赛区评阅编号（由赛区组委会填写）：

**2023年高教社杯全国大学生数学建模竞赛**

**承诺书**

我们仔细阅读了《全国大学生数学建模竞赛章程》和《全国大学生数学建模竞赛参赛规则》（以下简称“竞赛章程和参赛规则”，可从http://www.mcm.edu.cn下载）。

我们完全清楚，在竞赛开始后参赛队员不能以任何方式，包括电话、电子邮件、“贴吧”、QQ群、微信群等，与队外的任何人（包括指导教师）交流、讨论与赛题有关的问题；无论主动参与讨论还是被动接收讨论信息都是严重违反竞赛纪律的行为。

**我们以中国大学生名誉和诚信郑重承诺，严格遵守竞赛章程和参赛规则，以保证竞赛的公正、公平性。如有违反竞赛章程和参赛规则的行为，我们将受到严肃处理。**

我们授权全国大学生数学建模竞赛组委会，可将我们的论文以任何形式进行公开展示（包括进行网上公示，在书籍、期刊和其他媒体进行正式或非正式发表等）。

我们参赛选择的题号（从A/B/C/D/E中选择一项填写）：

我们的报名参赛队号（12位数字全国统一编号）：

参赛学校（完整的学校全称，不含院系名）：

参赛队员(打印并签名) ：1.

2.

3.

指导教师或指导教师组负责人 (打印并签名)：

（指导教师签名意味着对参赛队的行为和论文的真实性负责）

日期：2024年6月12日

**（请勿改动此页内容和格式。此承诺书打印签名后作为纸质论文的封面，注意电子版论文中不得出现此页。以上内容请仔细核对，如填写错误，论文可能被取消评奖资格。）**

赛区评阅编号： 全国评阅编号：

(由赛区填写） （全国组委会填写）

**2023年高教社杯全国大学生数学建模竞赛**

**编号专用页**

赛区评阅记录（可供赛区评阅时使用）：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 评  阅  人 |  |  |  |  |  |  |
| 备  注 |  |  |  |  |  |  |

送全国评阅统一编号:

