不会影响所生成路线的质量。图 7a 还展示了一个类似示例，其中必需边（6，7）的两个节点都是车场节点。

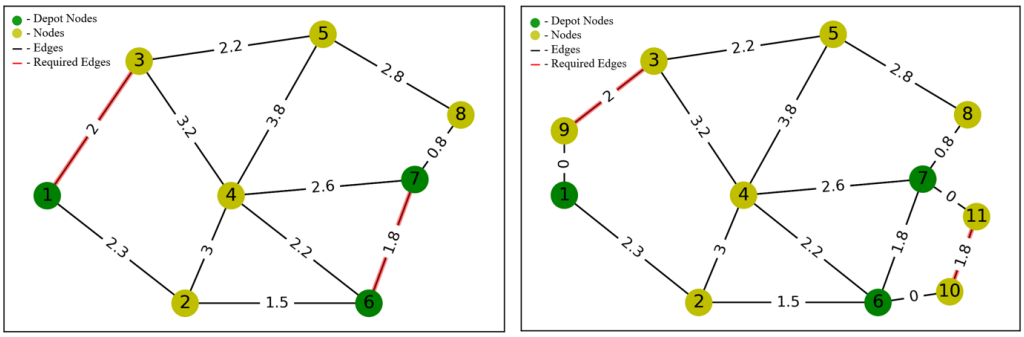


图 7：(a) 一个 MD-RPP-RV 实例，其中必需边的一端或两端为仓库节点；(b) 带有虚拟节点和边的修改后 MD-RPP-RV 实例。

4.4 结果

本节展示了我们在 A、B 和 C 集合中的实例上，对所提出的多趟算法、现有的基于启发式的方法，以及使用 Gurobi 求解混合整数线性规划所进行测试的结果。这些测试在配备 128 个物理核心、128 个逻辑处理器和 8 个 CPU 核心的 AMD EPYC 7763 64 核处理器上运行，最多使用 32 个线程，每个 CPU 核心配备 8 GB 内存。

4.4.1 基准实例结果

表2 列出了所用缩写，表3和4分别介绍了基准实例并给出了所得结果。

表 2：缩写

缩写	全称
PS	路径扫描算法
CS	构造-打击算法
AM	增广合并算法
MT	多趟算法
tu	时间单位
M	最大行程时间
ET	执行时间

表3：A 组实例信息（N =节点数， E =边数， $E_u =$ 需求边数， $K =$ 车辆数， $C =$ 车辆容量， $N_d =$ 仓库数）

实例 Name	N	E	E_u	C	K	N_d
A.1	8	11	3	18	1	2
A.2	11	19	5	40	2	3
A.3	7	21	8	16	4	2
A.4	7	21	8	12	4	2
A.5	12	22	7	40	3	3
A.6	11	22	8	18	4	3
A.7	12	22	8	40	4	3
A.8	12	22	7	44	3	3
A.9	12	22	8	40	4	3
A.10	13	23	7	60	3	3
A.11	12	25	9	38	4	3
A.12	12	26	9	40	4	3
A.13	13	26	7	44	3	3
A.14	10	28	9	198	4	3
A.15	8	28	10	16	5	2
A.16	8	28	10	14	5	2
A.17	11	33	11	18	5	3
A.18	9	36	13	16	6	2
A.19	11	44	14	18	7	3
A.20	22	45	16	38	8	5
A.21	27	46	13	18	6	6
A.22	27	51	16	18	8	6
A.23	11	55	18	16	9	3

