基于改进遗传算法的移动储能车优化调度模型

摘要

随着城市建设和经济的高速发展，原有的城区变电所供电容量常常无法满足负荷高峰期的用电需求。通过使用移动储能车配合电网进行削峰填谷，完成重要单位保电任务，可以改善电能质量，提升配电用户满意度。根据用户用电需求、城市交通流量和电网电价规划移动储能车行驶路径优化与充放电策略具有非常重要的意义。

本文根据题目和附件所提供的信息，对问题建模并设计关键决策变量，通过分析证明约减变量解空间，使用遗传算法和邻域搜索算法，寻找具体问题的最佳方案。

针对问题一：第一，我们首先对数据进行分析，主要分析用电缺口多大，充放电价格有什么特征。第二，结合数据分析结论和 9 个关键问题，提出新的约束条件，缩小解空间。第三，对车辆行驶方案建模，最要包括车辆调度模型和路径选择模型。第四，根据题目设计新算子，改进遗传算法。第五，设计解码器，将改进遗传算法与车辆模型结合。第六，分析结果，提出改进方案。

针对问题二：首先，针对光伏发电的价格数据进行分析，提出移动储能车的充电方案。分别基于（1）满足三类用户需求；（2）满足一、二类用户用电需求的方案分析用车数量。确定出满足三类用户需求和满足一、二类用户用电需求的最少使用车辆数与型号。随后，根据车辆位置和台区变的用电负荷数据规划出移动储能车的安置点和移动方向。结合光伏发电功率趋势和光伏充电电价，建立起相应的车辆调度方案，将车辆调度问题转化为求解获得最大收益的移动车放电功率和移动方向问题。最后，使用改进编码方式的遗传算法求解移动车每一时刻的放电功率，以及 5 号车前往的起始台区变编号，分别得到（1）满足三类用户需求，（2）满足一、二类用户需求的优化解决方案及最终收益。

针对问题三：针对不确定旅行及服务时间下的车辆调度问题，本文首先建立了充电站排队系统的仿真模型，利用离散事件仿真的方法对于储能车的等待时间和充电时间进行了求解；其次，对于台区变负荷需求和道路拥堵情况的概率分布，利用采样的方法生成了每个时间段的台区变负荷需求和拥堵系数；最后，沿用问题一的算法框架，基于遗传算法对于问题求解。初步结果表明，结合仿真和采样的遗传算法能够有效地解决不确定旅行及服务时间下的车辆调度问题。

最后，我们对提出的模型进行全面的评价：本文的模型贴合实际，能合理解决提出的问题，具有实用性强，算法效率高等特点，本文设计算法框架适用于车辆调度优化问题、考虑利润的旅行商问题以及不确定旅行及服务时间下的车辆调度问题。

关键词：移动储能车调度　遗传算法　邻域搜索算法　排队论　不确定因素建模

目录

1 问题综述

1.1 问题背景

移动储能车行驶路径优化与充放电策略主要根据用户用电需求、城市交通流量和电网电价进行灵活规划。不同的路径选择方案会对应不同的行驶成本与充放电方案，最终影响充放电服务收益。需要根据未来 24 小时用电负荷预测功率，同时考虑负荷峰谷差和配电网运行成本，需要给出移动储能车充放电运行策略。相关策略需满足以下条件：

1）一天为一个调度周期，每小时为一个时间间隔。

2）移动储能发车时间为早上 7 点，且回到储能车集合存放点的时间不能超过晚上 22 点。

3）移动储能车每天出发前电池满电量，选择合适路线前往台区变压器

4）移动储能车进入台区后以不超过额定功率进行放电，放电功率不能超过台区总用电需求。

5）规定同一时段内，同一个台区变仅允许接入一台移动储能车。

6）储能车电量消耗完毕或完成当天任务后，需行驶至快速充电站进行电池充电。

7）同一个充电站同一时段只允许一台车充电，同一时段有其它车时则按先到先服务原则充电。

8）移动储能车仅能停留在快速充电站和台区变所在节点，其余节点不能停留。

9)节点 1 为移动储能车集合存放点，节点 6、13、18、22 分别接入四座快速充电站，其中城区节点 5、14、19、27 各装有一台台区变。

1.2 问题分析

问题一：已知未来 24 小时分时电价的相关数据。拥有的 50kW/200kWh(MV#1)、100kW/400kWh(MV#2)各有 5 辆移动储能车，储能车驾驶时速 30kM/h。MV#1 的行驶成本为 1 元/公里，MV#2 的行驶成本为 2 元/公里。充电站的快速充电功率为 400kW。移动储能车每天仅允许完整充放电一次。试根据台区变负荷需求，按照客户优先级确保满足各类用户的用电需求并使得移动储能车总充放电服务收益最大，建立移动储能车调度方案（行驶路径、调度时间和充电时间等）的数学模型并求解。

该问题涉及多个方面。从储能车出发，到最终返回车场停放整个流程，需要重点考虑以下几个问题：

(1) 问题 1: 不同储能车选择选择哪个或者哪几个台区变放电？
(2) 问题 2: 储能车选择哪条路径前往台区变？
(3) 问题 3: 移动储能车补充的发电后电网是否能够满足各类用户需求？特别是完全满足第三类用户需求是否会减少总收益？
(4) 问题 4：储能车放电顺序，不同储能车放电过程中间是否能有间隔，是否交替放电？不同车在不同时间段在不同台区变以什么功率放电？
(5) 问题 5: 储能车是否需要全部放完电后再去充电？
(6) 问题 6：储能车选择哪个充电站充电？
(7) 问题 7：储能车选择哪条路径前往充电站？
(8) 问题 8：储能车到充电站后是否立即充电，什么时间充电？
(9) 问题 9：储能车是否能够按时返回车场？

将上述问题可以分为两类问题。

第一类问题是确定行驶路径问题。该问题可以分解成两个步骤求解。第一步确定 4 个台区变，4 个变电站，以及停车场之间最短路径。该问题是典型的有权图中最短路径问题，且节点数量较少，可以使用迪杰斯特拉算法(Dijkstra)求最优解。

第二类问题是在规划车辆放电策略，在按时回车场，满足用电缺口情况下，追求利润最大化。是典型多目标优化问题，且解空间较大。我们首先对题目进行分析，增加新的约束条件，缩减解的规模。第二，建立车辆状态模型和调度方案收益模型。第三，创新设计新的编码方式，根据题目实际调整改进遗传算法，提高收敛速度和解的质量。第四，确定解码方式，将调度方案收益模型和改进后的遗传算法结合。

问题二：假定车辆一天内可以多次充放电，目前是在派出移动储能车最少的情形下，收益最大化。通过数据分析，我们首先确定在只保障第一第二类用户需求时，只需要派出 3 辆大功率储能车，在保障所有用户用电需求时，需要派出 4 辆大功率储能车。其次我们调整了遗传算法编码，最终得到规划方案。

2 模型假设与符号说明

2.1 模型基本假设

(1) 移动储能车每天开始时从车辆集合存放点发车，一天结束后，移动储能车需驶回原有车辆集合存放点进行维护管理。

(2) 规定同一时段内，同一个台区变仅允许接入一台移动储能车，同一个充电站同一时段只允许一台车充电。

(3) 每台移动储能车在一天调度结束后电量必须为满电状态，为下一个调度周期做准备。

(4) 移动储能车仅能停留在快速充电站和台区变所在节点，其余节点不能停留。

(5) 一类、二类用户突然中断供电影响较大，假设无论如何都要确保一类、二类用户用电。在充裕电力时要充分保障三类用户用电。

(6) 只考虑移动储能车储能运行约束，不考虑储能设备充放电行为对电力系统潮流分布的影响。同时忽略储能电池充放电行为对其寿命损耗造成的影响。

(7) 移动储能车的储油量满足一天的出行需求。

(8) 在第一第二问，假定移动储能车驾驶时速始终为 30kM/h。

2.2 符号说明

表 1 符号说明

符号	含义	注释
N	台区变节点	取值为 5,14,19,27
M	充电站节点	取值为 6,13,18,22
V	储能车的行驶速度	第一、第二问 30km/h
T_k	车辆 k 七点出发后时间	
P_M	充电站的快速充电功率	恒为 400kW
P_{1e}	MV#1 储能车输出功率	最大为 50kW
P_{2e}	MV#2 储能车输出功率	最大为 100kW
W_{kz}	储能车 k 的电池容量	MV#1 为 200kW/h

符号	含义	注释
		MV#2 为 400kW/h
a_{Nk}	车 k 在台区变节点 N 的放电开始时间	
b_{Nk}	车 k 在台区变节点 N 的放点结束时间	
c_{Mk}	车 k 在充电站节点 M 的充电开始时间	
d_{Mk}	车 k 在充电站节点 M 的充电结束时间	
p_{Nt}	台区变节点 N 在 t 时刻的缺口负荷(kW)	
$p_{\mathrm{Nk}t}$	车辆 k 在台区变 N 服务 t 时刻的放电功率	
W_{kt}	车辆 k，t 时刻剩余电量(kwh)	
x_{ijk}	车辆 k 从节点 i 到节点 j 时为 1，否则为 0	
R_{Nt}	t 时刻的放电价格（¥/kWh）	
R_{Mt}	t 时刻的充电价格（¥/kWh）	
C_k	车辆 k 在路上消耗的总成本	
t_{Nk}	车辆 k 在台区变 N 的服务时间段列表	
t_{Mk}	辆 k 在充电站 M 的服务时间段列表	
R_{Nk}	车辆 k 在台区变 N 的服务时间段的放电价格列表	
R_{Mk}	车辆 k 在充电站 M 的服务时间段的充电价格列表	
p_{Nk}	车辆 k 在台区变 N 的服务时间段的放电功率列表	

3 问题一分析与模型建立

3.1 问题分析

若假定车辆在台区变任意时刻能以任意功率放电，并可以移动到多个台区变放电，问题解空间太大。通过分析题目，我们提取利润最大化的目标，以及按时充满电返回车场，保障用电缺口的两个关键约束。结合题目条件，将这个规划方案细分为以下 9 个问题。

(1) 问题 1: 不同储能车选择选择哪个或者哪几个台区变放电？

(2) 问题 2: 储能车选择哪条路径前往台区变？

(3) 问题 3: 移动储能车补充的发电后电网是否能够满足各类用户需求？特别是完全满足第三类用户需求是否会减少总收益？

(4) 问题 4：储能车放电顺序，不同储能车放电过程中间是否能有间隔，是否交替放电？不同车在不同时间段在不同台区变以什么功率放电？

(5) 问题 5: 储能车是否需要全部放完电后再去充电？

(6) 问题 6：储能车选择哪个充电站充电？

(7) 问题 7：储能车选择哪条路径前往充电站？

(8) 问题 8：储能车到充电站后是否立即充电，什么时间充电？

(9) 问题 9：储能车是否能够按时返回车场？

在回答 9 个问题前，**第一**，我们首先对数据进行分析，主要分析用电缺口多大，充放电价格有什么特征。**第二**，结合数据分析结论和 9 个关键问题，提出新的约束条件，缩小解空间。**第三**，对车辆行驶方案建模，最要包括车辆调度模型和路径选择模型。**第四**，根据题目设计新算子，改进遗传算法。**第五，**设计解码器，将改进遗传算法与车辆模型结合。**第六，**分析结果，提出改进方案。

3.2 数据分析

3.2.1 用电缺口分析

为了保障各类用户用电，我们首先需要计算各个时间段电力缺口。根据题目中，各台区变各时刻负荷负载率不超过 1，额定功率（MW）=台区变额定容量 S（MVA）×功率因素，功率因素取 0.8。可以算出每个台区变的额定功率$P_{额}$，记附件 1 中每个台区变在各个时刻的负荷为P_t，计算台区变的负荷需求$P_t - P_{额}$，用p_{Nt}表示。当$p_{Nt} > 0$时，代表台区变节点 N 在 t 时刻电力缺口。我们计算了各个台区电缺电情况及相关统计信息。

表 2 各个时段台区变电量缺口

用电缺口时段\台区变	#1 台区	#2 台区	#4 台区	#3 台区	总缺电情况
9:00-10:00	0.04	0	0	0	0.04
10:00-11:00	**0.073**	0	0	0	0.073
11:00-12:00	**0.074**	0	0	0	0.074
12:00-13:00	**0.078**	0.045	0.032	**0.085**	0.24
13:00-14:00	**0.069**	0.042	0.032	**0.07**	0.213

14:00-15:00	**0.059**	0.05	0	**0.072**	0.181
15:00-16:00	0.044	0.044	0.037	0	0.125
16:00-17:00	0.0474	**0.085**	0.037	0.036	0.2054
17:00-18:00	0.0329	**0.08**	0.022	0.05	0.1849
18:00-19:00	0.04	**0.072**	0.028	0.05	0.19
19:00-20:00	0	**0.082**	0.032	0.04	0.154
总和	0.5573	0.5	0.22	0.403	1.6803

表 3 各个台区电电力缺口统计信息

	#1 台区	**#2 台区**	**#4 台区**	**#3 台区**
各个台区总缺电量	0.5573	0.5	0.276	0.403
各个台区变缺电时段数量	10	8	7	7
各个台区总缺电量/缺电时间段	0.05573	0.0625	0.0394	0.0575
各个台区一定需要高功率供电车的时间段数量	5	4	0	3
各个台区一定需要高功率供电车的发电量	0.353	0.319	0	0.227

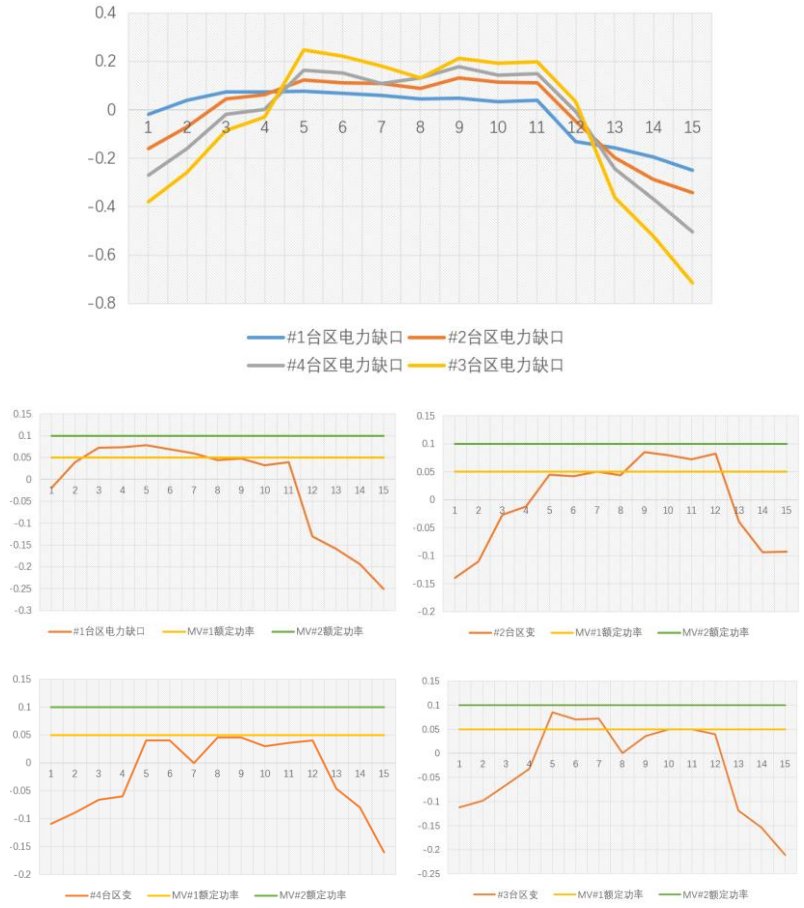

图 1 各时段各个台区变缺电情况变化图

1.缺电总量分析。总缺电量 1.680MWh，移动储能车的满载电池容量为 5*（200kWh+400kWh）=3MWh，总缺电量是车辆总发电量的 56%。

2.缺电的时间分布分析。四个台区变共有 10+8+7+7=32 个时间段缺电。假定车辆满功率放电情况下，能够提供(200/50+400/100)*5=40 个时间段电力。缺电时间段是发电时间段的 80%。

3.一定需要 MV#2 大功率车情况分析。因为一个台区变只允许最多一辆车供电，当供电缺口大于小功率车额定功率时，为了保障用户用电需求一定需要大功率供电，因此需要特别分析这类情况。需要高功率车总的发电量是 0.899MWh。大功率车所带负荷是 0.4*5=2MWh。所需发电量是车辆负荷是 44.95%。其次，供电时长分析需要高功率车时间段共 5+4+3=12 个，高功率车满功率放电，能够服务时长（400/100）*5=20 小时。前者是后者的 60%。

4.分不同台区分析。台区 1 缺高功率电 0.353MWh，总缺电量 0.5573MWh，至少需要一辆高功率车和一辆低功率车,提供 0.4MWh 高功率电和 0.2MWh 低功率电。能够初步满足需求。台区 2 缺高功率电 0.319，总缺电量 0.5，至少需要一辆高功率车和一辆低功率车,提供 0.4MWh 高功率电和 0.2MWh 低功率电。台区 4 缺高功率电 0，总缺电量 0.276MWh，需要一辆高功率车，或者两辆低功率车供电，提供 0.4MWh 电。台区 3 缺高功率电 0.227MWh，总缺电量 0.403MWh，需要一辆高功率车和一辆低功率车供电，提供 0.4MWh 高功率电和 0.2MWh 低功率电。在尽可能少用高功率车时，空闲 2 辆高功率车。台区 4 可以用一辆高功率车替换 2 辆低功率车，此时剩余 1 辆高功率车，2 辆低功率车。

表 4 各个台区最少需要车辆（尽可能少用高功率车）

	#1 台区	**#2 台区**	**#4 台区**	**#3 台区**
MV#1	1	1	2	1
MV#2	1	1	0	1

4.充电情况分析。充电站充电功率为 400kW，200/400=0.5h 可以充满一辆低功率车，400/400=1h 时可以充满 1 辆高功率车。4 个充电站，充满 5 辆低功率车和 5 辆高功率车共需要 2h 左右。台区 1 最后 3 小时无用电缺口，其余 3 个台区最后 2 小时无用电缺口。停车场到各个台区变最远距离小于 20km，用时小于 0.6h。考虑有些车辆在 19 点前已经放完电，因此我们认为剩余 2h 时间是足够剩余车辆充电并回到车场的。