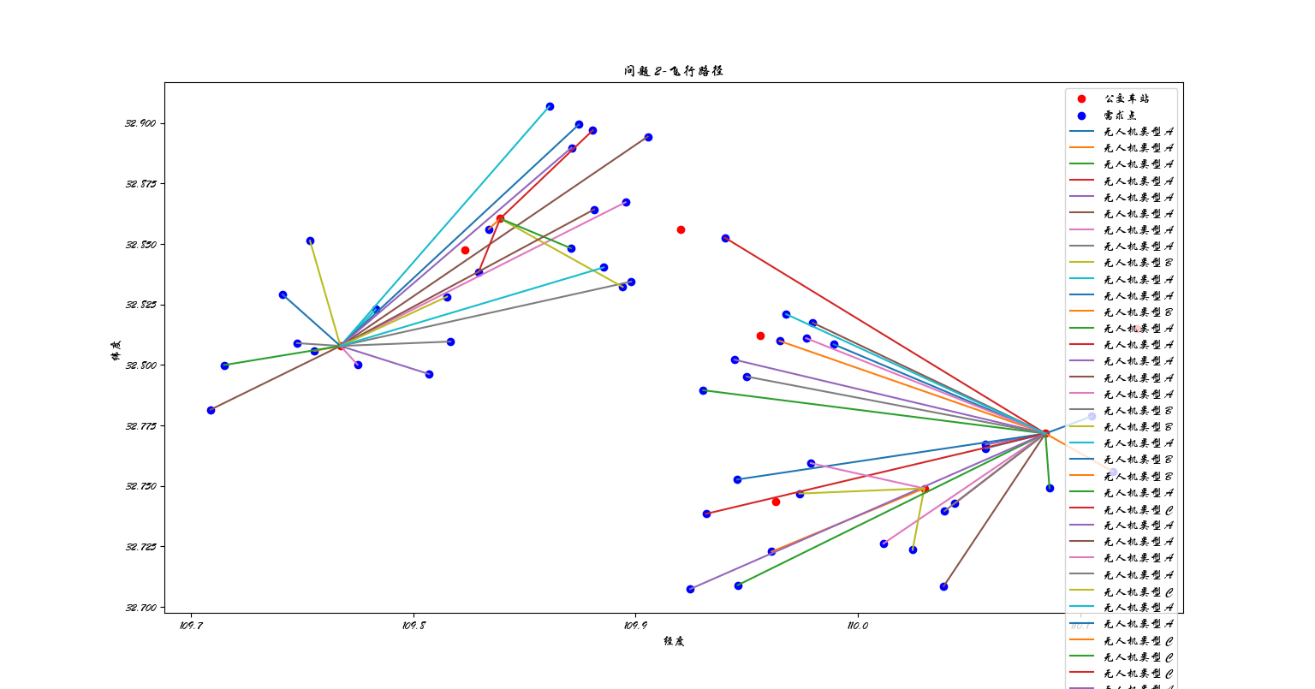
（赛区组委会填写）

**（请勿改动此页内容和格式。此编号专用页仅供赛区和全国评阅使用，参赛队打印后装订到纸质论文的第二页上。注意电子版论文中不得出现此页。）**



**参赛队编号： 257**

**赛题类型代码： A**



**基于线性规划的无人机配送模型**

# 摘 要

农村地区由于地形复杂、交通不便，传统配送方式面临效率低下、成本高昂的挑战。随着无人机技术的快速发展，农村公交与异构无人机协同配送模式应运而生，旨在结合地面公交的广泛覆盖和无人机的空中灵活性，实现高效、经济的配送服务。该模式通过公交将无人机和货物送至各站点，再由无人机完成最后一公里配送，有效提升了配送效率，同时降低了运营成本。

针对问题一，我们采用线性规划方法建立模型，以最小化配送成本。线性规划是一种数学方法，用于在一组线性不等式或等式约束条件下，找到线性目标函数的最优值。首先确保所有配送点的需求都能被满足，无人机的载荷和飞行距离不超过其最大限制。在满足上述约束的条件下，通过贪心算法算出无人机数量，调整决策变量的值，通过python求解得到最小费用为3475元。

针对问题二，我们采用混合整数线性规划方法建立模型，以实现在三种类型无人机均可使用的情况下最小化总配送成本。首先进行多目标优化，在满足所有配送点需求的前提下，考虑不同类型的无人机，再次优化它们的使用比例和任务分配。载荷与飞行距离：确保无人机的载荷和飞行距离不超过其最大限制，同时考虑公交车的携带能力。最后进行成本效益分析，在满足约束条件的同时，通过调整决策变量值，通过python求解得到最小成本约为1403.90元。

针对问题三，与问题二相似，我们采用混合整数线性规划方法建立模型，以在考虑取货收入的情况下，实现最小化总配送成本。首先优化综合配送与取货，在配送方案中同时考虑配送和取货任务，优化无人机的任务分配，考虑取货带来的收入，实现经济收益最大化。确保无人机的载荷和飞行距离不超过其最大限制，同时考虑公交车的携带能力。其次优化成本与收入权衡在配送成本最小化，通过python求解得到最小成本为1389元。

**关键词：** 线性规划、混合整数线性规划、无人机配送系统、贪心算法、python

# 一、问题重述

问题一要求我们仅使用A类无人机，设计一套公交与无人机协同配送方案，目标是最小化总配送成本。这包括考虑无人机的飞行特性、载荷能力、配送点的需求以及公交系统的运行时序，以制定出最经济高效的配送策略。

问题二要求在三种类型的无人机（A、B、C类）均可使用的情况下，需要综合考虑无人机性能、配送点需求、公交系统时序，以及成本效益，实现最优的任务分配和资源调度，制定一个最小化总配送成本的协同配送方案。

问题三在问题二的基础上增加取货量，要求制定配送方案以最小化总成本，同时考虑取货带来的利润。需优化无人机类型选择、任务分配，以及配送和取货的协调，实现经济效益最大化。