表4：基准实例结果 s

实例 Name	PS结果 ET	PS结果 M	CS结果 ET	CS结果 M	AM结果 ET	AM结果 M	MT结果 ET	MT结果 M	混合整数线性规划结果 ET	混合整数线性规划结果 M	间隙百分比
A.1	0.0	214	–	–	0.0	236	0.0	64	0.1	64	0.0
A.2	0.0	264	0	264	0.0	264	0.0	244	117.1	148	64.9
A.3	0.0	113	–	–	0.0	113	0.0	16	6.7	15	6.7
A.4	0.0	100	0	197	0.0	102	0.0	32	166.4	8	300.0
A.5	0.0	233	0	273	0.0	297	0.0	136	3255.8	126	7.9
A.6	0.0	115	–	–	0.0	115	0.0	138	76.6	16	281.3
A.7	0.0	148	–	–	0.0	167	0.0	163	339.6	34	305.9
A.8	0.0	257	–	–	0.0	314	0.0	149	75.9	42	288.1
A.9	0.0	263	–	–	0.0	263	0.0	61	2788.1	134	11.2
A.10	–	–	–	–	–	–	0.0	361	211.3	59	511.9
A.11	0.0	244	–	–	0.0	285	0.0	121	65486.5	120	0.8
A.12	0.0	245	–	–	0.0	245	0.0	148	20170.9	110	34.5
A.13	–	–	–	–	–	–	0.0	166	22853.3	141	17.7
A.14	0.0	255	0	350	0.0	337	0.0	63	434.4	48	31.3
A.15	0.0	110	–	–	0.0	114	0.0	16	273.2	13	23.1
A.16	0.0	104	–	–	0.0	110	0.0	13	547.1	10	30.0
A.17	0.0	201	–	–	0.0	209	0.0	52	*	*	–
A.18	0.0	203	–	–	0.0	226	0.0	16	55548.1	13	23.1
A.19	0.0	106	0	113	0.0	118	0.1	48	*	*	–
A.20	0.0	146	–	–	0.0	154	0.1	31	*	*	–
A.22	0.0	208	–	–	0.0	219	0.0	66	*	*	–
A.22	0.0	436	–	–	0.0	436	0.1	103	*	*	–
A.23	0.0	106	–	–	0.0	115	0.1	44	*	*	–

在表 4中，未能完全求解某个实例的算法以单个短划线表示（–）。对于集合A中的一些小型实例，部分算法的执行时间不足50毫秒；由于执行时间四舍五入到小数点后一位，这些结果在表 *4*中显示为0。0。对于集合A中的部分实例，Gurobi 优化器在运行超过30小时后因分配的内存耗尽，无法使用MILP公式求得最优解。这些实例的结果在表中以 * 表示。

为衡量多趟算法构造的解相对于通过 Gurobi 求解混合整数线性规划得到的最优解的质量，我们按如下方式计算相对差异。设 M_{MT} 为多趟算法所得解的最大行程时间，设 M_{MILP} 为混合整数线性规划最优解的最大行程时间。指标“间隙”表示这两个值之间的相对差异百分比：

$$\text{Gap} = \frac{M_{MT} - M_{MILP}}{M_{MILP}} \times 100 \tag{17}$$

仅考虑我们为 A 组实例求得最优解的实例时，平均差距为 114 %。尽管 MT算法的解与最优解之间的质量偏差较大，但这种权衡是为了获得最优解而付出的大量执行时间。例如，对于表4中的集合A实例 A.13，MT算法在一秒内就解决了该实例，而 Gurobi 优化器则耗时超过 6 小时才获得最优解，且间隙为 17.7 %。这凸显了MT算法快速生成路线的能力十分重要。除了实例 A.6 和 A.7 外，MT算法在所有集合A实例中都获得了比文献中的所有基于启发式的方法更好的解。下一小节将介绍真实世界实例的结果。

4.4.2 真实世界实例结果

表5 和 6 介绍了真实世界实例及其结果。由于这些实例的节点和边数量远多于基准实例，我们无法使用 Gurobi 求解混合整数线性规划并获得最优解，因为这需要大量的计算资源和内存。

表5：B组和C组实例信息（N =节点数， E =边数， E_u = 需求边数， K = 车辆数， C = 车辆容量， N_d = 仓库数）

实例 Name	N	E	E_u	C	K	N_d
B.1	75	130	39	31	19	16
B.2	133	214	28	31	14	27
B.3	222	344	115	31	57	45
B.4	192	353	113	31	56	39
B.5	290	423	44	31	22	59
B.6	288	494	163	31	81	58
B.7	374	622	85	31	42	75
B.8	461	879	286	31	143	93
C.1	75	260	39	31	19	16
C.2	133	428	28	31	14	27
C.3	222	688	115	31	57	45
C.4	192	706	113	31	56	39
C.5	290	846	26	31	13	59
C.6	288	988	163	31	81	58
C.7	374	1244	81	31	40	75
C.8	461	1758	286	31	143	93