小结：通过对缺电数据和表格分析，我们得出以下三个结论。1.车辆无论是电量、发电功率还是数量都是富足的。2.时间是充足的。3.台区 1、台区 2、台区 3 需要配置一辆高功率车。

3.2.2 充放电价格分析

各个时段充放电价格如下图所示

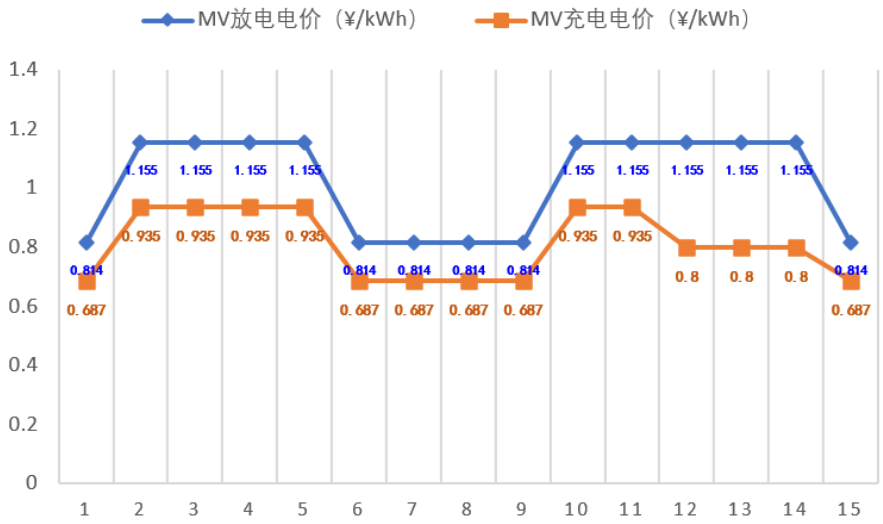

图 2 各时段电价变化图

通过对图标初步分析，结合之间结论，我们可以得到以下信息。

1.放电价格只有两种，分别是 0.814¥/kWh 和 1.115¥/kWh。

2.车辆最快放完电需要 4h，此时若马上充电电价为 0.935¥/kWh，在假定时间是充足情况下，我们认为可以等到 12 时之后再充电。

3.16 时到 18 时电价为 0.935¥/kWh，此时电价高于之前部分发电价格，在假定时间是充足情况下，我们认为可以等到 18 时之后再充电。

3.3 提出新假设

在对数据进行初步分析后，按照小车实际移动情况和决策，可以得出以下流程图。

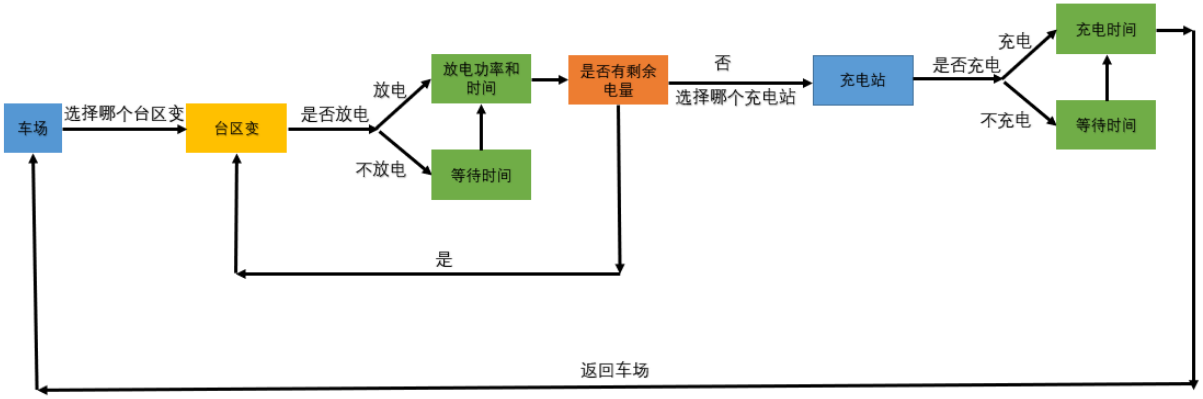

图 3 车辆行驶流程图

从中我们可以梳理出 9 个关键问题。在对数据有初步认知情况下，逐个分析每个问题，提出新约束条件。

3.3.1 不同储能车选择选择哪个或者哪几个台区变放电？

根据前期分析，从满足顾客需求角度出发，**我们认为台区变 1、2、3 各需要一辆大功率车。**由于车辆数量和时间充足，做出这样的假设不会导致算法有没有可行解，且让车辆充分放电可以增加收益。我们根据这样假设可以直接找出一个平凡解。即从移动储能车集合存放点发车后，将第一类客户所在的#3 台区停放 2 辆 MV#1 型移动储能车，

50kW 的额定功率能够满足 12:00-20:00 前超出台区用电负荷的需求；在第一类客户所在的#4 台区停放 1 辆 MV#2 移动储能车，100kW 的额定功率能够满足 12:00-15:00 前的放电服务需求，1 辆 MV#1 型移动储能车能够满足 15:00-20:00 的放电服务需求。第二类客户所在的#1 台区停放 1 辆 MV#2 移动储能车能够满足 09:00-15:00 前的放电服务需求，1 辆 MV#1 型移动储能车能够满足 15:00-20:00 的放电服务需求。

剩余车辆选择台区变需要作为决策变量，由后续算法决定。

车辆前往多个台区变优势：1.在高功率车有限时，可以满足高功率要求。2.可以前往其他没有车辆的台区变放电，减少等待时间，增加放电收益。缺电 1.可能会在路程上浪费时间和费用，导致车辆无法及时返回车场以及总收益下降。基于时间、车辆充足假设，我们认为高功率车是充足的，没有台区变空闲时间短。

因此我们认为前往多个台区变放电优势不明显，**假定车辆只能选择一个台区变。**

3.3.2 储能车选择哪条路径前往台区变？

该问题是典型的有权图中最短路径问题，且节点数量较少，可以**使用迪杰斯特拉算法(Dijkstra)求最优解。**

迪杰斯特拉算法采用贪心算法的策略，将所有顶点分为已标记点和未标记点两个集合，从起始点开始，不断在未标记点中寻找距离起始点路径最短的顶点，并将其标记，直到所有顶点都被标记为止。

3.3.3 移动储能车补充的发电后电网是否能够满足各类用户需求？特别是完全满足第三类用户需求是否会减少总收益？

首先，计算#1-#4 台区变的高峰期供电缺口。根据换算公式：额定功率（MW）=台区变额定容量 S（MVA），功率因素=0.8，得到 24 小时每时刻下#1-#4 台区变的负荷与额定功率。由于台区变电站负载率不超过 1，因此#1-#4 台区变均存在高峰期供电缺口。

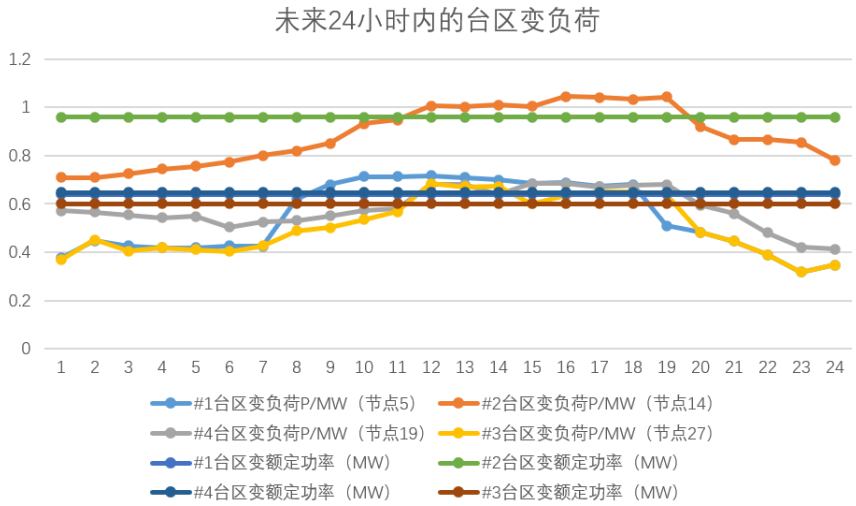

图 4：#1-#4 台区变负荷与其额定容量

第二，对移动储能车全部电量进行计算，初步判断是否满足高峰期用电负载。台区变电站的最小用电负荷缺口为负荷与额定功率之差，缺口总额为：1.6803*10^3（kWh）。

移动储能车的满载电池容量为 5*（200kWh+400kWh）=3*10^3（kWh）。对比可知，验证电能满足高峰期用电负载，即移动储能车电量足够填补台区变负荷外的客户用电缺口。

第三，我们认为在保证第一、二类客户需求的同时，尽可能地满足第三类客户需求以提高移动储能车的放电服务收益。根据计算得出的最小负荷缺口，本文提出一种平凡解的调度方案，以证明调度方案存在更优解使得其具有满足第三类客户需求的潜质：

从移动储能车集合存放点发车后，将第一类客户所在的#3 台区停放 2 辆 MV#1 型移动储能车，50kW 的额定功率能够满足 12:00-20:00 前超出台区用电负荷的需求；在第一类客户所在的#4 台区停放 1 辆 MV#2 移动储能车，100kW 的额定功率能够满足 12:00-15:00 前的放电服务需求，1 辆 MV#1 型移动储能车能够满足 15:00-20:00 的放电服务需求。第二类客户所在的#1 台区停放 1 辆 MV#2 移动储能车能够满足 09:00-15:00 前的放电服务需求，1 辆 MV#1 型移动储能车能够满足 15:00-20:00 的放电服务需求。

此时，车辆仍然剩余 MV#1 移动储能车 2 辆，MV#2 移动储能车 3 辆。同理，安排第三类客户所在的#2 台区停放 1 辆 MV#1 移动储能车能够满足 12:00-16:00 前的放电服务需求，1 辆 MV#2 型移动储能车能够满足 16:00-20:00 的放电服务需求。

因此，我们得出能够在保证不中断所有类型客户供电的情况下，使用移动储能车为第三类客户完全提供放电服务，既提升服务质量，也提升收益。

3.3.4 储能车放电顺序，不同储能车放电过程中间是否能有间隔，是否交替放电？不同车在不同时间段在不同台区变以什么功率放电？

储能车放电顺序作为决策变量，由后续算法决定。

针对不同车辆放电之间是否会有间隔。存在间隔优势 1.避免在低电价放电。2.不需要放电时，不放电节省车辆电量，避免出现后续无法应对缺口情形。缺点 1.缺电时间较为集中，如果不同车辆充放电之间有间隔，可能出现电力短缺情形。2.中间一段时间不放电，也会拖延之后车辆返回车场时间。由于本文假定数据车辆电量充足，且在低电价放电也能获得正收益。所以需要尽可能耗电，以获得更多收益。3.3.1 的平凡解证明，不存在无法应对电力缺口问题。

因此本文假设不同车辆放电之间不存在间隔时间。

其次不同车辆是否能够交替放电。即 A 车放电一段时间，B 车放电一段时间，A 车再放电一段时间，即 ABA 情形放电。

如果 AB 车辆属于同一类型车辆，例如都属于 MV#1 型车，我们认为这样交替放电与 AB 情形相比（即 A 全部放完电后，B 再全部放完电），情形一 B 车到充电站时间，比情形二 A 车到充电站时间晚。情形一 A 车到充电站时间，与情形二 B 车到充电站时间相同。这样造成充电站未被充分利用。且两种情形收益相同。因此，我们假设同一型号车辆不需要交替放电。

同理不同型号车辆之间交替放电不能增加总收益，还可能延迟车辆到车场时间。但不太车型之间交替放电也有其优势，即若电力缺口大于小型 MV#1 车功率，但小于大型 MV#2 型功率时，可以考虑交替放电。但经过前期分析，我们认为车辆数量和容量完全可以在不交替放电时，满足用电缺口需求，有较多富余，此时交替放电方案与不交替放电方案相比，无显著优势，且未能充分利用充电站，可能耽误车辆充电时间。

最终我们假设车辆不会交替放电。

3.3.5 储能车是否需要全部放完电后再去充电？

车辆在未全部放电时就回去充电，优势 1.避免回不到车场。2.避免低电价放电，高电价充电。缺点 1.收益减少。

3.3.1 的平凡解证明全部放完电再去充电完全可行。

充电站充电功率为 400kW，200/400=0.5h 可以充满一辆低功率车，400/400=1h 时可以充满 1 辆高功率车。4 个充电站，充满 5 辆低功率车和 5 辆高功率车共需要 2h 左右。台区 1 最后 3 小时无用电缺口，其余 3 个台区最后 2 小时无用电缺口。停车场到各个台区变最远距离小于 20km，用时小于 0.6h。考虑有些车辆在 19 点前已经放完电，因此我们认为剩余 2h 时间是足够剩余车辆充电并回到车场的。

因此假定车辆全部放完电后再去充电。

其次为了避免高价放电，低价充电。假设禁止车辆在 7 时到 12 时以及 16 时到 18 时充电。

3.3.6 储能车选择哪个充电站充电？

不同充电站充电价格一致，充电功率一致。选着充电站唯一影响因数是，小车从台区变出发，经过充电站，最后到达车场的距离最近。**使用迪杰斯特拉算法(Dijkstra)找到最短路径，穷举组合后选择最短组合。**

经过测算，不存在（有可能去最近充电站再返回 1 号点的距离，大于到其他充电站再返回 1 号点的距离）。因此，为了避免移动储能车产生更多的路程成本，**因此我们选择最近的充电站进行充电。**

3.3.7 储能车选择哪条路径前往充电站？

该问题是典型的有权图中最短路径问题，且节点数量较少，可以**使用迪杰斯特拉算法(Dijkstra)求最优解。**

3.3.8 储能车到充电站后是否立即充电，什么时间充电？

同 3.3.5 分析，车辆回车场的时间是充足的，**为了避免高价放电，低价充电。假设禁止车辆在 7 时到 12 时以及 16 时到 18 时充电。**

3.3.9 储能车是否能够按时返回车场？

同 3.3.5 分析，车辆回车场的时间是充足的，**假定车辆能返回车场。**

3.3.10 小结

综合上述分析，我们可以得出以下 X 点新假设。

1）台区变 1、2、3 各需要一辆大功率车。

2）使用移动储能车满足第三类客户需求

3）**车辆只能选择一个台区变。**

4）不同车辆放电之间不存在间隔时间。

5）车辆不会交替放电。

6）车辆全部放完电后再去充电。

7）禁止车辆在 7 时到 12 时以及 16 时到 18 时充电。

8）选择最近的充电站进行充电。

在后续编码求解过程中。1.使用迪杰斯特拉算法(Dijkstra)求最优解路径。2.车辆选择台区变和储能车放电顺序需要作为决策变量，由后续算法决定。

根据上述假设，我们可以得出新的车辆调度流程。

1.首先将车辆分成 4 个不同组对应不同的台区变。

2.7 点车辆同时发车前往 4 个台区变。

3.在台区变等待一段时间后，按照一定的顺序放电。

4.每辆车在不同时间段放电功率不同，所有电耗尽后立即前往充电站。下一个顺序车紧接着放电。

5.车辆在禁止充电时间段外充电。

6.车辆充电完成后前往车场。

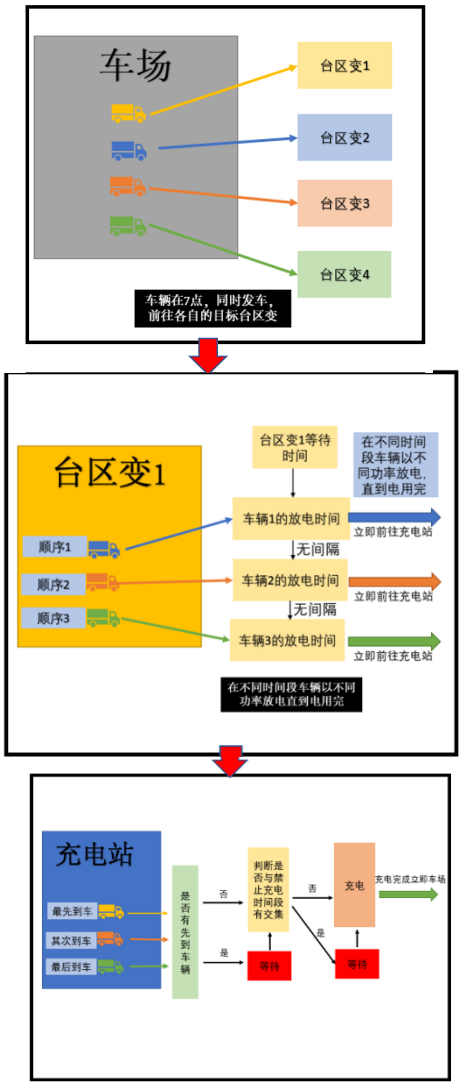

图 5 车辆充放电流程示意图

3.4 车辆调度模型建立

3.4.1 目标函数

总收益包括车辆放电收入，车辆充电支出，车辆行驶支出三个部分。

$$maxB = \sum_{k=1}^{10}\sum_{N} t_{\mathrm{Nk}}R_{1Nk}p_{\mathrm{Nk}} - \sum_{k=1}^{10}\sum_{M} t_{\mathrm{Mk}}R_{2Mk}P_{M} - \sum_{k=1}^{10} D_{kN}C_{k}$$

车辆放电收入=车辆放电时间段*车辆放电时价格*车辆放电时功率。

$$\sum_{k=1}^{10}\sum_{N} t_{\mathrm{Nk}}R_{1Nk}p_{\mathrm{Nk}}$$

其中t_{Nk}为车辆k在台区变N放电时间段，$t_{Nk} = [a_{Nkup} - a_{Nk}, 1, \dots, 1, b_{Nk} - b_{Nkdown}]$，$a_{Nk}$为车 k 在台区变 N 的放电开始时间，$a_{Nkup}$为$a_{Nk}$向上取整。例如$a_{Nk} = 7.3$，$a_{Nkup} = 8$。$b_{Nk}$车 k 在台区变 N 的放点结束时间，$b_{Nkdown}$为$b_{Nk}$向下取整。例如$b_{Nk} = 7.3$，$b_{Nkdown} = 7$。$R_{1Nk}$为车辆 k 在台区变 N 放电时价格，$R_{1Nk} = [R_{1[a_{Nkdown}]}, R_{1[a_{Nkdown}+1]}, \dots, R_{1[b_{Nkdown}]}]$，$R_{1[a_{Nkdown}]}$为$a_{Nkdown}$时间的放电价格。$p_{\mathrm{Nk}}$为车辆 K 在台区变 N 放电时功率，$p_{\mathrm{Nk}} = [p_{Nk[a_{Nkdown}]}, p_{Nk[a_{Nkdown}+1]}, \dots, p_{Nk[b_{Nkdown}]}]$。