# 二、模型假设与符号说明

1. 假设所有同一类型的无人机性能相同，没有个体差异。
2. 假设无人机和公交车的载重能力是固定的，并且每次任务只能携带最大载荷。
3. 假设无人机的飞行距离和时间仅受其最大飞行距离和速度的限制。
4. 假设配送点的需求量是已知的，并且在配送期间不会发生变化。
5. 假设天气和环境条件对无人机飞行没有影响，所有飞行均在理想条件下进行。
6. 假设无人机的使用成本、固定费用和变动费用在模型求解期间保持不变。
7. 假设公交车按照预定的发车时间和路线运行，没有延误。
8. 假设各个配送任务之间没有相互依赖关系，可以独立进行优化。
9. 假设无人机在执行任务期间的安全性得到保障，没有飞行安全风险。
10. 假设客户的需求在配送期间是固定的，不会新增或取消。
11. 假设配送点和公交站点的地理坐标信息准确无误。

# 三、符号说明

|  |  |  |  |
| --- | --- | --- | --- |
| 符 号 |  | 意义 |  |
| Fa |  | A类无人机固定费用 |  |
| Xij |  | 表示从公交站点i到需求点j是否有无人机飞行任务 |  |
| dij |  | 表示公交站点i到需求点j的距离 |  |
| wi |  | 表示公交站点i是否有无人机飞行任务 |  |
| Qa\_km |  | A类无人机每公里费用 |  |
| La |  | A类无人机最大载重 |  |
|  |  | 从公交站点i到需求点j使用第k类无人机的运输费用 |  |
|  |  | K类无人机固定费用  K类无人机最大飞行距离 |  |

# 四、问题分析

4.1问题一的分析

问题一要求我们仅使用A类无人机，设计一套公交与无人机协同配送方案，以实现最小化总配送成本。这个问题涉及到多个关键因素，包括无人机的飞行特性、载荷能力、配送点的需求、以及公交系统的运行时序。

首先，A类无人机具有特定的最大载货量、飞行距离和速度，这些特性限制了无人机的配送范围和效率。我们需要根据这些参数，确定无人机从每个公交站点出发能够覆盖的配送区域。其次，配送点的需求包括配送需求和取货需求，这要求我们在配送方案中同时考虑送货和取货的效率。配送点的地理分布也会影响无人机的飞行路径和时间安排。

再者，公交系统的运行时序决定了无人机的起降时间和可能的等待时间。我们需要根据公交的发车时间和站点，合理安排无人机的配送任务，以减少等待时间和提高配送效率。

最后，总配送成本包括无人机的固定使用费用和根据飞行里程计算的变动成本。我们需要在满足所有配送需求的前提下，优化无人机的使用次数和飞行路径，以实现成本最小化。综上所述，问题一的解决方案需要综合考虑无人机的性能参数、配送点的需求、公交系统的运行时序，以及成本控制因素，通过数学建模和优化算法，制定出一套经济高效的配送方案。

4.2问题二的分析

问题二扩展了问题一的范围，允许使用三种类型的无人机（A、B、C类），以制定最小费用的协同配送方案。这一问题的复杂性在于无人机类型的多样性，每种类型的无人机具有不同的飞行特性、载荷能力和成本结构。

首先，我们需要考虑每种无人机的最大载货量和飞行距离，这些参数决定了无人机的配送能力和服务范围。例如，C类无人机可能具有最大的载荷能力，但可能伴随着更高的单位配送成本和固定费用。

其次，配送点的需求多样性要求我们对不同类型的无人机进行合理分配。某些配送点可能因为距离较远或需求量大，更适合使用载荷量大、飞行距离远的无人机，而其他近距离或需求量小的点则可能由A类或B类无人机服务。

再者，公交系统的运行时序和路线对无人机的起降点和配送时间窗口有直接影响。我们需要根据公交的发车时间和路线，以及无人机的飞行速度和续航能力，合理安排无人机的起降站点和配送顺序。

最后，总配送成本的计算需要综合考虑无人机的固定费用和变动费用。在满足所有配送点需求的同时，我们需要优化无人机的类型选择、使用次数和飞行路径，以达到成本最小化的目标。

问题二的解决方案要求我们进行多目标优化，即在满足配送需求和提高配送效率的同时，实现成本的最小化。这可能涉及到复杂的决策过程，包括但不限于线性规划、整数规划或启发式算法等数学优化方法。通过这些方法，我们可以为每种类型的无人机分配合适的任务，制定出一套综合考虑成本和效率的最优配送方案。

4.3问题三的分析

问题三在问题二的基础上增加了取货需求，并引入了取货收入的概念，这为配送方案的优化带来了新的经济考量。在这一问题中，无人机不仅要完成配送任务，还要执行取货任务，且每公斤取货能带来0.5元的收入。