表6：B组和C组实例结果（B.1-B.8 = B组实例，C.1-C.8 = C组实例，ET =执行时间（秒），M = 最大行程时间（分钟））

	PS结果		CS结果		AM结果		MT结果	
实例 Name	ET	M	ET	M	ET	M	ET	M
B.1	2	216	–	–	2	203	3.2	28.7
B.2	–	–	–	–	–	–	0.4	356.7
B.3	60	194	–	–	55	124	292.3	30.8
B.4	–	–	–	–	–	–	29.0	127.2
B.5	–	–	–	–	–	–	1.2	563.0
B.6	107	223	–	–	103	223	313.4	130.3
B.7	–	–	–	–	–	–	4.3	593.6
B.8	158	227	–	–	174	227	224.3	122.6
C.1	8	214	–	–	4	123	5.6	25.2
C.2	–	–	–	–	–	–	0.8	578.2
C.3	–	–	–	–	164	207	492.3	28.6
C.4	–	–	–	–	–	–	41.8	135.8
C.5	0	241	–	–	–	–	0.7	808.1
C.6	360	246	–	–	329	246	447.3	30.7
C.7	1	656	–	–	–	–	3.4	449.3
C.8	398	228	–	–	588	228	319.7	122.2

表6中，未能完全求解某个实例的算法以单个短横线表示（–）。所提出的多趟算法求解了所有真实世界

实例，并构造出优于CARP启发式方法的解。所有CARP启发式方法合计只能求解16个真实世界实例中的10个。平均而言，多趟算法能在3分钟（136.2秒）内求解真实世界实例（这些实例包含75至461个节点和130至1758条边）。

4.5 计算工作量分析

对理论计算复杂度的分析有助于更好地理解算法的效率和可扩展性。在这方面，多趟算法的理论上界和下界对于评估其性能及其适用于各种现实应用场景的程度至关重要。

上界（O）根据输入估算算法在最坏情况下的运行时间（见表1）。这些信息对于确定算法的可扩展性，以及识别算法可能无法满足预期性能要求的场景至关重要。另一方面，下界（Ω）通过分析最佳情况输入，提供求解问题所需最少操作次数的理论保证。这些信息有助于评估算法固有的复杂度，并发现优化机会。

命题1 *MT*算法的理论上界为 $O(|E|^2|N|^2)$，其中 $|N|$ 和 $|E|$ 分别为MD-RPP-RV实例的节点数和边数。

命题 2*MT*算法的理论下界为 $\Omega(K)$，其中 K 表示车辆数。

命题 1 和命题 2 的证明见附录。

5 结论

本文将 MD-RPP-RV 问题定义并表述为一个混合整数线性规划，并提出了一种新的启发式方法——多趟算法。该算法是在研究并深入了解为解决 CARP 而开发的基于启发式的方法的不足之处后提出的。

为了测试多趟算法并将其与基于启发式的方法进行比较，我们根据文献和现实道路地图创建了实例。此外，我们还创建了模拟风对车辆行驶时间影响的实例，这与启发本研究的冬季天气情景相关。

我们通过使用 Gurobi 求解 MILP，获得了较小实例的最优解；但由于内存和计算能力限制，无法为较大的真实世界实例找到最优解。结果表明，对于所有实例，所提出的多趟算法都比基于启发式的方法得出了质量更高的解。多趟算法所得解的质量与最优解质量之间的平均差距为 114%。多趟算法平均在 3 分钟内求解了较大的真实世界实例。

关于未来研究方向，目前存在许多被隐含考虑的假设，例如运行期间不会发生车辆故障，而这一假设在某些情况下可能并不现实。一个可能的未来研究方向是研究存在多次车辆故障的问题，并开发能够构建鲁棒解的方法。另一个可能的未来研究方向是对温度对车辆电池的影响进行建模，并将其与多趟算法相结合，以产生更符合实际的结果。设施选址问题（确定仓库的最佳位置，以最小化所需的总时间）同样值得研究。