车辆充电支出=车辆充电时间段*车辆放电时价格*车辆放电时功率。

$$\sum_{k=1}^{10}\sum_{M} t_{\mathrm{Mk}}R_{2Mk}P_{M}$$

其中t_{Mk}为车辆 k 在快充电站 M 充电时间段，$t_{Mk} = [a_{Mkup} - a_{Mk}, 1, \dots, 1, b_{Mk} - b_{Mkdown}]$，$a_{Mk}$为车 k 在快充电站 M 的充电开始时间，$a_{Mkup}$为$a_{Mk}$向上取整。例如$a_{Mk} = 7.3$，$a_{Mkup} = 8$。$b_{Mk}$车 k 在快充电站 M 的充电结束时间，$b_{Mkdown}$为$b_{Mk}$向下取整。例如$b_{Mk} = 7.3$，$b_{Mkdown} = 7$。$R_{1Mk}$为车辆 k 在快充电站 M 充电时价格，$R_{1\mathrm{Mk}} = [R_{1[a_{Mkdown}]}, R_{1[a_{Mkdown}+1]}, \dots, R_{1[b_{Mkdown}]}]$，$R_{1[a_{Mkdown}]}$为$a_{Mkdown}$时间的充电价格。$p_{\mathrm{Mk}}$为车辆 K 在快充电站 M 放电时功率，$p_{\mathrm{Mk}} = [p_{Mk[a_{Nkdown}]}, p_{Mk[a_{Mkdown}+1]}, \dots, p_{Mk[b_{Mkdown}]}]$。

车辆行驶支出=车辆行驶路程*车辆行驶成本

$$\sum_{k=1}^{10} D_{kN}C_{k}$$

D_{kN}为车辆 k 选择台区变 N 后，从车场到台区变 N 距离+台区变 N 到最近充电站距离+充电站到车场距离。C_{k}为车辆每公里行驶成本。

3.4.2 约束

需要满足各类用户需求。

$$p_{Nt} \le p_{\mathrm{Nkt}} \le P_{1e}, k = 1,2,3,4,5$$
$$p_{Nt} \le p_{\mathrm{Nkt}} \le P_{2e}, k = 6,7,8,9,10$$

其中p_{Nt}为台区变节点 N 在 t 时刻的电力缺口，p_{Nkt}车辆 k 在台区变 N 服务 t 时刻的放电功率，P_{1e}为小功率储能车的额定功率，P_{2e}为大功率储能车的额定功率。

车辆 k 在 22 点前返回车场。

$$T_{k} \le 15$$

3.5 路径选择模型建立

针对最短路径问题，目前常用的方法包括 Dijkstra 算法、A-Star 算法、Floyd 算法和 Bellman-Ford 算法等。其中，Dijkstra 算法凭借寻优速度快、代码实现简单等特点，在路径优化、节点调控等领域得到了广泛应用。其核心思想在于以起点为中心向外层逐步扩展，求解从起点到其他所有节点的最短路径，直到扩展到终点。

Dijkstra 算法[1]是基于贪心思想实现的，利用这种类似广度优先搜索的方法来解决赋权图的单源最短路径问题，即指定一个点（源点）到其余各个节点的最短路径。在本题中车辆行驶的交通网络属于拓扑结构相对固定、路径权值相对稳定的网络，因此我们可以采用 Dijkstra 算法来找出道路阻断后，受影响的车辆里程最短的路径。

将地图抽象为一个带权重的有向图$G(N,D)$，其中N为节点集合，D为存在联结关系的所有节点间连线的集合，代表着边。定义起点为N_s，终点为N_e，$d_{ij}^{\min}$为节点i出发到节点j的最短距离，a_{ij}为节点i到节点j的有向路径，S为由起点N_s出发，所到路径最短的节点集合，U则是还未求出最短路径的节点集合。定义d_{ij}如果存在由节点i指向节点j的边，则等于从节点i出发，到达节点j的距离，否则$d_{ij}=\infty$，且l_{ij}不存在。初始化$S=\{N_s\}$，定义起点N_s到自身的最短路径表示为a_{ss}，最短距离$d_{ij}^{\min}=d_{ss}=0$。

由于不能在其他节点停留，我们只求解车场到各个台区变最短路径，各个台区变到充电站路径，充电站到车场路径。

3.6 Dijkstra 算法求解最短路径

依据据 Dijkstra 算法的基本思想，搜索起点N_s到达终点N_e之间的最短路径过程包括以下步骤。

第 1 步，找到起点N_s所有单跳可以到达的邻节点，从中找到距离最短的相邻节点(设为N_1)，将其放入节点集合S中，即有$S=\{N_s,N_1\}$，则起点N_s到节点N_1的最短路径距离为$d_{ij}^{\min}=d_{s1}$，最短有向路径为a_{s1}。

第 2 步，找出集合S中的节点$\{N_s,N_1\}$所有单跳可达的邻节点，将其设为N_2、N_3。以N_2为例，起点N_s到节点N_2的有向路径分别为a_{s2}或$a_{s1}\rightarrow a_{12}$，对应的路径距离为d_{s2}或者$d_{s1}\rightarrow d_{12}$。相同的原理我们能够找出起点N_s到节点N_3的距离和有向路径。

第 3 步，如果起点N_s到节点N_2的最短路径距离小于起点N_s到节点N_3的最短路径距离，则将节点N_2加入集合S，且有$d_{ij}^{\min}=\min\{\min\{d_{s2},d_{s1}+d_{12}\},\min\{d_{s3},d_{s1}+d_{13}\}\}$。此时有$S=\{N_s,N_1,N_2\}$，起点$N_s$到节点$N_2$的最短距离为$d_{s2}^{\min}$。

第 4 步，以此类推，遍历集合U中其他所有的节点并更新集合S，直到终点N_e被加入到集合S中时，算法结束。

将本题的 29 个节点数据读入后，利用 Dijkstra 算法规划的路径如下表所示：

表 5 从起点到各台区变节点的路径规划

起点	终点	最短路径	距离
1	5	1-4-5	3.7
1	14	1-4-5-6-9-24-14	13.6
1	19	1-4-7-21-19	11.2
1	27	1-4-7-21-22-27	11.7

表 6 从各台区变节点到充电站节点的路径规划

起点	终点	最短路径	距离
5	6	5-6	2.1
14	13	14-13	3.6
19	18	19-18	2.4
27	22	27-22	2.3

表 7 从各充电站节点回到起点的路径规划

起点	终点	最短路径	距离
6	1	6-5-2-1	5.8
13	1	13-10-6-3-2-1	12.4
18	1	18-19-21-7-4-1	13.6
22	1	22-21-7-4-1	9.4

3.7 使用改进遗传算法求解车辆调度模型

3.7.1 算法描述

本文研究的移动储能车的优化调度问题是典型的组合优化问题，属于 NP-hard 问题。可以采用智能搜索算法求解该类问题。目前存在多种只能搜索算法，如遗传算法(GA)、粒子群算法(PSO)、蚁群算法(ACO)和化学反应优化算法(CRO)等。

遗传算法（Genetic Algorithm, GA）是模拟达尔文生物进化论的自然选择和遗传学机理的生物进化过程的计算模型，是一种通过模拟自然进化过程搜索最优解的方法。其以直接对结构对象进行操作，不存在求导和函数连续性的限定、具有内在的隐并行性和更好的全局寻优能力、采用概率化的寻优方法以及不需要确定的规则就能自动获取和指导优化的搜索空间，自适应地调整搜索方向等优点，被广泛使用在组合优化问题当中。由于遗传算法具有比较完备的数学模型和理论，在解决很多 NP—Hard 问题上具有良好的性能，针对移动储能车的优化调度问题，本文采用遗传算法进行求解。

3.7.2 算法框架

基本遗传算法[2] (也称标准遗传算法或简单遗传算法，Simple Genetic Algorithm，简称 SGA)是一种群体型操作，该操作以群体中的所有个体为对象，只使用基本遗传算子(Genetic Operator)：选择算子(Selection Operator)、交叉算子(Crossover Operator)和变异算子(Mutation Operator)，其遗传进化操作过程简单，容易理解，是其它一些遗传算法的基础，它不仅给各种遗传算法提供了一个基本框架，同时也具有一定的应用价值。选择、交叉和变异是遗传算法的 3 个主要操作算子，它们构成了遗传操作，使遗传算法具有了其它方法没有的特点。遗传算法的步骤包括染色体的编码、解码、初始群体的生成、适应度值评估监测以及遗传算子。为了加快收敛，我们增加二元精英竞标赛和领域搜素算法两个环节。

其算法流程图如下：

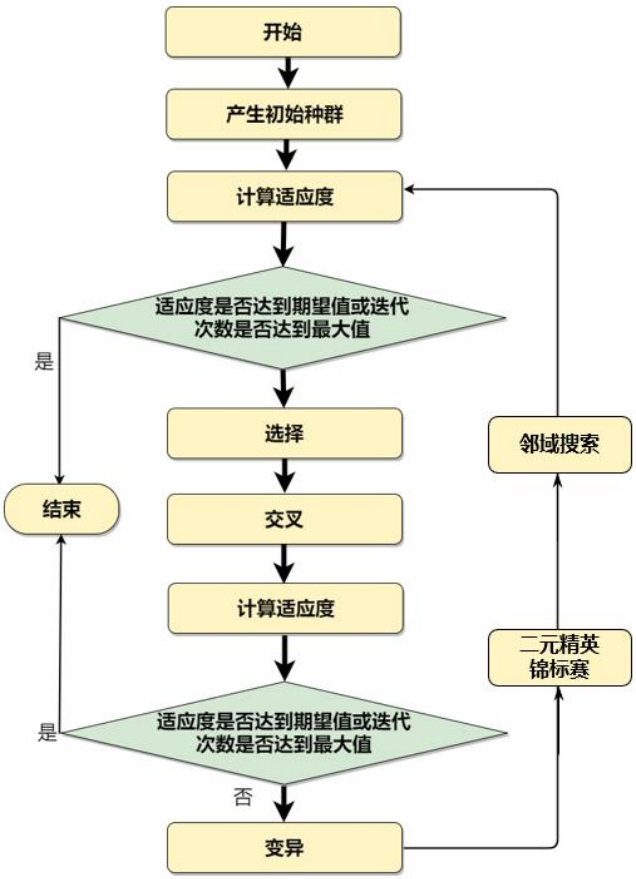

图 6 遗传算法流程图

3.7.3 问题编码

解空间中的解在遗传算法中的表示形式。从问题的解(solution)到基因型的映射称为编码，即把一个问题的可行解从其解空间转换到遗传算法的搜索空间的转换方法。遗传算法在进行搜索之前先将解空间的解表示成遗传算法的基因型串(也就是染色体)结构数据，这些串结构数据的不同组合就构成了不同的点。而解码则是遗传算法染色体向问题解的转换。

在上述分析基础上，我们将编码分成两个部分，第一是对台区变编码，第二是对储能车编码。

（一）台区变编码

对于四个台区变来说，需要优化的变量就是储能车来放电的时间，因此对四个台区变编码四位，代表四个台区变第一次被服务的等待时间。

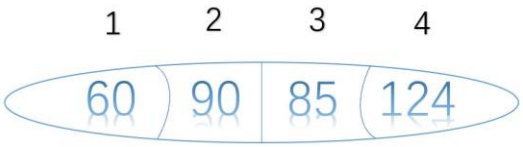

图 7 台区变染色体编码

如上图所示，1,2,3,4 号分别对应节点 5,14,19,27 的台区变，此条染色体代表的现实意义就是 1 号台区变首次被储能车服务的等待时间为 60 分钟，即 8 点开始被储能车服务，若储能车 8 点前到台区变 1，需要 8 点后开始服务。若车辆 8 点前未能到台区变 1，则到达台区变 1 后，马上开始服务；2 号台区变首次被储能车服务的时间为 8 点 30，以此类推。

（二）储能车编码

对于每一辆储能车来说，需要决策的变量有去哪一个节点的台区变、什么时候开始放电以及每一时刻采用多大的功率进行放电。什么时刻放电可以编码为车的放电顺序，放电顺序编码为 1-10 的自然数，当同一台区变有多台储能车在等待服务时，按照序号的大小给他们进行排序。假设每一辆车都放电直到电池容量耗尽，可以根据放电功率计算出放电的结束时间，也就是顺序中下一辆车的开始放电时间。考虑只有两种电价，我们认为电价不同，车辆应该以不同功率放电。在保证满足台区变功率需求的前提下，在放电价格高的时候采用高功率放电，放电价格低的时候采用低功率放电。

因此编码包括 4 个部分。车辆选择的台区变，车辆在台区变的排序，高电价时预计比额定功率少输出功率，低电价预计比额定功率少输出功率。

以第一辆高功率储能车为例，编码如下图所示：

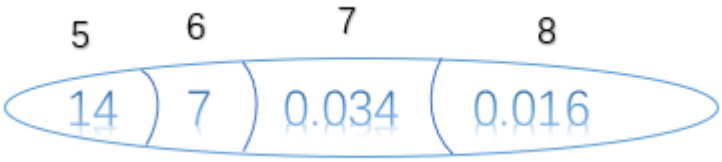

图 8 第一辆高功率储能车编码

染色体中 5-8 号位分别代表第一辆高功率储能车服务台区变为节点 13 号、放电顺序为第 7 个、低电价预计放电功率为，额定功率-7 号位算子，即 0.1MW-0.034MW=0.066MW，高电价预计放电功率为，额定功率-8 号位算子，即 0.1MW-0.016MW=0.084MW。

在解码过程中我们会根据实际电力缺口，以及编码给定预计功率中较大者，作为实际输出功率。

编码时先对五辆高功率储能车进行编码，再对五辆低功率储能车进行编码。对于剩下九个储能车编码方式相同，每辆车占用四位编码，分别编为 9-44 号编码。如下图所示：

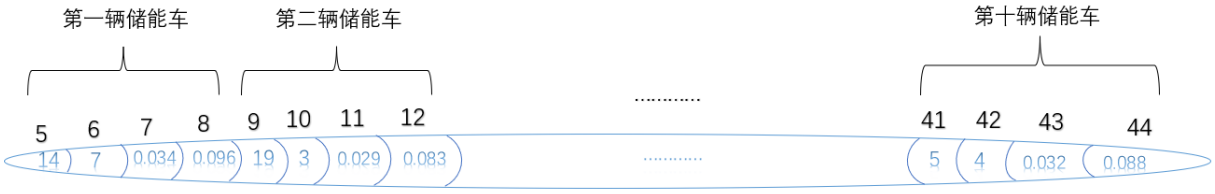

图 9 储能车编码示意图

（三）适应度函数

本文优化的指标为所有储能车的收益，调度储能车时在满足所有台区变的供电需求时，尽可能的让储能车在充电价格低的时候去充电，在放电价格高的时候进行放电，目的是最大化收益。

适应度函数跟优化目标有关，是评估染色体好坏的标准，适应度值越高，代表所求问题的解越优。本问题希望收益最大化，因此可以写出适应度函数如下式：

$$Fitness(B)=\sum_{k=1}^{10}\sum_{N}t_{\mathrm{Nk}}R_{1Nk}p_{\mathrm{Nk}}-\sum_{k=1}^{10}\sum_{M}t_{\mathrm{Mk}}R_{2Mk}P_{M}$$

式中t_{Nk}代表车辆 k 在台区变 N 的服务时间段列表，该变量可以由从车站到台区变时间，台区变等待时间，等待前面车辆放电时间（由之前车辆输出功率决定），该车放

电时间（车辆本身输出功率决定）获得。

t_{Mk}代表车辆 k 在充电站 M 的服务时间段列表由从台区变出发时刻，台区变到充电站距离，是否在禁止充电时间，车辆电池容量获得。

R_{1Nk}代表车辆 k 在台区变 N 的服务时间段的放电价格列表和R_{2Mk}代表车辆 k 在充电站 M 的服务时间段的充电价格列表可以通过查询时段电价表获得。

p_{Nk}代表车辆 k 在台区变 N 的服务时间段的放电功率列表，由目前电价高或低，选择编码中给定的预计输出功率，结合实际电力缺口得出实际输出功率。

P_M代表充电站的快速充电功率，恒为 400kW。

（四）遗传算法初始解产生

结合先验知识，采用在给定范围内生成随机数的方式产生初始解。对于台区变的四个编码，我们认为车辆应该在 2 小时内开始放电，因此随机生成 0-180 的自然数。

对于每一辆储能车的四位编码，第一位在台区变四个节点中任取一个。

第二位在 1-20 生成一个自然数。

储能车的第三位，第四位编码在[0,0.001,0.002,0.003，……，0.020]中的选择两个数。

（五）交叉运算

交叉即对两个相互配对的染色体依据交叉概率 Pc 按某种方式相互交换其部分基因，从而形成两个新的个体。交叉运算是遗传算法区别于其他进化算法的重要特征，它在遗传算法中起关键作用，是产生新个体的主要方法。

此问题的交叉方法我们采用整数交叉的方式。在进行交叉运算之前，首先要删除染色体中所插入的负数 M，只留下表示检查点序号的数；然后随机选择两个交叉位置和数目，把个体和个体极值或者群体极值进行交叉。以下式的 A，B 染色体进行交叉为例：

$$A = 1\ 9\ 15\ 11\ \left[6\ 13\ 2\ 17\right]\ 10\ 18\ 14\ 3\ 8\ 7\ 19\ 4\ 16\ 5\ 20\ 12$$

$$B = 1\ 27\ 5\ 23\ \left[21\ 34\ 13\ 2\right]\ 20\ 33\ 10\ 11\ 8\ 31\ 16\ 29\ 26\ 38\ 30\ \ 12$$

对 A 和 B 染色体交叉后，我们得到

$$A = 1\ 9\ 15\ 11\ \left[21\ 34\ 13\ 2\right]\ 10\ 18\ 14\ 3\ 8\ 7\ 19\ 4\ 16\ 5\ 20\ 12$$

$$B = 1\ 27\ 5\ 23\ \left[6\ 13\ 2\ 17\right]\ 20\ 33\ 10\ 11\ 8\ 31\ 16\ 29\ 26\ 38\ 30\ \ 12$$