首先，取货需求的加入意味着无人机的载荷能力需要同时考虑配送和取货的总和，这可能会影响到无人机的飞行路径和任务分配。一个配送点同时有配送和取货需求，无人机的载荷能力必须能够满足这一综合需求。

其次，取货收入为配送方案带来了额外的经济激励。在计算总成本时，我们需要将取货收入纳入考虑，从而可能改变某些配送任务的经济可行性。这要求我们在优化配送方案时，不仅要考虑成本最小化，还要考虑收益最大化。

再者，无人机的飞行特性和成本结构在考虑取货收入后可能会发生变化。例如，虽然C类无人机的单位配送成本较高，但如果其载荷能力大，能够在一次飞行中完成大量的取货任务，那么其总体经济效益可能会优于其他类型的无人机。

最后，问题三的解决方案需要综合考虑配送和取货任务的协调，以及取货收入对总成本的影响。我们需要通过数学建模和优化算法，制定出一套既能满足配送和取货需求，又能最大化取货收入，从而实现总体成本最小化的配送方案。

综上所述，问题三的分析和解决方案需要在问题二的基础上，进一步考虑取货任务的经济价值和无人机的载荷能力，通过精细化的任务分配和成本收益分析，实现配送方案的最优设计。

# 五、模型建立与求解

5.1问题一的模型建立与求解

5.1.1.目标函数



Xij表示从公交站点i到需求点j是否有无人机飞行任务(1表示有，0表示无)

dij表示公交站点i到需求点j的距离。

wi表示公交站点i是否有无人机飞行任务(1表示有，0表示无)





5.1.2.约束条件

由附录可知无人机需满足

1.容量限制





2.飞行距离限制





3.站点是否有无人机任务



4.需求点覆盖



5.二元配置

，

通过python求解画图得

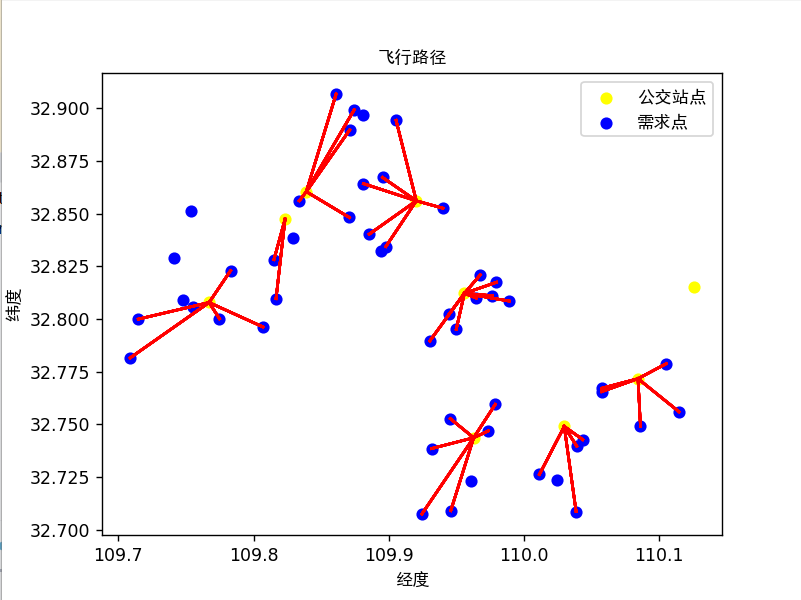
****

图1 A无人机路径图

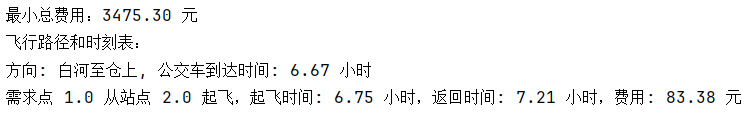
****

图2 公交与无人机协同配送方案

通过贪心算法，详细求出只考虑使用A类无人机时，飞行路径和时刻表:

从公交站点(110.08442, 32.771676)到需求点(110.1053385, 32.77881526)的飞行距离为2.45km，费用为1.96元

从公交站点(110.08442, 32.771676)到需求点(110.1147032, 32.75599834)的飞行距离为3.79km，费用为3.03元

从公交站点(110.08442, 32.771676)到需求点(110.0862574, 32.74905239)的飞行距离为2.52km，费用为2.02元

从公交站点(110.08442, 32.771676)到需求点(110.0575508, 32.76712584)的飞行距离为3.02km，费用为2.42元

从公交站点(110.029866, 32.748994)到需求点(110.0386243, 32.70855831)的飞行距离为4.59km，费用为3.67元

从公交站点(110.029866, 32.748994)到需求点(110.0115086, 32.72619993)的飞行距离为3.25km，费用为2.60元

从公交站点(110.029866, 32.748994)到需求点(110.0390602, 32.73965997)的飞行距离为1.45km，费用为1.16元

从公交站点(110.029866, 32.748994)到需求点(110.0246454, 32.72360718)的飞行距离为2.88km，费用为2.30元

从公交站点(110.08442, 32.771676)到需求点(110.0575847, 32.76553658)的飞行距离为3.06km，费用为2.44元

从公交站点(109.962839, 32.743622)到需求点(109.9456331, 32.7526657)的飞行距离为2.16km，费用为1.73元

从公交站点(109.962839, 32.743622)到需求点(109.9612274, 32.72286471)的飞行距离为2.31km，费用为1.85元

从公交站点(109.962839, 32.743622)到需求点(109.94592, 32.70899877)的飞行距离为4.28km，费用为3.42元

从公交站点(109.962839, 32.743622)到需求点(109.9316682, 32.73848444)的飞行距离为3.51km，费用为2.81元

从公交站点(109.962839, 32.743622)到需求点(109.9245376, 32.70740885)的飞行距离为5.85km，费用为4.68元

从公交站点(109.767127, 32.807855)到需求点(109.7087533, 32.7815564)的飞行距离为7.11km，费用为5.69元

从公交站点(109.767127, 32.807855)到需求点(109.7748005, 32.80016336)的飞行距离为1.21km，费用为0.96元

从公交站点(109.767127, 32.807855)到需求点(109.7475891, 32.80903496)的飞行距离为2.17km，费用为1.74元

从公交站点(109.767127, 32.807855)到需求点(109.7534532, 32.85129032)的飞行距离为5.05km，费用为4.04元

从公交站点(109.767127, 32.807855)到需求点(109.783015, 32.82296929)的飞行距离为2.43km，费用为1.95元

从公交站点(109.767127, 32.807855)到需求点(109.7410728, 32.82914197)的飞行距离为3.73km，费用为2.99元

从公交站点(109.767127, 32.807855)到需求点(109.7554844, 32.80581363)的飞行距离为1.31km，费用为1.05元