致谢

本研究由马里兰大学的马里兰州工业合作伙伴关系（MIPS）项目和 SpringGem 资助。第一作者的工作部分得到了 Alex Mehr博士（Ph.D. ’03）通过设计决策支持实验室研究与教育基金提供的慷慨捐赠，以及陆军合作协议 W911NF2120076 的支持。我们谨此感谢 SpringGem 的 Menglin Jin博士提供的帮助与支持。我们还要感谢 Michael Otte博士为本研究提供的信息和反馈。作者感谢马里兰大学提供的高性能计算资源（Zaratan），这些资源用于开展本文所报告的研究。

参考文献

[1] Gurobi Optimization, L..（2023）。Gurobi Optimizer参考手册。[2] Eiselt, H. A., Gendreau, M., & Laporte, G.（1995）。弧路径问题，第一部分：中国邮差问题。《运筹学》，43(2)，231-242。[3] Eiselt, H. A., Gendreau, M., & Laporte, G.（1995）。弧路径问题，第二部分：乡村邮差问题。《运筹学》，43(3)，399-414。[4] Edmonds, J., & Johnson, E. L.（1973）。匹配、欧拉回路与中国邮差。《数学规划》，5，88-124。[5] Lenstra, J. K., & Kan, A. R.（1976）。关于一般路径规划问题。《Networks》，6(3)，273-280。[6] Golden, B. L., and Wong, R. T.（1981）。容量约束弧路径问题。《*Net- works*》，*11*(3)，305-315。[7] Golden, B. L., DeArmon, J. S., and Baker, E. K.（1983）。针对一类路径规划问题的算法计算实验。《计算机与运筹学》，*10*(1)，47-59。[8] Pearn, W. L.（1989）。容量约束弧路径问题的近似解。《计算机与运筹学》，*16*(6)，589-600。[9] Christofides, N.（1973）。图的最优遍历。*Omega*，*1*(6)，719-732。

[10] Fern´andez, E., & Rodr´ıguez-Pereira, J.（2017）。多仓库乡村邮差问题。《TOP》，25(2)，340-372。[11] Tirkolaee, E. B., Hosseinabadi, A. A. R., Soltani, M., Sangaiah, A. K., & Wang, J.（2018）。城市服务领域中多趟绿色容量弧路径问题的混合遗传算法。《可持续性》，10(5)，1366。[12] Amberg, A., Domschke, W., & Voß, S.（2000）。多中心容量弧路径问题：一种使用容量树的禁忌搜索算法。《欧洲运筹学杂志》，124(2)，360-376。[13] Hu, H., Liu, T., Zhao, N., Zhou, Y., & Min, D.（2013）。一种结合扰动的多仓库容量弧路径问题混合遗传算法。《应用科学杂志》，13(16)，3239-3244。[14] Kansou, A., & Yassine, A.（2010）。多仓库容量弧路径问题的新上界。《国际元启发式杂志》，1(1)，81-95。[15] Chen, H., Cheng, T., & Shawe-Taylor, J.（2018）。最小最大多仓库乡村邮差问题（mmmdrpp）的均衡路线设计：警务巡逻案例。《国际地理信息科学杂志》，32(1)，169-190。[16] Frederickson, G. N., Hecht, M. S., & Kim, C. E.（1976年10月）。若干路径问题的近似算法。载于《第17届计算机科学基础年会（SFCS 1976）》（第216-227页）。IEEE。[17] Ahr, D., & Reinelt, G.（2002）。最小最大 k 中国邮差问题的新启发式算法和下界。载于《算法——ESA 2002：第10届欧洲算法研讨会，意大利罗马，2002年9月17日至21日，论文集10》（第64-74页）。Springer Berlin Heidelberg。[18] Brand˜ao, J., and Eglese, R.（2008）。容量约束弧路径问题的确定性禁忌搜索算法。《计算机与运筹学》，*35*(4)，1112-1126。[19] Lacomme, P., Prins, C., and Sevaux, M.（2006）。双目标容量约束弧路径问题的遗传算法。《计算机与运筹学》，*33*(12)，3473-3493。[20] Dorigo, M., Birattari, M., and Stutzle, T.（2006）。蚁群优化。《*IEEE*计算智能杂志》，*1*(4)，28-39。[21] Tirkolaee, E. B., Alinaghian, M., Hosseinabadi, A. A. R., Sasi, M. B., and Sanga- iah, A. K.（2019）。多趟容量弧路径问题的改进蚁群优化。《计算机与电气工程》，*77*，457-470。[22] Polacek, M., Doerner, K. F., Hartl, R. F., and Maniezzo, V.（2008）。具有中间设施的容量约束弧路径问题的变邻域搜索。《启发式算法杂志》，*14*(5)，405-423。[23] Hertz, A., and Mittaz, M.（2001）。无向容量弧路径问题的变邻域下降算法。《交通科学》，*35*(4)，425-434