（六）变异运算

变异即依据变异概率 Pm 将个体编码串中的某些基因值用其它基因值来替换，从而形成一个新的个体。遗传算法中的变异运算是产生新个体的辅助方法， 它决定了遗传算法的局部搜索能力，同时保持了种群的多样性。具体实现步骤如下：首先在种群中随机选择 3 个基因位点 14、28 和 13，然后将 3 个基因位点之前的染色体片段位置互换，例如：

$$C = 1\ 32\ 15\ 40\ \left[14\right]\ 22\ 26\ 36\ \left[28\right]\ 30\ 18\ 34\ 8\ \left[13\right]\ 37\ 23\ 35\ 29\ 3\ 12$$

变异后得到

$$C = 1\ 32\ 15\ 40\ \left[13\right]\ 22\ 26\ 36\ \left[14\right]\ 30\ 18\ 34\ 8\ \left[28\right]\ 37\ 23\ 35\ 29\ 3\ 12$$

（七）二元精英竞标赛

由于种群中的个体数目十分庞大,选择合适的选择策略对于遗传算法整体性能有很大的影响。如果一个选择算法选择多样性降低，便会导致种群过早的收敛到局部最优点而不是我们想要的全局最优点，也就是所谓的“早熟”。而选择策略过于发散则会导致算法难以收敛到最优点。因此在这两点中我们需要进行平衡才能使遗传算法以一种高

效的方式收敛到全局最优点，因此采用二元锦标赛的选择策略。锦标赛是一种具有高效率执行和易于实现的选择算子，原理是每次从种群中取一定数量N的个体，选择其中适应度较好的进入子代种群，直到子代种群达到原来的种群规模，保证了适应度较好的基因保留到下一代。实现步骤如下：

1. 确定选择个体数量N。（二元锦标赛选择$N=2$的个体）；
2. 从种群中随机选择N个个体（以相同的概率选择每个个体），根据个体适应度值，选择其中适应度值最好的个体进入下一代种群；
3. 重复选择过程，直到新的种群规模达到原来的种群规模。

（八）领域搜素算法

为了避免 GA 对优秀个体的搜索不充分的情况，最优个体结合邻域搜索算法进行局部最优解的改进。

自适应大邻域搜索(Adaptive Large Neighborhood Search, ALNS)算法根据算子的历史表现与使用次数选择下一次迭代使用的算子，通过算子间的相互竞争来生成当前解的邻域结构，而在这种结构中有很大概率能够找到更好的解。

ALNS 的算子有：

- 毁坏算子：随机破坏解的一部分，主要有随机移除、最差移除、相似移除等方法；
- 修复算子：重建被破坏的解，主要有随机插入、贪婪插入等方法；

ALNS 算法至少选择两组毁坏与修复算子，并在邻域动态调整权重，每个毁坏和修复算子在搜索期间出现的频率受到权重控制。迭代时根据算子权重和轮盘赌的方式进行选择、调整破坏算子和修复算法。其中，自适应权重的计算公式为：

$$\omega_d = \begin{cases} \omega_d, & if\ u_d = 0 \\ (1-\rho)\omega_d + \rho\dfrac{s_d}{u_d}, & if\ u_d > 0 \end{cases}$$

ω_d是算子权重，s_d为算子分数，u_d为算子的使用次数，ρ是权重更新系数，作为一种可调节参数控制权重变化的速度。

在开始时，所有算子具有相同的权重和分数，随着迭代过程，根据算子的不同表现予以加分，例如：

- 毁坏/修复后得到新的全局最优解，加 1.5 分；
- 毁坏/修复后没有得到新的全局最优解：
  （1） 比当前解好：加 1.2 分；
  （2） 比当前解差：若可以接受，加 0.8 分；否则加 0.6 分。

3.8 遗传算法解码器

在得到改进后遗传算法输出的算子编码后，我们按照以下步骤解码，并计算适应度。

将编码分成两个部分，第一是对台区变编码，第二是对储能车编码。台区变编码控制储能车最早放电的时间。储能车编码包括 4 个部分。车辆选择的台区变，车辆在台区变的排序，高电价时预计比额定功率少输出功率，低电价预计比额定功率少输出功率。

步骤一：调整编码，增加车辆状态记录表格

根据之前假设，台区变 1、2、3 各需要一辆大功率车。因此我们直接将前 3 台高功率车的车辆选择的台区变编码，直接修改 1,2,3。

增加车辆状态记录表格。每辆车记录 3 个信息。车辆到当前状态耗时，车辆当前状态剩余电量，车辆到当前状态收益。

步骤二：车辆从车场到指定台区变

1.查询编码可以获得每个车辆将要到达台区变。

2.查询之间计算的最短路径，获得车辆到台区变最短距离。

3.根据最短距离、车辆速度、车辆行驶成本，更新每一辆车状态记录中，车辆到当前状态耗时和车辆到当前状态收益。

步骤三：车辆在台区变放电结束

1.通过查询车辆顺序编码，将同一个台区变的车辆排序，编码小的先放电。

2.根据台区变等待时间，确定第一辆车开始放电时间。

3.根据实时电价是高价还是低价，查询编码确定不同时间段预计放电功率。查询该时间段该台区变电力缺口，确定缺口功率。取预计放电功率和缺口功率中较大者作为每个时间段实际放电功率。若缺口功率大于车辆最大放电功率，直接结束计算。

4.根据实际放电功率，每一时段电价，车辆当前状态剩余电量，迭代更新车辆状态记录表格，直到电量耗尽结束。最终得到车辆在台区变放电结束时间和放电收益

5.之后车辆开始放电时间为上一辆车结束放电时间。按照同样的方法，计算车辆实际放电功率，迭代更新车辆状态表格，直到所有车辆放电结束。

步骤四：车辆从台区变到充电站

1.查询编码可以获得每个车辆将要到达台区变。按照之前计算的台区变最优充电站，选择充电站。

2.查询之间计算的最短路径，获得车辆到充电站最短距离。

3.根据最短距离、车辆速度、车辆行驶成本，更新每一辆车状态记录中，车辆到当前状态耗时和车辆到当前状态收益。

步骤五：车辆在充电站充电

1.根据车辆状态表格，查询车辆到充电站充电顺序。

2.第一辆比较第一辆车到充电站时间和禁止充电时间，得到车辆充电时间，充电结束时间以及充电花费。更新车辆状态表格。

3.之后车辆最早预计开始充电时间为上一辆车结束充电时间，以及车辆到达充电站时间中较小者。

4.按相同方法得到车辆充电时间，充电结束时间以及充电花费。更新车辆状态表格。

步骤六：车辆从充电站到车场

1.查询之间计算的最短路径，获得车辆到车场最短距离。

2.根据最短距离、车辆速度、车辆行驶成本，更新每一辆车状态记录中，车辆到当前状态耗时和车辆到当前状态收益。

步骤六：得到适应度值

1.查询车辆状态记录表格，对每一辆车收益求和，得到初步总收益收益。

2.若发现车辆超时返回车场，对超时部分给予一定惩罚，得到适应度值

3.9 结果分析

运行相关算法画出收益随种群代数增长曲线如上图所示，可以看出初代种群的收益基本没有，随着种群代数增加，收益能够快速增长，在种群代数到达大约 200 代的时候，收益逐渐收敛并达到**最大值 632.88**，证明了算法的有效性和稳定性。

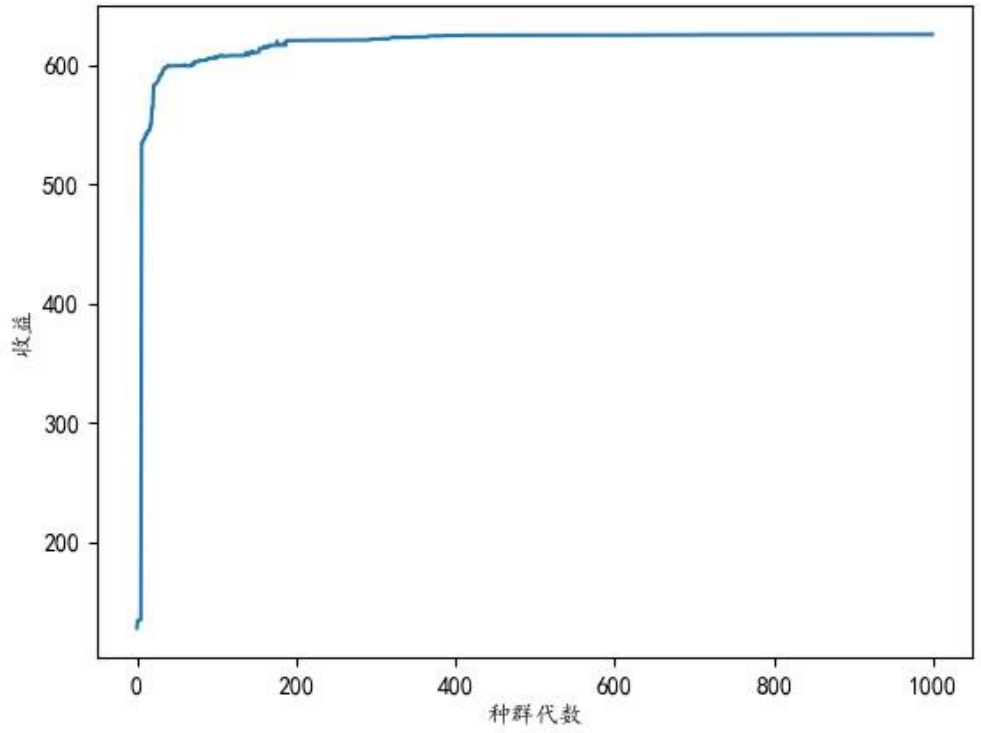

图 10 遗传算法收益迭代曲线

得到的车辆最终规划方案如下表所示：

表 8 车辆在各个台区变服务情况规划方案

车辆	服务台区变节点	开始放电时间	高放电功率	低放电功率	放电结束时间	去往充电站节点
MV#2-1 号车	5	8:01	94.01	60.03	12:22	6
MV#2-2 号车	14	14:40	45.79	40.78	20:28	13
MV#2-3 号车	27	7:59	99.95	50.87	12:00	22
MV#2-4 号车	5	12:22	99.73	85.04	16:55	6
MV#2-5 号车	27	12:00	99.95	72.64	17:05	22
MV#1-1 号车	5	16:55	49.03	40.74	21:00	6
MV#1-2 号车	14	10:10	40.07	37.87	14:40	13
MV#1-3 号车	27	17:05	49.95	44.74	21:06	22
MV#1-4 号车	19	9:46	28.43	27.28	15:16	18

MV#1-5 号车	19	15:16	30.94	37.93	20:57	18

表 9 车辆在各个充电站充电情况规划

车辆	充电节点	开始充电时间	结束充电时间	返回车场时间	收益
MV#2-1 号车	6	12:27	13:27	13:38	155.13
MV#2-2 号车	13	20:35	21:35	22:00	75.53
MV#2-3 号车	22	12:04	13:04	13:23	140.31
MV#2-4 号车	6	18:00	19:00	19:11	13.73
MV#2-5 号车	22	18:00	19:00	19:18	-3.89
MV#1-1 号车	6	21:04	21:34	21:45	81.99
MV#1-2 号车	13	14:48	15:18	15:43	22.37
MV#1-3 号车	22	21:10	21:40	22:00	68.55
MV#1-4 号车	18	15:21	15:51	16:18	23.80
MV#1-5 号车	18	21:02	21:32	22:00	55.37

表 10 车辆路线图

车辆	车场到台区变	台区变到充电站	充电站到车场	行驶距离
MV#2-1 号车	1-4-5	5-6	6-5-2-1	11.6
MV#2-2 号车	1-4-5-6-9-24-14	14-13	13-10-6-3-2-1	29.6
MV#2-3 号车	1-4-7-21-22-27	27-22	22-21-7-4-1	23.4
MV#2-4 号车	1-4-5	5-6	6-5-2-1	11.6
MV#2-5 号车	1-4-7-21-22-27	27-22	22-21-7-4-1	23.4
MV#1-1 号车	1-4-5	5-6	6-5-2-1	11.6
MV#1-2 号车	1-4-5-6-9-24-14	14-13	13-10-6-3-2-1	29.6
MV#1-3 号车	1-4-7-21-22-27	27-22	22-21-7-4-1	23.4
MV#1-4 号车	1-4-7-21-19	19-18	18-19-21-7-4-1	27.2
MV#1-5 号车	1-4-7-21-19	19-18	18-19-21-7-4-1	27.2

我们所提方案在追求利润最大化前提下，满足所有用户用电需求，做到车辆行驶路程最优，且所有车辆都按时返回车场。

虽然我们没有对车辆在停车场充电时间进行显性编码，即只是决定车辆在充电站禁止充电时间，没有决定车辆是否在充电站等待最优电价，以及等待多长时间。但在最终规划的方案车辆充分利用最后一小时低电价优惠，获得最优方案。

从最后车辆返回停车场时间，可以发现有 3 辆车在最后时刻返回车场，这表明算法能够让车辆充分利用时间，获得最优方案。

4 问题二分析与模型建立

4.1 问题分析

问题 2 题目数据与问题 1 相比，增加光伏发电数据。我们首先对光伏发电数据进行分析。

问题 2 包括两个问题。第一个问题是最少需要几辆车能满足用户用电需求。我们分只满足第一第二类用户用电需求，满足全部用户用电需求等两种情况讨论。通过数据分析和平凡解确定最少用车数量和用车型号。

第二个问题在确定用车数量和型号时，满足用电需求时怎么优化车辆路线，充放电策略，使得收益最大化。只满足第一第二类用户用电需求情形较为简单。我们从利用高电价时放剩余电量和尽可能利用低电价充电两个角度分析，并提出车辆调度方案。

满足所有用户用电需求情况较为复杂。虽然目标函数以及约束与第一问相同，无需改进模型。但由于可以多次充放电假设改变，以及太阳能发电时刻变化，导致充电成本时刻变化，车辆决策空间复杂。我们首先分析数据，缩减问题规模。其重新调整了编码方式和解码方式，最后使用第一问的改进遗传算法求解。

4.2 光伏发电分析

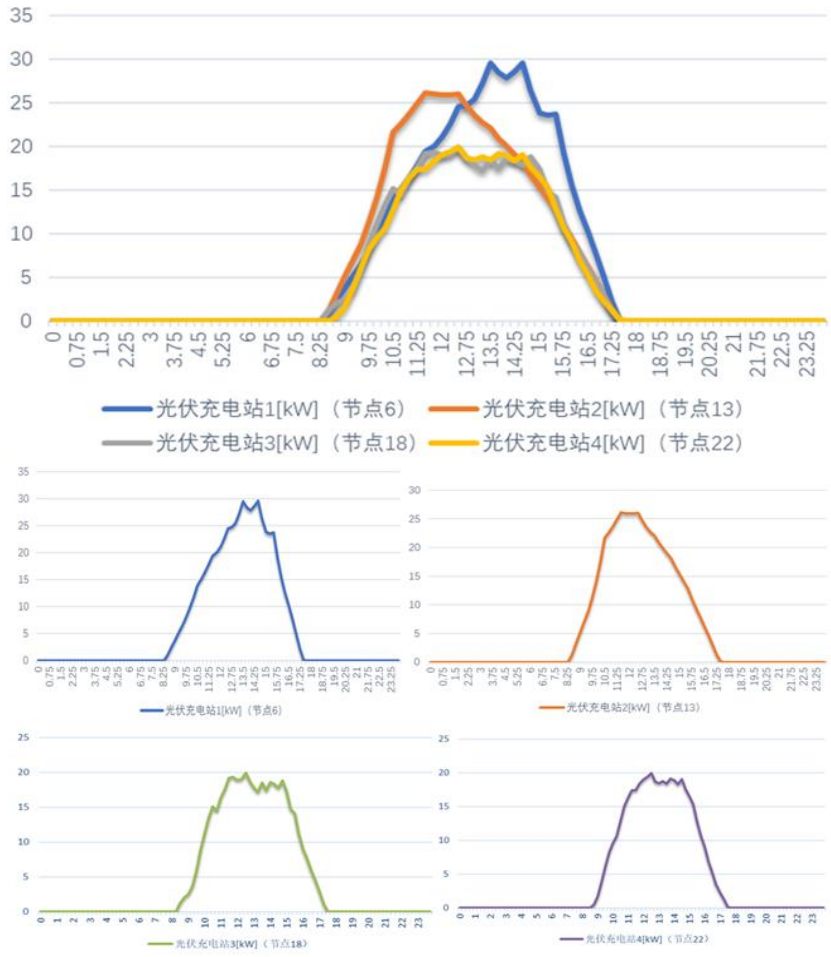

图 11 光伏发电每时刻电价分析

首先，对光伏充电的输出功率数据进行简要分析。光伏充电站 1 的发电功率在 8：15-13：30 时段之间递增，于 13：30 达到功率峰值 29.546（kW），随后在 14：30-17：

30 时段递减。光伏充电站 2 的发电功率在 8：15-11：30 时段之间递增，于 11：30 达到功率峰值 26.12（kW），随后在 12：30-17：30 时段递减。光伏充电站 3 的发电功率在 8：30-13：30 时段之间递增，于 12：30 达到功率峰值 19.96（kW），随后在 14：45-17：30 时段递减。光伏充电站 4 的发电功率在 8：15-13：30 之间递增，于 12：30 达到功率峰值 19.939（kW），随后在 14：30-17：30 时段递减。

4.3 用车数量分析

4.3.1 满足第一、二类用户用电需求

题目将用户分为三类，三类用户分类标准如下。

类型	分类标准
一类用户	突然中断供电会造成人身伤亡或引起周围环境严重污染、造成经济上巨大损失；将会造成社会秩序严重混乱或在政治上产生严重影响的客户
二类用户	突然中断供电对经济上有较大损失，将会造成社会秩序严重混 乱或在政治上产生较大影响的客户
三类用户	不属于上述一类和二类负荷的其它用户

题目要求按照客户优先级确保满足各类用户的用电需求。我们认为一二类用户断电，影响较大，在只满足第一、二类用户用电需求时，即完全不考虑#2 台区参考各时段台区变电力缺口，我们发现在 12:00-13:00 时三个台区都有缺电情况，即至少需要 3 辆大型储能车。