从公交站点(109.767127, 32.807855)到需求点(109.7147417, 32.79995734)的飞行距离为5.88km，费用为4.70元

从公交站点(109.839046, 32.860495)到需求点(109.8807093, 32.89696579)的飞行距离为6.15km，费用为4.92元

从公交站点(109.767127, 32.807855)到需求点(109.8070677, 32.79622985)的飞行距离为4.62km，费用为3.69元

从公交站点(109.920425, 32.856136)到需求点(109.9054481, 32.89437141)的飞行距离为4.56km，费用为3.65元

从公交站点(109.920425, 32.856136)到需求点(109.8954509, 32.86724756)的飞行距离为3.03km，费用为2.43元

从公交站点(109.920425, 32.856136)到需求点(109.8979229, 32.83444574)的飞行距离为3.47km，费用为2.78元

从公交站点(109.920425, 32.856136)到需求点(109.8942179, 32.83224374)的飞行距离为3.94km，费用为3.15元

从公交站点(109.839046, 32.860495)到需求点(109.8610985, 32.90687042)的飞行距离为5.70km，费用为4.56元

从公交站点(109.839046, 32.860495)到需求点(109.8744682, 32.89939698)的飞行距离为5.84km，费用为4.67元

从公交站点(109.839046, 32.860495)到需求点(109.8338804, 32.85616627)的飞行距离为0.75km，费用为0.60元

从公交站点(109.839046, 32.860495)到需求点(109.870924, 32.848223)的飞行距离为3.79km，费用为3.03元

从公交站点(109.823329, 32.847468)到需求点(109.8292467, 32.83825122)的飞行距离为1.22km，费用为0.97元

从公交站点(109.839046, 32.860495)到需求点(109.8711312, 32.88979101)的飞行距离为4.82km，费用为3.86元

从公交站点(109.920425, 32.856136)到需求点(109.8813363, 32.8642824)的飞行距离为4.43km，费用为3.55元

从公交站点(109.962839, 32.743622)到需求点(109.978788, 32.75943454)的飞行距离为2.49km，费用为1.99元

从公交站点(109.823329, 32.847468)到需求点(109.8166563, 32.8096699)的飞行距离为4.26km，费用为3.41元

从公交站点(109.823329, 32.847468)到需求点(109.8151216, 32.82822489)的飞行距离为2.32km，费用为1.86元

从公交站点(109.920425, 32.856136)到需求点(109.885638, 32.84032485)的飞行距离为4.24km，费用为3.39元

从公交站点(109.956003, 32.812194)到需求点(109.9890984, 32.80854774)的飞行距离为3.70km，费用为2.96元

从公交站点(109.956003, 32.812194)到需求点(109.9647812, 32.80993619)的飞行距离为1.01km，费用为0.80元