[24] Hertz, A., Laporte, G., 和 Mittaz, M. (2000). 用于容量约束弧路径问题的禁忌搜索启发式算法。《运筹学》，*48*(1), 129-135.[25] Lacomme, P., Prins, C., 和 Ramdane-Ch´erif, W. (2001). 用于容量约束弧路径问题及其扩展问题的遗传算法。载于进化计算应用研讨会(pp. 473-483). Springer, 柏林, 海德堡.[26] Belenguer, J. M., 和 Benavent, E. (2003). 用于容量约束弧路径问题的割平面算法。《计算机与运筹学》，*30*(5), 705-728.[27] Usberti, F. L., Fran¸ca, P. M., 和 Fran¸ca, A. L. M. (2012). 用于弧路径问题的分支定界算法。*SBPO* 第*44*届年刊，里约热内卢, 652.[28] Corber´an, A., Plana, I., 和 Sanchis, J. M. (2007). 用于有向非对称一般路径问题及其特例的分支割平面算法。《网络：国际期刊》，*49*(4), 245-257.[29] Bekta¸s, T., Erdo˘gan, G., 和 Røpke, S. (2011). 广义车辆路径问题的数学建模及分支切割算法。《交通科学》，*45*(3), 299-316.[30] Land, A.H., Doig, A.G. (2010). 求解离散规划问题的一种自动方法。载于：等人编，《整数规划50年：1958—2008》。Springer, 柏林, 海德堡。https://doi.org/10.1007/978-3-540-68279-0 5[31] Semsch, E., Jakob, M., Pavlicek, D., & Pechoucek, M. (2009). 复杂城市环境中的自主车辆监视。载于2009年 IEEE/WIC/ACM 网络智能与智能代理技术国际联合会议（第2卷，pp. 82-85）。IEEE.[32] Puri, A. (2005). 用于交通监视的无人机调查。南佛罗里达大学计算机科学与工程系，1-29.[33] Lee, K. S., Ovinis, M., Nagarajan, T., Seulin, R., & Morel, O. (2015). 使用无人机的自主巡逻与监视系统。载于2015年 IEEE 第15届环境与电气工程国际会议（EEEIC）（pp. 1291-1297）。IEEE.[34] Sundar, K., & Rathinam, S. (2013). 存在加油站时无人机路径规划的算法。《IEEE 自动化科学与工程汇刊》，11(1), 287-294.[35] Yu, K., Budhiraja, A. K., & Tokekar, P. (2018). 配备移动充电站的无人机路径规划算法。载于2018年 IEEE 国际机器人与自动化会议（ICRA）（pp. 5720-5725）。IEEE.[36] Choi, Y., Robertson, B., Choi, Y., & Mavris, D. (2019). 面向基于小型无人飞行器系统的城市配送的多趟车辆路径问题。《航空器杂志》，56(6), 2309-2323。

[37] Dorling, K., Heinrichs, J., Messier, G. G., & Magierowski, S. (2016). 无人机配送的车辆路径问题。IEEE《系统、人和控制论汇刊：系统》，47(1), 70-85.[38] Choi, Y., Choi, Y., Briceno, S. I., & Mavris, D. N. (2019). 一种针对基于无人机系统的配送系统的扩展节约算法。载于 AIAA SciTech 2019论坛（p. 1796）。[39] Dijkstra, E. W. (1959). 关于图的两个问题的注记。《数值数学》，1(1), 269-271. [40] Mavic 2 - 产品信息 - 大疆。大疆官方。（无日期）。检索于2022年，网址：https://www.dji.com/mavic-2/info。[41] Sathyamurthy, Eashwar, (2021). 多无人机弧路径问题的多航次算法。硕士论文，马里兰大学帕克分校，马里兰州。