表 11 各个时段台区变电量缺口

用电缺口时段\台区变	#1 台区	#4 台区	#3 台区	总缺电情况
9:00-10:00	0.04	0	0	0.04
10:00-11:00	**0.073**	0	0	0.073
11:00-12:00	**0.074**	0	0	0.074
12:00-13:00	**0.078**	0.032	**0.085**	0.195
13:00-14:00	**0.069**	0.032	**0.07**	0.171
14:00-15:00	**0.059**	0	**0.072**	0.131
15:00-16:00	0.044	0.037	0	0.081
16:00-17:00	0.0474	0.037	0.036	0.1204
17:00-18:00	0.0329	0.022	0.05	0.1049
18:00-19:00	0.04	0.028	0.05	0.118
19:00-20:00	0	0.032	0.04	0.072
总和	0.5573	0.22	0.403	1.1803

假定初始时 1 号车辆分配到#1 台区，2 号大型车辆分配到#4 台区，3 号大型车辆分配到#3 台区。这个方案可以被证明是在调度车辆数目最小（N=3）的情况下，唯一可行的调度方案。由于 1 号车无法满足#1 台区变的全部供电要求， #3 台区变和#4 台区变分别在 14:00-15:00 和 15:00-16:00 不存在电量缺口。并且，3 号车的剩余电量为 173kWh，4 号车的剩余电量为 336kWh。因此，为了保证台区变不断电，需要 1 号车补充到足够的电量，再代替 3 号车或 4 号车放电时。然而，单次充电不足以补充到足够电量，需要由 4 号车代替 1 号车在#1 台区变放电，1 号车前往充电站 4 进行第一次充电，再到达#4 台区放电。随后，由 3 号车代替 1 号车在#4 台区放电，1 号车前往充电站 3 进行第二次充电，最终在#3 台区变放电直到电量用完。

表 12 各个台区最少需要车辆（使用 MV#2 车）

	#1 台区	**#4 台区**	**#3 台区**
MV#2	1	1	1

根据表 X 计算车辆各时刻的准确状态，2 号 14:00 时经过最短路径 19-21-7-4-5 前往#1 台区，到#1 台区 14:22:36，1 号车经过充电站 4 充电 8 分钟，再于 15 时到达#4 台区，剩余约电量 98kWh。3 号 15:00 时经过最短路径 19-18-27 前往#4 台区，到#4 台区 15:13:48，1 号车经过充电站 3 充电 30 分钟，再于 15 时到达 4 台区，剩余约电量 300kWh。3 号车电量约 173kWh 也能满足#4 台区剩余时间供电。同理，经计算车辆在 20:00 点前往 3 个最近充电站充电，都可以在 22:00 点前返回车场。

表 13 各充电站到各个台区变最短距离

	充电站 1（节点 6）	充电站 2（节点 13）	充电站 3（节点 18）	充电站 4（节点 22）
#1 台区变负荷（节点 5）	2.1	8.7	13.7	8.2
#3 台区变负荷（节点 27）	12	13	4.5	2.3
#4 台区变负荷（节点 19）	13.4	18.9	2.4	6.4

通过分析，我们发现 3 辆 MV#2 移动储能车即可满足一二类用电需要。

4.3.2 满足所有用户用电需求

在验证满足所有用户需求下能够采取的最少车辆数时，默认选择更高功率的 MV#2 移动储能车，因此其能够维持更长久的供电时间和更高的供电功率，更久时间供电，可以尽量满足用户需求，且发电量大，收益也会增加。后续分析时优先选择使用 MV#2 移动储能车。

由表分析可知，在 12:00-13:00，4 个台区都有用电缺口，为保证所有用户不在中途断电，至少需要 4 辆车。

表 14 各个时段台区变电量缺口

用电缺口时段\台区变	**#1 台区**	**#2 台区**	**#4 台区**	**#3 台区**	**总缺电情况**
9:00-10:00	0.04	0	0	0	0.04

10:00-11:00	**0.073**	0	0	0	0.073
11:00-12:00	**0.074**	0	0	0	0.074
12:00-13:00	**0.078**	0.045	0.032	**0.085**	0.24
13:00-14:00	**0.069**	0.042	0.032	**0.07**	0.213
14:00-15:00	**0.059**	0.05	0	**0.072**	0.181
15:00-16:00	0.044	0.044	0.037	0	0.125
16:00-17:00	0.0474	**0.085**	0.037	0.036	0.2054
17:00-18:00	0.0329	**0.08**	0.022	0.05	0.1849
18:00-19:00	0.04	**0.072**	0.028	0.05	0.19
19:00-20:00	0	**0.082**	0.032	0.04	0.154
总和	0.5573	0.5	0.22	0.403	1.6803

首先我们分析 4 辆大型储能车是否能够满足所有电力缺口。

#1 台区用电缺口总和 0.5573MVh，大于大型储能车储电量 0.4MVh。因此需要一辆车和 1 台区车交换。发现#4 台区在 14:00-15:00 不缺电。因此两者必须互换。同样地，#2 电力缺口总和 0.5MVh 大于大型储能车储电量 0.4MVh，为#3 台区在 15:00-16:00 不缺电，因此我们#2 台区和#3 台区储能车必须互换。发现即使原来在#3 台区储能车在路上充电，再到#2 台区，剩余电量仍然小于#2 台区 16:00-20:00 电力缺口。

同样的即使#1 台区和#3 台区车辆互换，#2 台区和#4 台区车辆互换。即使在路上充电，一开始在#1 台区车辆，无法满足#3 台区 16:00-20:00 电力缺口。

通过分析，我们发现 4 辆车无法满足用电需要。

此外，还可以证明存在不止一种平凡解使得调度 5 辆的移动储能车，能够保障所有用户。其中一种平凡解的分配方案如下。

假如初始时 1 号，2 号大型车辆分配到#1 台区，3 号大型车辆分配到#2 台区，4 号大型车辆分配到#3 台区，5 号大型车辆分配到#4 台区。假定所有车辆，严格按照电力缺口发电，不多供电。1 号车按电量缺口，在 9:00-13:00 发电，在 14:00-15:00 到快充电站充电，15:30 左右接替#2 台区的 3 号大型车辆供电。3 号在 15:30-17:00 时到充电站充电，并前往#3 台区，接替 5 号大型车辆。5 号大型车辆在充电满电后，能在不晚于 19:00 点前回到车场。剩下车辆在 20:00 点前往 4 个最近充电站充电，都可以在 22:00 点前返回车场。

表 15 各个台区最少需要车辆（使用 MV#2 车）

	#1 台区	**#2 台区**	**#4 台区**	**#3 台区**
MV#2	2	1	1	1

通过上述分析，我们认为要满足所有用户用电需求，至少需要 5 辆 MV#2 移动储能车。

4.4 满足一、二类用户用电需求的车辆调度方案

通过上述分析，在确定车辆移动方向的前提下，目前方案需要考虑三个决策问题，下面对这些决策变量选取加以分析：

问题 1：1 号车从#1 台区前往#4 台区期间，是否需要利用空余时间补充剩余电量？

需要补充剩余电量。首先，从#1 台区变直接到达#4 台区变的路程为 11.3km，而途径充电站 4 到达#4 台区的路程为 14.6km，期间 8 分钟以 400kW 的满功率进行充电，可以补充 53.33kWh 电量。14：00-15：00 的充电电价为 0.687（¥/kWh），而放电电价 1.155（¥/kWh），因此获得的额外利润为 24.96，扣去路程额外成本 6.6，获得额外收益为 18.36。

问题 2： 1 号车从#4 台区变前往#3 台区变放电前是否需要充电，以及从成本角度考虑等待多久开始充电？

首先，需要对 1 号车进行充电。1 号车在从#4 台区变前往#3 台区变的过程中，1 号车的电量余额约为 100kWh，不足以完成#3 台区的 15：00-19：00 供电需求，需要前往增加路程最少的充电站 3 进行充电。第二，1 号车前往#3 台区变中距离不需要进行等待立即充电，因为可以进行充电的时间窗口 30 分钟。光伏充电相比快速充电价格更低廉，而 15：00-17：30 的光伏发电功率是递减的，因此应该达到充电站 3 后立即使用光伏充电。

问题 3：1-3 号车辆应该何时结束放电，等待多长时刻开始充电？

由于在 16：00-21：00 期间放电价格为 1.155（¥/kWh），而充电价格 18：00-21：00 为 0.8（¥/kWh），21：00-22：00 为 0.687（¥/kWh）。为了创造更高的收益，需要尽可能地放完全部的电，再前往进行充电。由于 21：00-22：00 电价，因此车辆需要在能够全部回到车场的情况下，尽可能晚地回到车场。

4.5 满足所有用户用电需求的车辆调度方案

满足所有用户用电需求时，由于小车可以多次充放电，由于太阳能发电功率不同，充电成本也时刻在变化，车辆决策空间复杂。首先需要分析数据，增加新约束，减少决策空间。目标函数以及约束与第一问相同，无需改进模型。由于决策空间变化，我们重新调整编码方式和解码方式，最后沿用第一问的改进遗传算法求解。

4.5.1 提出新约束条件

表 16 各个时段台区变电量缺口

用电缺口时段\台区变	#1 台区	#2 台区	#4 台区	#3 台区	总缺电情况
9:00-10:00	0.04	0	0	0	0.04
10:00-11:00	**0.073**	0	0	0	0.073
11:00-12:00	**0.074**	0	0	0	0.074
12:00-13:00	**0.078**	0.045	0.032	**0.085**	0.24
13:00-14:00	**0.069**	0.042	0.032	**0.07**	0.213
14:00-15:00	**0.059**	0.05	0	**0.072**	0.181
15:00-16:00	0.044	0.044	0.037	0	0.125
16:00-17:00	0.0474	**0.085**	0.037	0.036	0.2054
17:00-18:00	0.0329	**0.08**	0.022	0.05	0.1849
18:00-19:00	0.04	**0.072**	0.028	0.05	0.19
19:00-20:00	0	**0.082**	0.032	0.04	0.154

总和	0.5573	0.5	0.22	0.403	1.6803

假定 1 号 MV#2 车分配至 1 台区，2 号 MV#2 车分配至 2 台区，3 号 MV#2 车分配至 3 台区，4 号 MV#2 车分配至 4 台区，5 号 MV#2 车灵活机动。

（一）分析 4 号车辆

4 台区总用电缺口小于 4 号车电池容量。因此不需要其他车辆前往接替 4 台区。如果其他车辆前往 4 台区接替 4 号车，浪费路程上花费，浪费时间，可能导致其他台区车辆电池用完，无车接替情况。

在确定 4 台区无车前来接替假设下，为了使 4 号车利润最大化，在有较多剩余电量情况下。剩余电量应该在高电价时即 16 时-21 时，尽可能放电。考虑 21 时-22 时充电价格较低，4 号车在保证 22 时回到车场前提下，尽可能利用 21 时-22 时低电价充电。

（二）分析 3 号车辆

3 台区总用电缺口略微大于 3 号车电池容量。15:00-16:00 时无需供电，3 号车可以前往最近充电点 3 号充电站 22 节点充电。因此也无需其他车辆接替 3 号车。若有其他车辆接替 3 号车，浪费路程上花费，浪费时间，可能导致其他台区车辆电池用完，无车接替情况。

考虑 3 车辆到达充电站后是否需要马上充电？1.15 时-16 时，3 号光伏发电量逐渐下降，在电网价格不变时，充电成本上升，车辆要尽快充电。2.尽快充电可以充尽可能多的电，15-16 时电价低，多充电，在 16 时-21 时高电价时，可以多放电，获得更多收益。

其余同 4 号车，我们规划 3 号车在保证 22 时回到车场前提下，尽可能利用 21 时-22 时低电价充电。

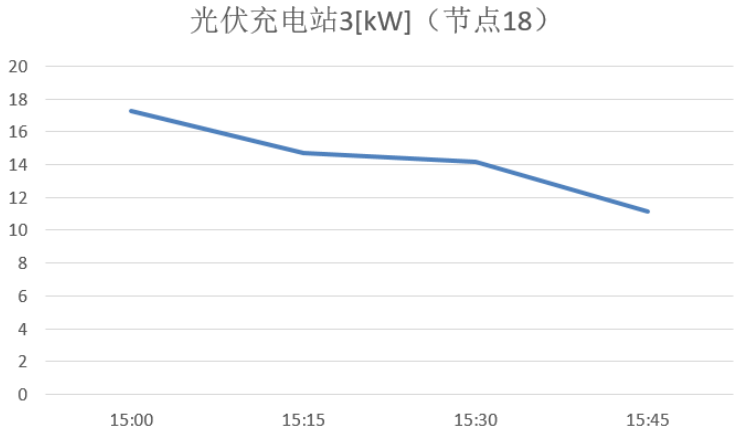

图 12 15:00-16:00 时 3 号光伏发电量变化情况

（三）分析 1、2、5 号车辆

1 号台区总用电缺口 0.5573MWh 大于 1 号车辆电池 0.4MWh，2 号台区总用电缺口 0.5MWh 大于 2 号车辆电池 0.4MWh。用电缺口没有空窗期，需要其他车辆接替。因此我们假设 1、2、5 号车辆在 1 台区和 2 台区之间灵活调度，在满足需求时获得最大收益。

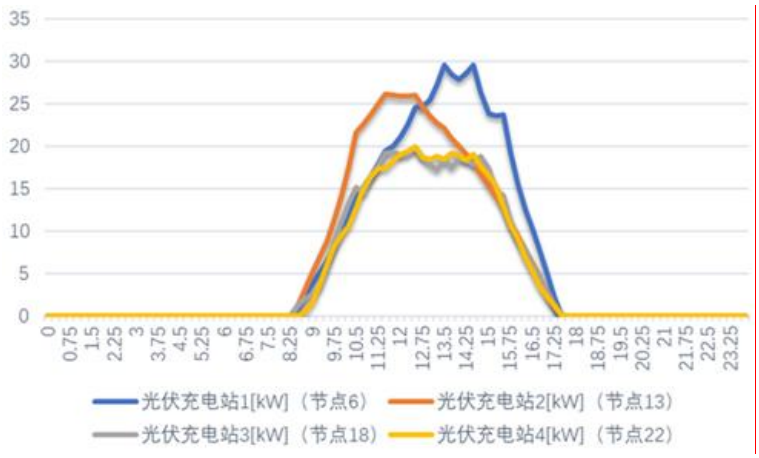

图 13 各个电站光伏发电情况图

从 1 号台区变前往 2 号台区变，最短路线为 5-6-9-24-14。需要经过 6 号充电节点。为了便于后续车辆移动，我们假定 1、2、5 号车没有电时，都会前往 6 号节点充电。6 号节点光伏发电功率相对较高，电价也相对优惠，可以获得更多收益。

（四）小结

1）1、2、5 号车辆在 1 台区和 2 台区之间灵活调度，在满足需求时获得最大收益，为我们接下来求解核心问题提供依据。

2）3 号车在 15 时-16 时尽快到达 22 节点充电，在保证 16 时回到 3 号台区前提下，尽可能多充电。在 16 时-21 时高电价时，尽可能多放电。全部电量放完后，在保证 22 时回到车场前提下，尽可能利用 21 时-22 时低电价充电。

3）4 号车有较多超过缺口的剩余电量，剩余电量应该在高电价时即 16 时-21 时，尽可能放电。在保证 22 时回到车场前提下，尽可能利用 21 时-22 时低电价充电。

4.5.2 模型建模求解

核心问题是规划 1、2、5 号车辆调度方案，该问题与第一问相同，**问题建模和求解我们与第一问模型和求解方案相同。**接下来我们重点说明编码方式和解码方式。

（一）问题编码设计

我们考虑到车辆每时刻都需要决策放电功率，是否移动到充电站充电，以及是否回车场。问题空间较为复杂，我们做出以下约束：由于车辆可以多次充电，充电功率 400W，充电速度快。为了获得更多收益，以及保证车辆回到车场，我们**假定在 20 时 40 分前，车辆会选择耗尽全部电量再充电，每次充电都将电池充满。在 20 时 40 分若车辆还在放电的，会立即断电，前往最近车场补充电量后，回到车场。**

移动方向，车辆在台区变需要选择去哪个充电站充电都与车辆的放电功率有关，因此对车辆每小时的放电功率进行编码，以及 5 号车初始前往的台区变进行编码，编码方案如下：

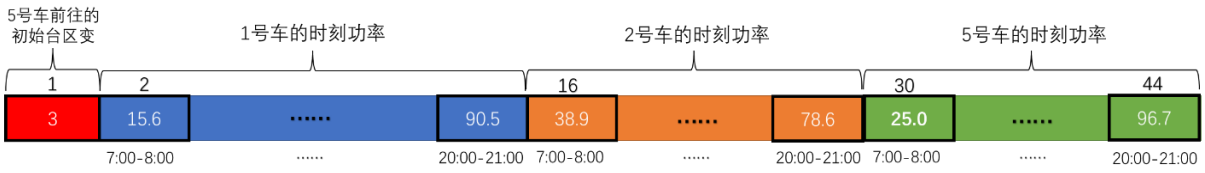

图 14 车辆移动与功率遗传信息编码

如图所示，问题（2）的遗传信息编码为 44 位。第 1 位为 5 号车在起始状态时选择前往的台区变号。每辆车具有 14 位长度的放电功率编码，以 1 号车为例，第 2 位为 7:00-8:00 时间段的功率，第 3 为 8:00-9:00 时间段，依此类推直到 20:00-21:00。第 2-15 位对应 1 号车运行时刻的放电功率，第 16-29 位对应 2 号车运行时刻的放电功率，第 30-44 位对应 5 号车运行时刻的放电功率。

（二）问题解码设计

以编码【[1] [100,97,96,93,95，……，99,100] [54,37,64,95,22，……，10,13] [70,64,44,25,37，……，14,23]】为例说明我们解码过程。

步骤一：确定放电顺序

编码假设 5 号车前往 1 号台区变。我们假定 1,2 号车先放电，放电放完后 5 号车补充。之后若某台区变出现车辆等待情况，等之前车辆全部电量耗尽后，等待车辆再接着放电。

步骤二：确定车辆实际功率

为了保证在 22 时车辆回到车场。在 20 点 40 分前：

[100,97,96,93,95，……，99,100]表示车辆 1 在 7 点-8 时，在台区变以 100w 放电，97 表示若在台区变则以 97w 放电。放电功率大于实际需求功率，无需调整。