从公交站点(109.956003, 32.812194)到需求点(109.9303732, 32.78956582)的飞行距离为3.80km，费用为3.04元

从公交站点(109.920425, 32.856136)到需求点(109.9401099, 32.85264625)的飞行距离为2.22km，费用为1.78元

从公交站点(109.956003, 32.812194)到需求点(109.944496, 32.802178)的飞行距离为1.69km，费用为1.35元

从公交站点(109.956003, 32.812194)到需求点(109.979708, 32.817449)的飞行距离为2.70km，费用为2.16元

从公交站点(109.956003, 32.812194)到需求点(109.976757, 32.811064)的飞行距离为2.31km，费用为1.85元

从公交站点(109.956003, 32.812194)到需求点(109.94999, 32.795207)的飞行距离为2.00km，费用为1.60元

从公交站点(109.962839, 32.743622)到需求点(109.973673, 32.746858)的飞行距离为1.26km，费用为1.00元

5.2问题二的模型建立与求解

5.2.1目标函数

最小化总费用:包括各类无人机的固定费用和运输费用。



**是从公交站点i到需求点j使用第k类无人机的运输费用，等于**

**:是第K类无人机的固定费用**

**使用第k类无人机是从站点i到需求点j是否有配送(0或1)，使用第k类无人机。**

5.2.2约束条件

1.需求点满足

2.最大飞行距离

3.最大载重

4.回到起始站

**通过python求解画图得**

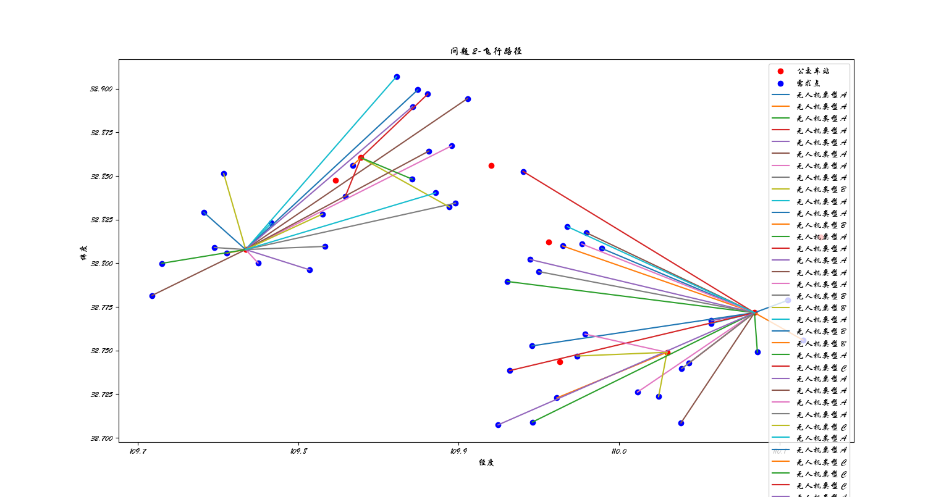
****

图2 ABC无人机路径图



求解得到最小成本约为1403.70元

5.3问题三的模型建立与求解

使用Haversine公式计算每个公交站点和每个需求点之间的距离，并存储在距离矩阵中。



是地球半径，约为6371公里。

分别是两个点的纬度。

分别是两个点的经度。

。

。

通过python求解画图得

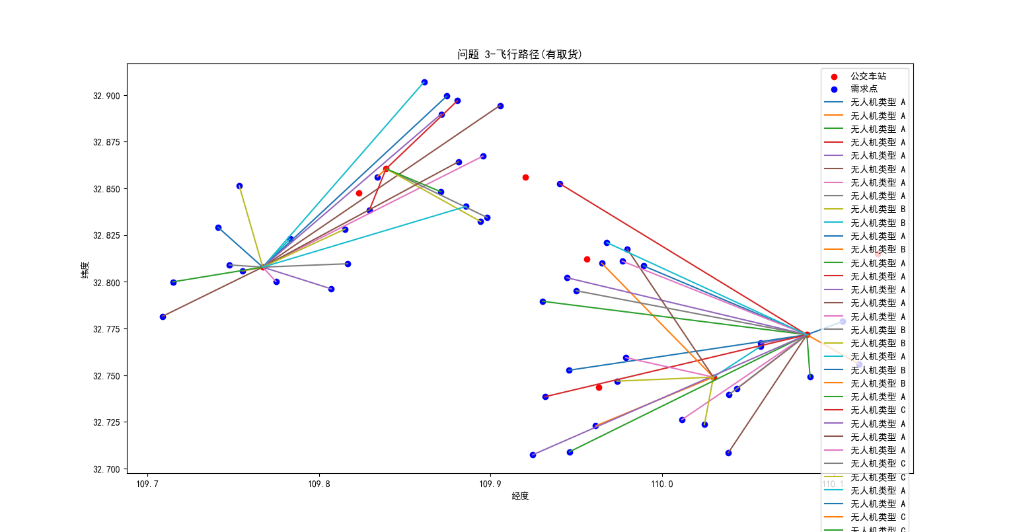
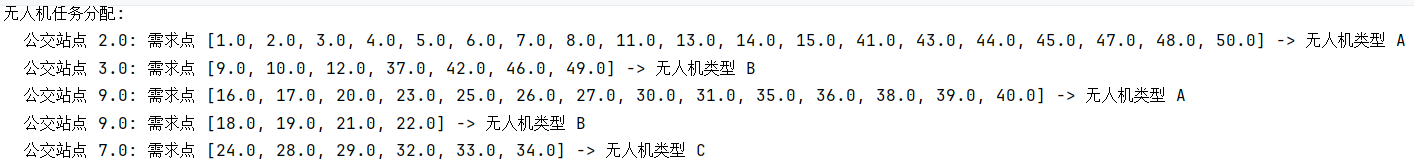
****

图3 ABC无人机路径图（有取货）



**图4 无人机任务分配**

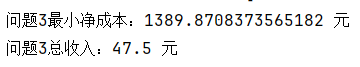
****

图5 问题三最小净成本和总收入

# 六、模型评价

6.1模型优缺点

6.1.1优点：

由于问题一限制使用单一类型的无人机，模型中没有整数变量，使得求解过程快速且高效。模型中只涉及一种类型的无人机，减少了模型的复杂性，简化了决策变量和约束条件。而且线性模型易于理解和实现，模型结构清晰，便于理解和实现，适合作为更复杂模型的基础。它还可以处理多种类型无人机的协同配送问题，增加了模型的适用性和灵活性，考虑了不同类型的无人机，可以更全面地优化配送方案，提高资源利用效率。并且能够处理实际运营中的整数约束，如无人机数量限制，更贴近实际应用。在配送成本最小化的基础上，考虑了取货带来的收入，使模型更加符合实际经济活动，通过同时考虑配送和取货任务，可以更有效地利用无人机的运输能力。允许制定更多样化的配送策略，以适应不同的配送和取货需求。该模型使用了贪心算法来进行启发式方法，还具有高度可移植性，例如在无人机配送模型中，如果配送需求或飞行条件发生变化，贪心算法可以快速重新计算并适应新的情况。还可以根据不同的策略进行调整，以适应不同的优化目标或约束条件。在无人机配送模型中，可以根据实际需求调整贪心策略，以优化配送成本、时间或服务质量。