附录

命题***1***的证明：

为了证明所提出的多趟算法的理论上界，考虑MD-RPP-RV的最坏情况实例，其中所有车辆都需要执行多行程路线以遍历必需边。在第3.3节中，使用所提出的多趟算法描述并求解了这样一个实例。

在最坏情况下，伪代码（第3.2节）中描述的多趟算法将一直处于while循环（第1行）中，直到遍历完所有必需边（E_u），所需的操作次数为 O（$|E_u|$）。在第2行的循环内部，查找最快的可用车辆最多需要 K步，即 O（$|E_u|\times$（K））。为了在第3行确定距车辆位置 n_k 最近的可行边，必须考虑所有 $|E_u|$必需边，以确定最近的可行边。由于多趟算法使用Dijkstra算法来确定这些距离，而其复杂度为 O（$|N|^2$），因此总体复杂度为 O（$|E_u|\times$（$K+|E_u|\times|N|^2$）。

由于不存在可行的单程路径，算法随后进入第 14 行的 else 语句。为确定集合 E_u 中最近的必需边（第 15 行），算法再次使用 Dijkstra算法 执行$|N|^2|E_u|$ 步，即 $O(|E_u|\times(K+2\times|E_u|\times|N|^2))$。第 16 行中，确定最近的可行仓库与确定最近的可行边类似。因此，需要考虑每个仓库节点，以确定距离 n_k 最近的仓库节点，其复杂度为 $O(|N_d|\times|N|^2)$，使算法的总复杂度达到 $O(|E_u|\times($ $K+2\times|E_u|\times|N|^2+|N_d|\times|N|^2))$。最后，在第 18 行，确定到 d^* 的最短路径需要 $|N|^2$，$O(|E_u|\times(K+2\times|E_u|\times|N|^2+|N_d|\times|N|^2+|N|^2))$。算法退出 while循环，但还需通过寻找最近的仓库（第 28 行）来完成车辆路径（第 26 行），此时算法的总复杂度变为 $O(|E_u|\times(K+2\times|E_u|\times|N|^2+|N_d|\times|N|^2+|N|^2)+K\times|N_d|\times|N|^2)$。

但是，根据用于测试的实例生成方式（第 4.2节），我们知道$|E_u|=\frac{|E|}{3}$，$K=\frac{|E_U|}{2}=\frac{|E|}{6}$ 以及 $|N_d|=\frac{|N|}{5}$。因此，我们可以表示上界

算法的上界可以用节点数和边数紧凑地表示。

$$O(|E_u| \times (K + 2 \times |E_u| \times |N|^2 + |N_d| \times |N|^2 + |N|^2) + K \times |N_d| \times |N|^2)$$
$$O(\frac{|E|}{3} \times (\frac{|E|}{6} + 2 \times \frac{|E|}{3} \times |N|^2 + \frac{|N|}{5} \times |N|^2 + |N|^2) + \frac{|E|}{6} \times \frac{|N|}{5} \times |N|^2)$$
$$O(\frac{|E|^2}{18} + 2 \times \frac{|E|^2}{9} \times |N|^2 + \frac{|N|^3}{5} \times \frac{|E|}{3} + |N|^2 \times \frac{|E|}{3} + \frac{|E|}{6} \times \frac{|N|^3}{5})$$

去掉低阶项和常数项后，上界可简化为：

$$O(|E|^2 \times |N|^2 + |N|^3 \times |E|)$$

由于大多数实例都是多重图，节点之间存在多条边，$|E| \geq |N|$。因此，
$|E|^2 \times |N|^2 \geq |N|^3 \times |E|$。由此，多趟算法的上界为：

$$O(|E|^2|N|^2)$$

命题 ***2*** 的证明：

为了证明理论下界，我们考虑最佳情况，即不存在需要遍历的必需边。在这种情况下，多趟算法直接进入第 26 行，并遍历所有车辆以检查是否存在未完成路线。该循环需要 K 步，因此，多趟算法的下界为：

$$\Omega(K)$$