[54,37,64,95,22，……，10,13]表示车辆 2 在 7 点-8 时，在台区变以 54w 放电，若此时台区变实际电力缺口为 80w，则会按照两者较大值，即 80w 放电。

[70,64,44,25,37，……，14,23] 表示车辆 5 在 7 点-8 时，在台区变以 70w 放电，此时车辆在排队等待，实际功率为 0。车辆在充电、行驶、排队等待，实际功率都为 0。

在在 20 点 40 分后，实际功率为 0。

步骤三：确定车辆移动方向

若车辆在台区变：在电量耗尽时刻判断。若在 20 时前，车辆按实际功率放电完成后，都会前往 6 号节点，1 号充电站充电。若在 20 时后，在 2 号台区变车辆前往最近的 13 号节点，2 号充电站充电。在 1 号台区变车辆前往最近的会前往 6 号节点，1 号充电站充电。

车辆在充电站：在电量完全充满时刻判断。

1.查询 1、2 台区变车辆此时电池剩余容量。

2.根据车辆编码放电功率计算电量耗尽时刻。

3.若一号台区变耗尽时刻都在 19 时后和二号台区变耗尽时刻都在 20 时后，则车辆会选择最近路线会车场。

4.否则车辆会前往最早耗电完的车辆所在的台区变。

4.6 结果分析

4.6.1 满足一、二类用户用电需求的车辆调度方案

通过之前理论分析和证明，我们得到车辆运动轨迹如下表所示，总收益 256.89。

表 17 满足一、二类用户用电需求的车辆运行表

	7 时-14 时	14 时-15 时
1 号车	经 1-4-5，到达一号台区变	在 14:22 由一号台区变出发，经 5-8-23-22 到充电站 4，充电 8 分钟，经 22-21-19，于 15 时到达四号台区变
2 号车	经 1-4-7-21-19，到达四号台区变	14 时取最短路径 19-21-7-4-5 前往一号台区变，14:22 到达一号台区变
3 号车	经 1-4-7-21-22-27，到三号台区变	三号台区变

	15 时-16 时	16 时-22 时
1 号车	15 时 13 分由四号台区变出发，经 19-18 到达三号台区变，在充电 32 分 24 秒，经 18-27 于 16 时到达三号台区变	20 时 37 分由三号台区变出发，经 27-22，到 22 节点充电，22-21-7-4-1。22 时到达车场
2 号车	一号台区变	20 时 48 分由由一号台区变出发，经 5-6，到 6 节点充电，6-5-2-1。22 时到达车场
3 号车	15 时取最短路径 27-18-19 由三号台区变前往四号台区变，中途不充电，15 时 13 分 48 秒到达四号台区	20 时 27 分由由四号台区变出发，经 19-18，到 18 节点充电，18-19-21-7-4-1。22 时到达车场

4.6.2 满足所有用户用电需求的车辆调度方案

运行相关算法画出收益随种群代数增长曲线如上图所示,可以看出初代种群的收益基本没有，随着种群代数增加，收益能够快速增长，在种群代数到达大约 200 代的时候，收益逐渐收敛并达到**最大值 332.56**，证明了算法的有效性和稳定性。

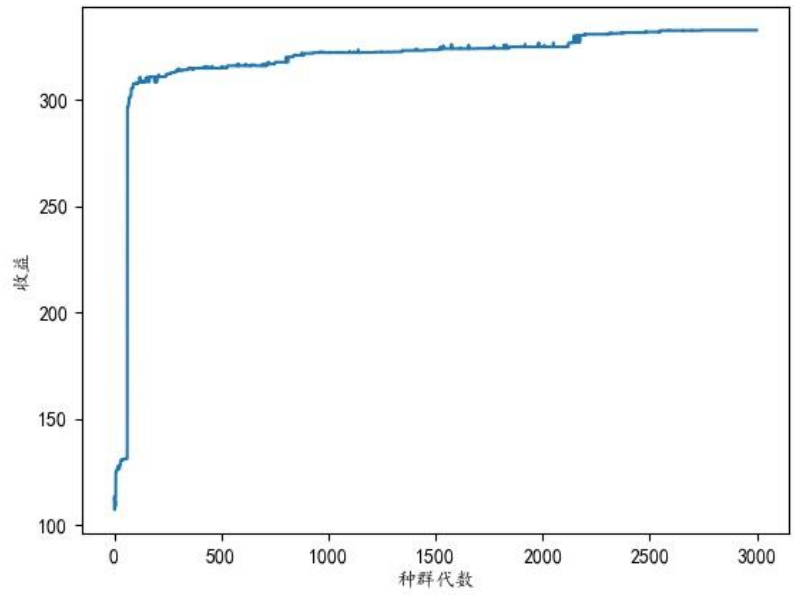

图 15 遗传算法收益迭代曲线

以下为车辆实际路线轨迹：

表 18 车辆移动轨迹

	7 时-12 时	12 时-14 时
1 号车	一号台区变	一号台区变出发，到一号充电站(6 号节点)充电，在前往二号台区变
2 号车	二号台区变	二号台区变出发，到一号充电站充电，在前往一号台区变
3 号车	三号台区变	三号台区变
4 号车	四号台区变	四号台区变
5 号车	一号台区变	一号台区变

	16 时-18 时	20 时-22 时
1 号车	二号台区变出发，到一号充电站充电，在前往一号台区变	一号台区变经一号充电站充电返回车场
2 号车	一号台区变	一号台区变经一号充电站充电返回车场
3 号车	三号台区变	三号台区变经四号充电站充电返回车场
4 号车	四号台区变	四号台区变经三号充电站充电返回车场
5 号车	一号台区变出发，到一号充电站充电，在前往二号台区变	二号台区变经二号充电站充电返回车场

	15 时-16 时	16 时-22 时
1 号车	15 时 13 分由四号台区变出发，经 19-18 到达三号台区变，在充电 32 分 24 秒，经 18-27 于 16 时到达三号台区变	20 时 37 分由三号台区变出发，经 27-22，到 22 节点充电，22-21-7-4-1。22 时到达车场
2 号车	一号台区变	20 时 48 分由由一号台区变出发，经 5-6，到 6 节点充电，6-5-2-1。22 时到达车场
3 号车	15 时取最短路径 27-18-19 由三号台区变前往四号台区变，中途不充电，15 时 13 分 48 秒到达四号台区	20 时 27 分由由四号台区变出发，经 19-18，到 18 节点充电，18-19-21-7-4-1。22 时到达车场

5 问题三模型建立与求解

在问题一中，车辆到达时间、充电时长等信息都是可计算的、确定的。在这样假定下，所做的路径安排也是相对确定的。然而在问题三中，车辆到达、充电时长、台区变负荷需求都是概率事件，成为了不确定的因素，车辆在路径上的耗费时间也因为加入了拥堵因素变得更加复杂。这类问题称为不确定旅行及服务时间下的车辆调度问题(SVRPTW)。相较于 VRP 问题，SVRPTW 问题的求解难度更高、涉及的知识面更广[3][4]。

根据题目可知车辆到达充电站和充电时长分别服从泊松分布和负指数分布，假设时间窗为硬时间窗约束，即车辆在任一充电站节点的开始充电时间必须位于对应充电站的充电时间窗范围内。由于车辆在任一充电站的到达时间间隔、充电站充电时长为随机变量，为了简化问题的求解难度，假设车辆到达时间间隔与充电时长是相互独立的。

为了后文叙述方便，定义相应的符号变量表示如下：

b：拥堵系数

μ：充电站单位时间充电的车辆数，平均(期望)服务率

λ：单位时间到达的车辆数

L_s：队长，系统中的车辆数（n）期望值

L_q：排队长，系统中排队等待服务的车辆数的期望值，记为L_q

W_s：逗留时间，指一个车辆在系统中的全部停留时间的期望值，记为W_s

W_q：等待时间，指一个车辆在系统中的排队等待时间的期望值，记为W_q

P_0：稳态概率

ρ：服务强度

TT_{ij}为车辆从节点i至节点j的行驶时间，

ST_i表示车辆在充电站i处的服务时间，

AT_i表示车辆在充电站i处的到达时间，

WT_i表示车辆在充电站i处等待服务的时间，

SS_i表示车辆在充电站i处的开始服务时间，

充电站i处对应的服务时间窗口为$[bi, ei]$。

其中$AT_j = AT_i + WT_i + ST_i + TT_{ij}$，$SS_i = \max\{AT_{ii}, bi\}$，$WT_i = \max\{bi - AT_i, 0\}$

5.1 问题建模

5.1.1 车辆到达模型

车辆到达服从泊松分布，假设储能车到达的平均速率为λ，那么单位时间内到达 n 辆车的概率为：

$$p\{X = n\} = \frac{\lambda^n}{n!}e^{-\lambda}, n = 0,1,2,\cdots, \lambda > 0$$

将上式参数λ引入时间因素t，即将λ换为λt，得到

$$p_n(t) = \frac{(\lambda t)^n}{n!}e^{-\lambda t}, t > 0, n = 0,1,2,\cdots$$

表示长为t的时间区间内到达n辆储能车的概率为$p_n(t)$，设t时间内到达车辆数为随机变量$N(t)$，则期望和方差

$$E[N(t)] = \lambda t, D[N(t)] = \lambda t$$

假设平均一个小时内到达两辆车，即$\lambda = 2$，利用实现的泊松分布随机数生成器生成十个车辆到达的时间，用于模型实现。设置相同的随机数种子即可重复试验。

5.1.2 充电时长模型

充电站的充电时长服从负指数分布，也就是充电站在单位时间内时间充电平均次数为固定值λ的事件的时间间隔的概率满足：

$$p(\mathrm{X} \leq \mathrm{t}) = 1 - e^{-\lambda t}, X \geq 0$$

其时间间隔变量的期望和方差都为$\frac{1}{\lambda}$

同样利用实现的负指数分布随机数生成器生成十个充电时长，用于后续的模型实现。设置相同的随机数种子即可重复试验。

5.1.3 台区变负荷模型

台区变负荷需求随时间的变化服从正态分布，假设台区变需求取[0,1]之间的浮点数，单位为 MW，时间变量的期望设为μ，方差为σ^2。则台区变需求$f(x)$与时间t的函数关系式为：

$$f(t) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(t-\mu)^2}{2\sigma^2}\right)$$

利用正态分布生成随机数的方法目前有很多，如 rejection sampling 方法、inverse CDF 方法、Box-Muller 算法等等。本文采用最简单的拒绝采样法，设计合适的时间期望和方差，可以生成台区变在每个时间段的需求，用于后续模型的求解。在本题中μ可以认为是台区变负荷需求集中的时间，结合问题背景，本文取$\mu = 12$。

5.1.4 充电站排队等候模型

根据题目假设，车辆到达充电站和充电时长分别服从泊松分布和负指数分布，又由于每一个充电站各自都会有车辆到达，各个充电站之间相互独立，因此对于一个充电站的排队模型，满足 M/M/1 模型。

$$\rho = \frac{\lambda}{\mu}$$

$$P_0 = 1 - \rho$$

$$P_n = \rho^n(1-\rho)$$

$$L_s = \frac{\lambda}{\mu - \lambda} = \frac{\rho}{1-\rho}$$

$$L_q = \frac{\lambda^2}{\mu(\mu-\lambda)} = \frac{\rho^2}{1-\rho} = L_s\rho$$

$$W_s = \frac{1}{\mu - \lambda}$$

$$W_q = \frac{\lambda}{\mu(\mu-\lambda)} = W_s\rho$$

借鉴问题一的背景实际，充电站给高功率储能车充电需要一个小时，低功率储能车充电需要半个小时，取平均为 45 分钟。假设车辆平均每小时到达 1 辆，充电站充电时长为 45 分钟，有

$$\lambda = 1\ 辆/h, \mu = \frac{60}{45}\ 辆/h = \frac{4}{3}\ 辆/h$$

服务强度为

$$\rho = \frac{\lambda}{\mu} = 0.75$$

稳态概率为

$$P_0 = 1 - \rho = 0.25$$

计算充电站排队系统主要工作指标：

$$L_s = \frac{\lambda}{\mu - \lambda} = 3\ 辆$$

$$L_q = L_s\rho = 2.25\ 辆$$

$$W_s = \frac{1}{\mu - \lambda} = 60\text{min}$$

$$W_q = W_s\rho = 45min$$

即系统中平均车辆数为 3 辆，排队等待充电的平均车辆数为 2.25 辆，车辆在系统中平均停留时间为 60 分钟，车辆在队列中平均等待时间为 45 分钟。

5.1.5 道路拥堵模型

问题三在问题一的基础上增加了道路拥堵情况，引入了拥堵系数b，不同的拥堵程度会在不同程度上影响车行驶的实际速度。

$$V_{实际} = bV$$

题目给出了不同拥堵程度下的拥堵系数分布。本文假设拥堵系数分布服从均匀分布，通过均匀采样获得每个区间的拥堵系数值。根据实际的行驶速度可以计算出相应的行驶时间。例如，在每个区间进行一次均匀采样后得到城市拥挤系数在每个时间区间内的取值如下表：

表 19 车辆拥堵情况

拥堵系数	时间段
0.83	22:00-8:00
0.54	14:00-17:00
0.41	10:00-12:00
0.19	12:00-14:00,19:00-22:00
0.05	8:00-10:00,17:00-19:00

5.2 模型实现

在实际实现过程中，由于队列是动态变化的，也就导致了每一辆储能车到达充电站等待的时间会不一样。为了更真实的描述储能车到达充电站的排队模型，采用离散事件仿真的方法来对排队模型进行描述和实现。在离散事件仿真中常用的方法有事件调度法、三段扫描法以及进程交互法，本文采用更易维护，建模简单的三段扫描法。在仿真实现之前进行 uml 建模。

5.2.1 建立静态模型

静态模型包括 Entity（实体类）、EventList（事件列表类）、充电站排队模型（系统仿真类）、PriorityEntityQueue（优先级队列类）、Energy vehicle（储能车类）、Charging station（充电站类）。

实体类是最核心的一个类，是仿真对象的封装。储能车类和充电站类是本题中对实体的一个泛化；系统仿真类是整个仿真模型的封装，在仿真过程中不断地调度事件到各个仿真对象上进行处理，因此和实体类是关联的关系；事件列表类是根据本课题用的三段扫描法的调度规则，在仿真过程中插入、删除实体并进行排序，是实体类的组成部分，同时也跟实体关联；优先级队列类是队列类的一种继承，在实体进入队列时根据其属性的优先级不同，进入优先级队列，在这个队列里优先级高的先服务。优先级队列类为实体类提供服务，因此这两个类之间为使用依赖关系。

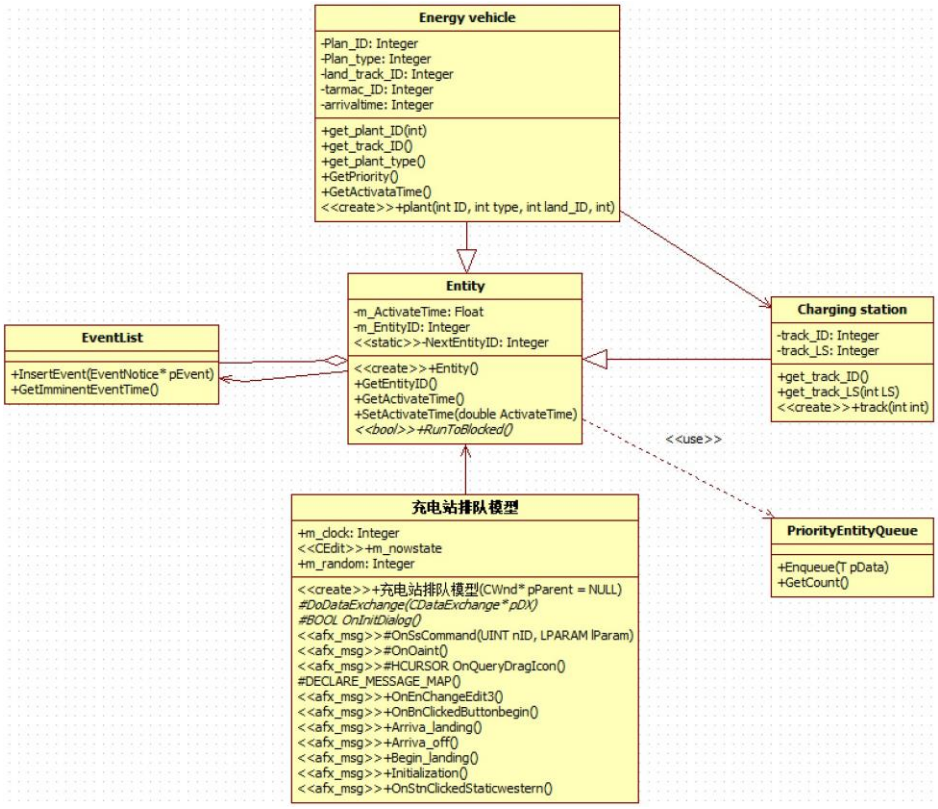

图 16 充电站排队模型类图

5.2.2 建立动态模型

(一) 顺序图

顺序图描述了实体之间交互的顺序过程。当储能车到达一个充电站时，与充电站交互的顺序图如下：

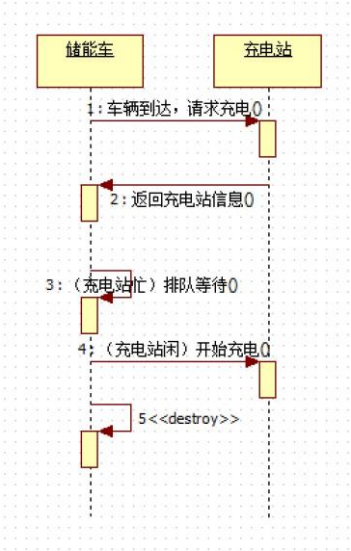

图 17 排队模型顺序图

（二）活动图

三段扫描法将事件分为 B 例程和 C 例程。其中 B 例程是主事件，例如实体到达事件、结束事件；C 例程是条件事件，例如一段活动的开始。在三段扫描法中，仿真器在运行过程中会每次同时扫描所有的 B 例程和 C 例程，从而进行仿真的推进。

根据三段扫描法可以将充电站排队问题分为以下三个事件：车辆到达（B 例程）、开始充电（C 例程）、充电结束（B 例程）。

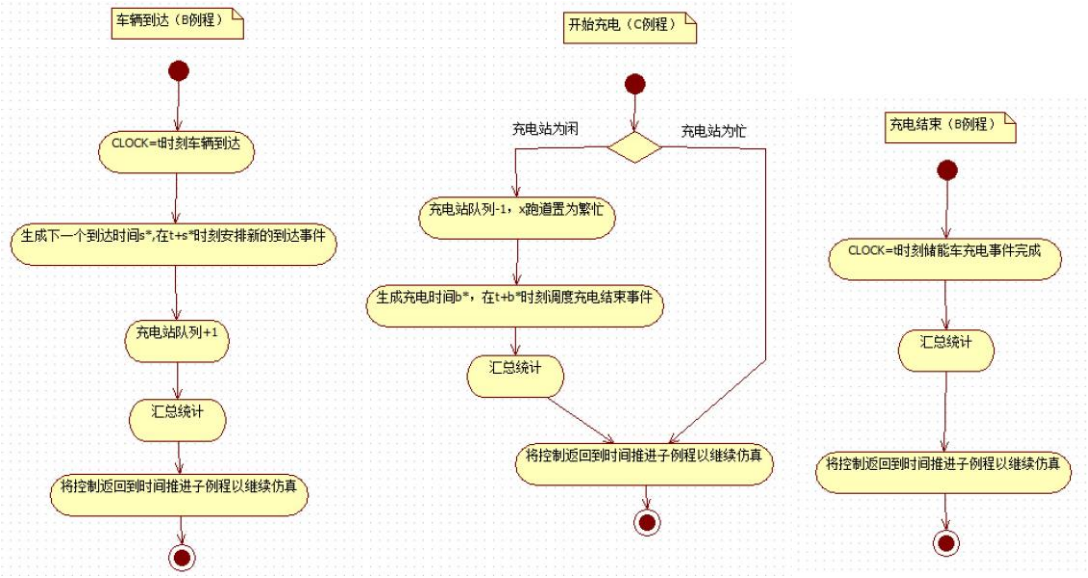

图 18 三段扫描法活动图

生成下一个到达时间和充电时间时利用前文所述的泊松分布和负指数分布随机数生成器生成。仿真结果记录储能车 k 到达充电站 i 进行充电的等待时间WT_{ki}和充电时长ST_{ki}，用于算法实现。

5.3 算法实现

算法沿用问题一的遗传算法框架，在问题一的基础上，有以下改动：

1）台区变的每小时负荷需求数据利用正态分布的拒绝采样法生成的数据进行算法实现；

2）计算收益时加上了拥堵系数，利用不同时间段拥堵系数的分布均匀采样获取拥堵系数来计算实际行驶速度、时间以及成本；

3）充电站增加了车辆到达，导致储能车到充电站时可能会进入队列等待。利用充电站排队的仿真模型输出记录的储能车到达充电站的充电等待时间以及充电时长进行算法实现。

5.4 结果分析

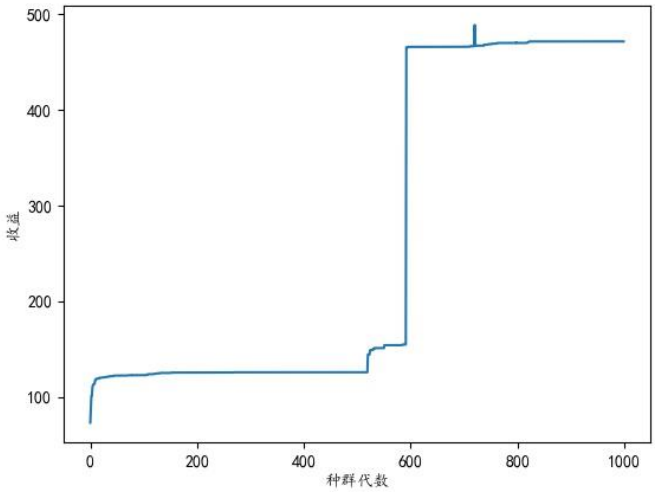

图 19 运行结果表格

取泊松分布的λ为 1，负指数分布的μ为 1.33，正态分布均值取 12，方差取 1。运行仿真以及遗传算法，迭代 1000 次，画出收益随种群代数增长曲线如上图所示，可以看出随着种群代数增加，收益能够快速增长，在种群代数到达大约 600 的时候，收益逐渐收敛并达到最大值 471.56，证明了算法的有效性和稳定性。

6 模型评价

6.1 模型的优点

(1) 模型充分分析数据特征，围绕利润最大化目标，和保证用户用电缺口以及按时返回车场等两个关键约束，按尽可能放电原则将增加新约束，将问题化简。

(2) 模型抓住储能车输出功率这一问题的重要因素，将复杂的车辆调度问题化简。模型的输出结果符合题目要求，能解决实际问题。

(3) 本文使用模型复杂度低，使用的改进遗传算法具有收敛速度快、评价值高等优点。

(4) 本文得到的方案能够最大化利用储能车的储能，削峰填谷，改善电力质量，提升用户供电服务满意度。能够充分利用时间，对不同电价，不同用电缺口，不同突发因素，做出恰当的应对方案，具有利润高，鲁棒性好等特征。

6.2 模型的不足

(1) 模型中有许多参数，如交叉率和变异率，并且这些参数的选择严重影响解的品质，而目前这些参数的选择大部分是依靠经验，且未进行参数敏感性分析。

(2) 算法的并行机制的潜在能力没有得到充分的利用。

(3) 模型对不确定性因素建模不足，实际环境中环境电力需求变化较快，且难以准确预测，且未考虑储能设备充放电行为对电力系统潮流分布的影响。

6.3 模型的推广

本文设计的遗传算法框架适用于车辆调度优化问题、考虑利润的旅行商问题以及不确定旅行及服务时间下的车辆调度问题。

参考文献

[1] [最短路径问题]—Dijkstra 算法最详解 https://zhuanlan.zhihu.com/p/129373740

[2] 遗传算法入门详解 https://zhuanlan.zhihu.com/p/100337680

[3] DouglasMoura Miranda&Samuel Vieira Conceição(2016).The vehicle routing problemwith hard time windows and stochastic travel and service time.Expert SystemsWith Applications,64,104-116.

[4] Xiangyong Li, Peng Tian&Stephen C.H.Leung b(2010).Vehicle routing problems with time windows and stochastic traveland service times:Models and algorithm.Int.J.Production Economics,125,137-145.

[5] 锦标赛选择算法及 matlab 实现 https://blog.csdn.net/future12356

[6] 冯国双. 白话统计[M]. 电子工业出版社, 2018.

[7] 张良均. Python 数据分析与挖掘实战[M]. 机械工业出版社, 2016.

[8] 茆诗松, 程依明, 濮晓龙, 等. 概率论与数理统计教程第二版[M]. 北京: 高等教育出版社, 2011

[9]《运筹学》教材编写组. 运筹学.第 4 版[M]. 清华大学出版社, 2012.

[10] 周志华. 机器学习[M]. 清华大学出版社, 2016.

[11] Rachel Schutt, Cathy O'Neil. 数据科学实战[M]. 人民邮电出版社, 2015.

[12] 姜启源, 谢金星, 叶俊. 数学模型.第 4 版[M]. 高等教育出版社, 2011.

[13] 韩中庚. 数学建模方法及其应用-第 2 版[M]. 高等教育出版社, 2009.

[14] 自适应大领域搜索算法(ALNS) 详解及 python 示例 https://blog.csdn.net/weixin_46651999

附　录

附录 I: 主要程序/关键代码

环境	操作系统: Windows (Version 11.22000.856) 编程语言: Python 3.8 (Anaconda Navigator 1.9.2) 编辑器: PyCharm 2018.3.2 (Professional Edition)

代码清单 1 Dijkstra 计算最短路径和路径长度

```
import pandas as pd
class Dijkstra:
    def __init__(self, graph, start, goal):
        self.graph = graph          # 邻接表
        self.start = start          # 起点
        self.goal = goal            # 终点

        self.open_list = {}         # open 表
        self.closed_list = {}       # closed 表

        self.open_list[start] = 0.0         # 将起点放入 open_list 中

        self.parent = {start: None}         # 存储节点的父子关系。键为子节点，值为父节点。方便做最后路径的回溯
        self.min_dis = None                 # 最短路径的长度

    def shortest_path(self):

        while True:
            if self.open_list is None:
                print('搜索失败，结束！')
                break
            distance, min_node = min(zip(self.open_list.values(), self.open_list.keys()))          # 取出距离最小的节点
            self.open_list.pop(min_node)
    # 将其从 open_list 中去除

            self.closed_list[min_node] = distance                           # 将节点加入 closed_list 中

            if min_node == self.goal:                                       # 如果节点为终点
                self.min_dis = distance
                shortest_path = [self.goal]                                 # 记录从终点回溯的路径
                father_node = self.parent[self.goal]
                while father_node != self.start:
                    shortest_path.append(father_node)
                    father_node = self.parent[father_node]
                shortest_path.append(self.start)
                print(shortest_path[::-1])                                  # 逆序
                print('最短路径的长度为：{}'.format(self.min_dis))
                print('找到最短路径，结束！')
                return shortest_path[::-1], self.min_dis          # 返回最短路径和最短路径长度

            for node in self.graph[min_node].keys():                        # 遍历当前节点的邻接节点
                if node not in self.closed_list.keys():                     # 邻接节点不在 closed_list 中
                    if node in self.open_list.keys():                       # 如果节点在 open_list 中
                        if self.graph[min_node][node] + distance < self.open_list[node]:
                            self.open_list[node] = distance + self.graph[min_node][node]
  # 更新节点的值
                            self.parent[node] = min_node                    # 更新继承关系
```

```
                    else:                                                    # 如果节点不在 open_
list 中
                        self.open_list[node] = distance + self.graph[min_node][node]
  # 计算节点的值，并加入 open_list 中
                        self.parent[node] = min_node                         # 更新继承关系


if __name__ == '__main__':
    mgraph = pd.read_excel('D:\XHT\湖南省数模\B题\distance_matrix.xls', header=None)

    g = {'1': {'2': 2, '4': 1.8},
         '2': {'1': 2, '3': 2.2, '5':1.7},
         '3': {'2': 2.2, '6': 1.6, '11':3.3},
         '4': {'1': 1.8, '5': 1.9, '7': 2.3},
         '5': {'2': 1.7, '4':1.9, '6':2.1, '8':2.7},
         '6': {'3': 1.6, '5': 2.1, '9': 2.7, '10':2.6},
         '7': {'4': 2.3, '8':2.8, '21':3},
         '8': {'23': 2.8, '9': 2.8, '7': 2.8, '5': 2.7},
         '9': {'6': 2.7, '24': 2.1, '10': 2.2, '8': 2.8},
         '10': {'11': 1.9, '13': 4, '9': 2.2, '6': 2.6},
         '11': {'12': 4, '10': 1.9, '3': 3.3},
         '12': {'15': 6.6, '13': 1.8, '11': 4},
         '13': {'12': 1.8, '10': 4, '14': 3.6},
         '14': {'15': 1.2, '24': 3, '20': 2.2, '13': 3.6},
         '15': {'12': 6.6, '14': 1.2, '16': 5.8},
         '16': {'17': 9.2, '29': 3.4, '15': 5.8},
         '17': {'16': 9.2, '18': 2.6, '28': 5.3},
         '18': {'19': 2.4, '17': 2.6, '27': 4.5},
         '19': {'18': 2.4, '21': 4.1},
         '20': {'14': 2.2, '25': 2.8,'29':4.4},
         '21': {'7': 3, '19': 4.1, '22':2.3},
         '22': {'23': 2.7, '21': 2.3, '27':2.3},
         '23': {'8': 2.8, '22': 2.7, '24':3.6, '26':2.2},
         '24': {'9': 2.1, '14': 3, '23':3.6, '25':3.7},
         '25': {'20': 2.8, '24': 3.7, '26':2.2,'29':1.4},
         '26': {'23': 2.2, '25': 2.2, '27':2.2, '28':1.5},
         '27': {'18': 4.5, '22': 2.3, '26':2.2,'28':1.8},
         '28': {'17': 5.3, '26': 1.5, '27':1.8, '29':2.2},
         '29': {'16': 3.4, '20': 4.4, '25':1.4, '28':1.2},
         }
    start = '19'
    goal = '5'
    dijk = Dijkstra(g, start, goal)
    dijk.shortest_path()
```

代码清单 2 移动储能车解码器

```
# encoding: utf-8

import json
import math
import numpy as np
import traceback
import logging
import pandas as pd
import copy

from math import radians, cos, sin, asin, sqrt

'''
```

```
随机输入变量，将变量分别放入zhantai，大车，小车
'''
class Fitness:
    def __init__(self, code):
        dis = np.load('dis.npy')
        self.need_ele_data = pd.read_excel("need_elect.xlsx").values
        self.taizhan_wait = code[:4]#np.random.rand(4) todo
        self.dache = []
        self.xiaoche = []
        for i in range(5):
            self.dache.append(code[4+4*i:4+4*(i+1)])##!!todo
            self.xiaoche.append(code[24+4*i:24+4*(i+1)])#!!todo
        self.dache = np.array(self.dache)
        self.xiaoche = np.array(self.xiaoche)
        # dis0
        dis_cha = [4, 13, 18, 26]
        self.park_disch_dis = [dis[0,dis_cha[i]] for i in range(4)] #到台区变距离
        #寻找快充站，4个放电站可选择4个充电站
        cha = [5, 12, 17, 21]

        # dis_cha_dis =[]
        # for j in cha:
        #     dis_cha_dis.append([dis[j,dis_cha[code[4+4*i]]] for i in range(4)])#行：充电站，
列：放电站
        # dis_cha_dis = np.array(dis_cha_dis)+np.array([np.tile(np.array(dis[i,0]),4) for i in cha])
        # disch_cha_index = np.argmin(dis_cha_dis,axis=1) #判断哪个台区变最近
        # self.disch_cha_dis = []
        # for i in range(4):
        #     self.disch_cha_dis.append(dis[cha[i], code[4 + 4 * disch_cha_index[i]]])
        self.disch_cha_dis = [2.1, 3.6, 2.4, 2.3]

        self.cha_park_dis = [dis[cha[i],0] for i in range(4)]
        self.taiqu_chongdian = {0: [0], 1: [1], 2: [2], 3: [3]}

        self.big_sta = np.array([[0.0, 400.0, 0.0,0]] * 5)
        self.small_sta = np.array([[0.0, 200.0, 0.0,0]] * 5)

        self.endtime = 0
        # 超参数


    def time_price(self,t):
        if t < 1:
            price = 0.814
        elif t < 5:
            price = 1.155
        elif t < 9:
            price = 0.814
        elif t < 14:
            price = 1.155
```

```
        elif t < 15:
            price = 0.814
        else:
            price = 10
        return price

    def chongdian_program(self, startime, car_need):
        if startime < 5:
            start_cha = 5
        elif startime < 9:
            start_cha = startime
        elif startime < 11:
            start_cha = 11
        else:
            start_cha = startime
        endtime = start_cha + car_need / 400
        start_cha1 = -1
        end_cha1 = -1
        if endtime <= 9:
            end_cha = endtime
            cost_cha = 0.687 * car_need
        elif endtime <= 11:
            end_cha = 9
            dissat = car_need / 400 - (end_cha - start_cha)
            start_cha1 = 11
            end_cha1 = start_cha1 + dissat
            cost cha = 0.687 * 400 * (end cha - start cha) + 0.8 * 400 * (end cha1 - start ch
a1)
        elif endtime <= 14:
            end_cha = endtime
            cost_cha = 0.8 * car_need
        else:
            end_cha = endtime
            if start_cha < 14:
                cost_cha = (14 - start_cha) * 400 * 0.8 + (end_cha - 14) * 400 * 0.687
            else:
                cost_cha = 0.687 * car_need
        self.endtime = endtime
        return cost_cha, start_cha, end_cha, start_cha1, end_cha1

    def calcu(self):
        """
        更新状态，计算适应度
        :return:
        """
        '''
        从停车场到台区变，更新时间和收益
        '''
        lost = 0
        taiqu = {0: [], 1: [], 2: [], 3: []}
        for i in range(5):
```

```
            big_choice = int(self.dache[i, 0])
            small_choice = int(self.xiaoche[i, 0])
            route_cost = 2 * self.park_disch_dis[big_choice]
            t = self.park_disch_dis[big_choice] / 30
            self.big_sta[i, 2] -= route_cost
            self.big_sta[i, 0] += t
            route_cost = 1 * self.park_disch_dis[small_choice]
            t = self.park_disch_dis[small_choice] / 30
            self.small_sta[i, 2] -= route_cost
            self.small_sta[i, 0] += t
            taiqu[big_choice].append([self.dache[i, 1], "大车", i])
            taiqu[small_choice].append([self.xiaoche[i, 1], "小车", i])

        '''
        车辆在台区变的顺序
        '''
        for i in range(len(taiqu.keys())):
            temp = np.array(taiqu[i])
            if len(temp) != 0:
                temp = temp[np.argsort(temp[:, 0].astype(float))]  # 降序排列!!!!
                for j in range(len(taiqu[i])):
                    if temp[j, 1] == "大车":
                        max_power = 100
                        sta = self.big_sta
                        che = self.dache
                    else:
                        max_power = 50
                        sta = self.small_sta
                        che = self.xiaoche

                    car_id = int(temp[j, 2])
                    if j == 0:
                        sta[int(car_id), 0] += self.taizhan_wait[i]  # 完成等待
                        # print(sta[int(car_id), 0])
                    else:
                        sta[int(car_id), 0] = end_time
                    power_low = max_power - che[car_id, 2]
                    power_high = max_power - che[car_id, 3]
                    temp_time = sta[car_id, 0]
                    end = 0
                    first = 1
                    while end == 0:  # 第一个和最后一个时间段特殊
                        cha_price = self.time_price(temp_time)  # 这个时间是高电价还是低电
价
                        if cha_price == 0.814:
                            out_power = power_low
                        else:
                            out_power = power_high
                        try:
```

```
                        need_power =round( 1000 * self.need_ele_data[int(temp_time), int(i)],2 ) # 电力缺口
                    except:
                        lost = 1
                        break
                    if need_power > max_power:
                        print("输出功率不够")
                        lost = 1
                        return -10000
                        # 需要break!!!!!!!!!
                    else:
                        act_power = max(need_power, out_power)  # 实际功率为算子设定的功率和电力缺口两者中较大者
                    if first == 1:
                        first = 0
                        time0 = math.ceil(temp_time) - temp_time
                        sta[int(car_id), 3]=temp_time
                    else:
                        if act_power < sta[car_id, 1]:
                            time0 = 1
                        else:
                            time0 = sta[car_id, 1] / act_power
                            end = 1
                    cost_eng = act_power * time0
                    sta[car_id, 0] += time0
                    sta[car_id, 1] -= cost_eng
                    sta[car_id, 2] += cha_price * cost_eng
                    temp_time = sta[car_id, 0]
                end_time = copy.deepcopy(sta[car_id, 0])

        """
        检查供电需求算法都被满足
        """

        taiqu_sta_time =[2,5,5,2]
        taiqu_end_time = [12,13,13,13]
        for i in range(len(taiqu.keys())):
            temp = np.array(taiqu[i])
            if len(temp) != 0:
                temp = temp[np.argsort(temp[:, 0].astype(float))]  # 降序排列!!!!
                for j in range(len(taiqu[i])):
                    if temp[j, 1] == "大车":
                        max_power = 100
                        sta = self.big_sta
                        che = self.dache
                    else:
                        max_power = 50
                        sta = self.small_sta
                        che = self.xiaoche
```

```
                car_id = int(temp[j, 2])
                if j ==0 and sta[int(car_id), 3]>taiqu_sta_time[i]:
                    print("开始需求未被满足",i)
                    # return -3000
                if j == len(taiqu[i])-1 and sta[int(car_id), 0]<taiqu_end_time[i]:
                    print("结束需求未被满足",i)
                    # return -3000


    """
    放电完成后马上从台区变到充电站
    """
    for i in range(len(taiqu.keys())):
        temp = np.array(taiqu[i])
        if len(temp) != 0:
            temp = temp[np.argsort(temp[:, 0].astype(float))]  # 降序排列!!!!
            for j in range(len(taiqu[i])):
                if temp[j, 1] == "大车":
                    max_power = 100
                    sta = self.big_sta
                    che = self.dache
                else:
                    max_power = 50
                    sta = self.small_sta
                    che = self.xiaoche

                car_id = int(temp[j, 2])
                dis1 = self.disch_cha_dis[int(che[int(car_id), 0])]
                sta[car_id, 0] += dis1 / 30.0
                sta[car_id, 2] -= (max_power / 50) * dis1
                if sta[car_id, 0] > 15:
                    # print("未能返回车场")
                    lost = 1
                    # return -50
                    # break!!!!!!!

    chongdian = {0: [], 1: [], 2: [], 3: []}
    for i in range(5):
        big_choice = int(self.taiqu_chongdian[int(self.dache[i, 0])][0])
        small_choice = int(self.taiqu_chongdian[int(self.xiaoche[i, 0])][0])
        chongdian[big_choice].append([self.big_sta[i, 0], "大车", i])
        chongdian[small_choice].append([self.small_sta[i, 0], "小车", i])

    """
    更新在充电站充电时间和费用
    """
    for i in range(len(chongdian.keys())):
        temp = np.array(list(chongdian[i]))
        if len(temp) != 0:
            temp = temp[np.argsort(temp[:, 0].astype(float))]
```

```
                for j in range(len(taiqu[i])):
                    if temp[j, 1] == "大车":
                        car_need_power = 400
                        sta = self.big_sta
                        che = self.dache
                    else:
                        car_need_power = 200
                        sta = self.small_sta
                        che = self.xiaoche
                    car_id = int(temp[j, 2])

                    if j == 0:
                        yuji_star = sta[car_id, 0]
                    else:
                        yuji_star = max(cha_end_time, sta[car_id, 0])
                    cost_cha, start_cha, end_cha, start_cha1, end_cha1 = self.chongdian_program
(yuji_star, car_need_power)
                    if start_cha1 == -1:
                        sta[car_id, 0] = end_cha
                    else:
                        sta[car_id, 0] = end_cha1
                    sta[car_id, 1] = car_need_power
                    sta[car_id, 2] -= cost_cha
                    cha_end_time = sta[car_id, 0]
                    """
                    充电完成后马上从充电站到停车场
                    """
                    sta[car_id, 0] += self.cha_park_dis[int(i)] / 30.0
                    sta[car_id, 2] -= self.cha_park_dis[int(i)] * (car_need_power / 200)

                    if sta[car_id, 0] > 15:
                        # print("未能返回车场")
                        lost = 1
                        # return -10 # break!!!!!!!
        Total_revenue = 0
        debuff = 0
        debuff2 = 400
        for i in range(4):
            if 16 > self.big_sta[i,0]>15:
                debuff += 100*(self.big_sta[i,0] - 15)
            elif self.big_sta[i,0]>16:
                debuff += 200*(self.big_sta[i,0] - 15)
            if 16 > self.small_sta[i,0]>15:
                debuff += 100*(self.small_sta[i,0] - 15)
            elif self.small_sta[i,0]>16:
                debuff += 200*(self.small_sta[i,0] - 15)

            Total_revenue += self.big_sta[i, 2]
            Total_revenue += self.small_sta[i, 2]
        if lost != 1:
```

```
                print('可行',Total_revenue - debuff)
            # print('debuff',debuff)
            return Total_revenue - debuff - debuff2*lost


code = [0, 2, 0, 0,
            0, 0, 30, 30,
            1, 100, 50, 50,
            3,  0, 30, 30,
            3, 0, 30, 30,
            0, 0,  30, 30,
            0, 4, 30, 30,
            2, 13, 30, 30,
            2, 18, 30, 30,
            1, 24, 30, 30,
            3, 26, 30, 30]
fit = Fitness(code)
fit.calcu()

'''
[0, 1, 0, 0,
            0, 0, 0, 0,
            1, 100, 0, 0,
            3,  0, 0, 0,
            3, 0, 0, 0,
            0, 0,  0, 0,
            0, 4, 0, 0,
            2, 13, 0, 0,
            2, 18, 0, 0,
            1, 24, 0, 0,
            3, 26, 0, 0]
'''
```

代码清单 3 遗传算法

```
import random
from operator import itemgetter
from decode import Fitness
import numpy as np
from tqdm import tqdm
class Gene:
    """
    This is a class to represent individual(Gene) in GA algorithom
    each object of this class have two attribute: data, size
    """

    def __init__(self, **data):
        self.__dict__.update(data)
```

```
        self.size = len(data['data'])  # length of gene


class GA:
    """
    This is a class of GA algorithm.
    """

    def __init__(self, parameter):
        """
        Initialize the pop of GA algorithom and evaluate the pop by computing its' fitness value.
        The data structure of pop is composed of several individuals which has the form like th
at:

        {'Gene':a object of class Gene, 'fitness': 1.02(for example)}
        Representation of Gene is a list: [b s0 u0 sita0 s1 u1 sita1 s2 u2 sita2]

        """
        # parameter = [CXPB, MUTPB, NGEN, popsize, low, up]
        self.parameter = parameter
        self.result = []
        self.best_indi = []
        low = self.parameter[4]
        up = self.parameter[5]

        self.bound = []
        self.bound.append(low)
        self.bound.append(up)
        '''
        [0, 1, 0, 0,
                0, 0, 0, 0,#前两位直接定义0 0
                1, 100, 0, 0,#第一位1
                3,  0, 0, 0,第一位为3
                3, 0, 0, 0,
                0, 0,  0, 0,
                0, 4, 0, 0,
                2, 13, 0, 0,
                2, 18, 0, 0,
                1, 24, 0, 0,
                3, 26, 0, 0]
        '''

        pop = []
        for i in range(self.parameter[3]):
            geneinfo = []
            # 初始解的构造[1,1,1,1,0, 4, 99, 98,1, 13, 55, 52,2, 18, 73, 71,3, 26, 84, 69, 3, 26,
78, 82,0, 4, 99, 98,1, 13, 55, 52,2, 18, 73,
            # 开始服务时间
            # geneinfo = list(np.random.rand(4)*5)
            #
            # for i in range(5):
```

```
                #       geneinfo.append(random.randint(0, 3))
                #       geneinfo.append(random.randint(0, 20))
                #       geneinfo.append(random.randint(0, 100))
                #       geneinfo.append( random.randint(0, 100))
                # for i in range(5):
                #       geneinfo.append(random.randint(0, 3))
                #       geneinfo.append(random.randint(0, 20))
                #       geneinfo.append(random.randint(0, 50))
                #       geneinfo.append( random.randint(0, 50))


                for pos in range(len(low)):
                    if pos%4 == 0 and pos != 0:
                        geneinfo.append(random.randint(self.bound[0][pos], self.bound[1][pos]))  # ini
tialise popluation
                    else:
                        geneinfo.append((self.bound[1][pos]-self.bound[0][pos])*random.random()+self.
bound[0][pos])  # initialise popluation

                fitness = self.evaluate(geneinfo)  # evaluate each chromosome
                pop.append({'Gene': Gene(data=geneinfo), 'fitness': fitness})  # store the chromosome
and its fitness

            self.pop = pop
            self.bestindividual = self.selectBest(self.pop)  # store the best chromosome in the populati
on

        def evaluate(self, geneinfo):
            """
            fitness function
            geneinfo 带入decode中计算
            """
            # x1 = geneinfo[0]
            # x2 = geneinfo[1]
            # x3 = geneinfo[2]
            # x4 = geneinfo[3]
            fit = Fitness(geneinfo)
            y = fit.calcu()#适应度值
            return y

        def selectBest(self, pop):
            """
            select the best individual from pop
            """
            s_inds = sorted(pop, key=itemgetter("fitness"), reverse=True)  # from large to small, retur
n a pop
            return s_inds[0]

        def selection(self, individuals, k):
            """
            select some good individuals from pop, note that good individuals have greater probabilit
y to be choosen
```

```
        for example: a fitness list like that:[5, 4, 3, 2, 1], sum is 15,
        [-----|----|---|--|-]
        012345|6789|101112|1314|15
        we randomly choose a value in [0, 15],
        it belongs to first scale with greatest probability
        """
        s_inds = sorted(individuals, key=itemgetter("fitness"),
                        reverse=True)  # sort the pop by the reference of fitness
        sum_fits = sum(ind['fitness'] for ind in individuals)  # sum up the fitness of the whole pop

        chosen = []
        for i in range(k):
            u = random.random() * sum_fits  # randomly produce a num in the range of [0, sum_fits], as threshold
            sum_ = 0.001
            for ind in s_inds:
                sum_ += ind['fitness']  # sum up the fitness
                if sum_ >= u:
                    # when the sum of fitness is bigger than u, choose the one, which means u is in the range of
                    # [sum(1,2,...,n-1),sum(1,2,...,n)] and is time to choose the one ,namely n-th individual in the pop
                    chosen.append(ind)
                    break
        # from small to large, due to list.pop() method get the last element
        chosen = sorted(chosen, key=itemgetter("fitness"), reverse=False)
        return chosen

    def crossoperate(self, offspring):
        """
        cross operation
        here we use two points crossoperate
        for example: gene1: [5, 2, 4, 7], gene2: [3, 6, 9, 2], if pos1=1, pos2=2
        5 | 2 | 4   7
        3 | 6 | 9   2
        =
        3 | 2 | 9   2
        5 | 6 | 4   7
        """
        dim = len(offspring[0]['Gene'].data)

        geninfo1 = offspring[0]['Gene'].data  # Gene's data of first offspring chosen from the selected pop
        geninfo2 = offspring[1]['Gene'].data  # Gene's data of second offspring chosen from the selected pop

        if dim == 1:
            pos1 = 1
            pos2 = 1
        else:
            pos1 = random.randrange(1, dim)  # select a position in the range from 0 to dim-1,
```

```
            pos2 = random.randrange(1, dim)

        newoff1 = Gene(data=[])  # offspring1 produced by cross operation
        newoff2 = Gene(data=[])  # offspring2 produced by cross operation
        temp1 = []
        temp2 = []
        for i in range(dim):
            if min(pos1, pos2) <= i < max(pos1, pos2):
                temp2.append(geninfo2[i])
                temp1.append(geninfo1[i])
            else:
                temp2.append(geninfo1[i])
                temp1.append(geninfo2[i])
        newoff1.data = temp1
        newoff2.data = temp2

        return newoff1, newoff2

    def mutation(self, crossoff, bound):
        """
        mutation operation
        """
        dim = len(crossoff.data)

        if dim == 1:
            pos = 0
        else:
            pos = random.randrange(0, dim)  # chose a position in crossoff to perform mutation.

        # crossoff.data[pos] = random.randint(bound[0][pos], bound[1][pos])
        if pos % 4 == 0 and pos != 0:
            crossoff.data[pos] =random.randint(self.bound[0][pos], self.bound[1][pos])  # initialise
popluation
        else:
            crossoff.data[pos] = (self.bound[1][pos] - self.bound[0][pos]) * random.random() + sel
f.bound[0][
                pos]  # initialise popluation
        return crossoff

    def GA_main(self):
        """
        main frame work of GA
        """
        popsize = self.parameter[3]

        print("Start of evolution")

        # Begin the evolution
        for g in tqdm(range(NGEN)):

            print("############### Generation {} ###############".format(g))
```

```
            # Apply selection based on their converted fitness
            ################
            # if 出现20次可行解，把解规模扩大一些，
            # ###########
            # geneinfo = []
            # for pos in range(len(low)):
            #       if pos%4 == 0 and pos != 0:
            #           geneinfo.append(random.randint(self.bound[0][pos], self.bound[1][pos]))  # i
nitialise popluation
            #       else:
            #           geneinfo.append((self.bound[1][pos]-self.bound[0][pos])*random.random()+sel
f.bound[0][pos])  # initialise popluation
            #
            # fitness = self.evaluate(geneinfo)  # evaluate each chromosome
            # self.pop.append({'Gene': Gene(data=geneinfo), 'fitness': fitness})  # store the chrom
osome and its fitness
            ##################
            popsize1 = 100
            selectpop = self.selection(self.pop, popsize1)
            while len(selectpop)!= popsize1:
                selectpop = self.selection(self.pop, popsize1)

            nextoff = []
            while len(nextoff) != popsize1:
                # Apply crossover and mutation on the offspring

                offspring = [selectpop.pop() for _ in range(2)]


                # Select two individuals

                if random.random() < CXPB:  # cross two individuals with probability CXPB
                    crossoff1, crossoff2 = self.crossoperate(offspring)
                    if random.random() < MUTPB:  # mutate an individual with probability M
UTPB
                        muteoff1 = self.mutation(crossoff1, self.bound)
                        muteoff2 = self.mutation(crossoff2, self.bound)
                        fit_muteoff1 = self.evaluate(muteoff1.data)  # Evaluate the individuals
                        fit_muteoff2 = self.evaluate(muteoff2.data)  # Evaluate the individuals
                        nextoff.append({'Gene': muteoff1, 'fitness': fit_muteoff1})
                        nextoff.append({'Gene': muteoff2, 'fitness': fit_muteoff2})
                    else:
                        fit_crossoff1 = self.evaluate(crossoff1.data)  # Evaluate the individuals
                        fit_crossoff2 = self.evaluate(crossoff2.data)
                        nextoff.append({'Gene': crossoff1, 'fitness': fit_crossoff1})
                        nextoff.append({'Gene': crossoff2, 'fitness': fit_crossoff2})
                else:
                    nextoff.extend(offspring)

            # The population is entirely replaced by the offspring
            self.pop = nextoff
```

```
            # Gather all the fitnesses in one list and print the stats
            fits = [ind['fitness'] for ind in self.pop]

            best_ind = self.selectBest(self.pop)

            if best_ind['fitness'] > self.bestindividual['fitness']:
                self.bestindividual = best_ind

            print("Best individual found is {}, {}".format(self.bestindividual['Gene'].data,
                                                           self.bestindividual['fitness']))

            self.result.append(max(fits))
            self.best_indi.append(self.bestindividual['Gene'].data)
            print("  Max fitness of current pop: {}".format(max(fits)))
        np.save('best_indi.npy', np.array(self.best_indi))
        np.save('result.npy',np.array(self.result))
        print("------ End of (successful) evolution ------")


if __name__ == "__main__":
    CXPB, MUTPB, NGEN, popsize = 0.8, 0.3, 1000, 10000  # popsize must be even number; 交叉概率，变异概率，传承代数，种群规模

    up = [1.2, 3, 4, 1.2,
          0, 0, 40, 6,
          1, 20, 60, 60,
          3, 20, 50, 20,
          3, 10, 30, 6,
          3, 20, 15, 10,#xiaoche
          3, 20, 15, 10,
          3, 20, 15, 10,
          3, 20, 30, 30,
          3, 20, 30, 30,
          3, 20, 17, 10]  # upper range for variables
    low = [0, 2, 2, 0,
           0, 0, 25, 0,
           1, 0, 25, 30,
           3, 0, 8, 0,
           0, 0, 8, 0,
           0, 0, 0, 0,
           0, 0, 0, 0,
           0, 0, 0, 0,
           0, 0, 10, 10,
           0, 0, 10, 10,
           0, 0, 0, 0]  # lower range for variables
    '''
    [0, 1, 0, 0,
           0, 0, 0, 0,#前两位直接定义0 0
           1, 100, 0, 0,#第一位1
```

```
            3,  0, 0, 0,第一位为3
            3, 0, 0, 0,
            0, 0,  0, 0,
            0, 4, 0, 0,
            2, 13, 0, 0,
            2, 18, 0, 0,
            1, 24, 0, 0,
            3, 26, 0, 0]

    '''
    '''
        CXPB, MUTPB, NGEN, popsize = 0.8, 0.3, 1000, 10000  # popsize must be even number;交叉概率，变异概率，传承代数，种群规模

    up = [1.2, 3, 3, 1.2,
          0, 0, 40, 6,
          1, 20, 60, 60,
          3, 20, 30, 30,
          3, 10, 30, 6,
          3, 20, 30, 6,#xiaoche
          3, 20, 30, 6,
          3, 20, 30, 6,
          3, 20, 30, 30,
          3, 20, 30, 30,
          3, 20, 17, 6]  # upper range for variables
    low = [0, 2, 2, 0,
           0, 0, 20, 0,
           1, 0, 30, 30,
           3, 0, 8, 10,
           0, 0, 8, 0,
           0, 0, 8, 0,
           0, 0, 8, 0,
           0, 0, 0, 0,
           0, 0, 0, 0,
           0, 0, 0, 0,
           0, 0, 0, 0]

    '''



    parameter = [CXPB, MUTPB, NGEN, popsize, low, up]
    run = GA(parameter)
    run.GA_main()
```