6.1.2缺点：

灵活性不足，无法考虑无人机的取货任务，无法充分利用无人机的双向运输能力。由于引入了整数变量，求解时间相比LP模型会有显著增加。：大规模问题可能需要更多的计算资源和时间。同时考虑配送和取货任务，增加了模型的复杂性，求解难度更高。需要处理更多的变量和约束，对求解器的性能提出了更高的要求。模型的实现和调试可能更加困难，需要更精细的算法调整。在实际应用中，无人机会受到天气、季节等因素的影响，具有较大的不确定性和波动性。为了提高模型的准确性和实用性，可以通过引入不确定性和波动性来改进模型。例如，可以使用模糊数学或随机规划方法来处理不确定性和波动性，或者通过建立多场景模型来考虑不同情况

**参考文献**

[1] 姜肩源等数学模型(第三版) [M]. 高等教育出版社.2003.8.

[2] 注晓银，邹庭荣，周保平.数学软件与数学实验(第二版) .北京:科学出版社，2012. 8.

[3] 姜启源，叶其孝:数学建模.北京:机械工业出版社.2009, 8.

[4]刘浩,韩晶.MATLABR2020a完全自学一本通[M].电子工业出版社:202009.596.

[5]包子阳,余继周,杨杉.智能优化算法及其MATLAB实例[M]电子工业出版社:202101.229.

[6]李艳波,李若尘,史博,等.基于改进模拟退火遗传算法的高速公路服务区自洽能源系统高能效优化[J.西安交通大学学报,2024,58(01):197-207+216.

[7]鞠立伟,吕硕硕,李鹏.新型电力系统需求侧灵活性资源时空协同优化与动态均衡机制研究综述[J/OL].电力建设,1-27[2024-05-25].

**附录：**

1.第一问

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.font\_manager import FontProperties

from itertools import combinations

font\_path = 'C:/Windows/Fonts/simhei.ttf'

font = FontProperties(fname=font\_path)

# A无人机信息

D\_max\_A = 27 # 最大飞行距离 (km)

Q\_max\_A = 9 # 最大载重 (kg)

C\_fixed\_A = 80 # 固定费用 (元/天)

C\_per\_km\_A = 0.8 # 每公里费用 (元/km)

drone\_speed = 16.7 / 3.6 # 无人机速度 (km/h), 由 m/s 转为 km/h

# 公交站点数据

stations\_data = pd.DataFrame({

'Station\_ID': [1, 2, 3, 4, 5, 6, 7, 8, 9],

'Longitude': [110.125713, 110.08442, 110.029866, 109.962839, 109.956003, 109.920425, 109.839046, 109.823329,

109.767127],

'Latitude': [32.815024, 32.771676, 32.748994, 32.743622, 32.812194, 32.856136, 32.860495, 32.847468, 32.807855]

})

# 需求点数据

demands\_data = pd.DataFrame({

'Demand\_ID': range(1, 51),

'Longitude': [

110.1053385, 110.1147032, 110.0862574, 110.0435344, 110.0575508,

110.0386243, 110.0115086, 110.0390602, 110.0246454, 110.0575847,

109.9456331, 109.9612274, 109.94592, 109.9316682, 109.9245376,

109.7087533, 109.7748005, 109.7475891, 109.7534532, 109.783015,

109.7410728, 109.7554844, 109.7147417, 109.8807093, 109.8070677,

109.9054481, 109.8954509, 109.8979229, 109.8942179, 109.8610985,

109.8744682, 109.8338804, 109.870924, 109.8292467, 109.8711312,

109.8813363, 109.978788, 109.8166563, 109.8151216, 109.885638,

109.9890984, 109.9647812, 109.9303732, 109.9401099, 109.944496,

109.979708, 109.976757, 109.94999, 109.973673, 109.967765

],

'Latitude': [

32.77881526, 32.75599834, 32.74905239, 32.74275416, 32.76712584,

32.70855831, 32.72619993, 32.73965997, 32.72360718, 32.76553658,

32.7526657, 32.72286471, 32.70899877, 32.73848444, 32.70740885,

32.7815564, 32.80016336, 32.80903496, 32.85129032, 32.82296929,

32.82914197, 32.80581363, 32.79995734, 32.89696579, 32.79622985,

32.89437141, 32.86724756, 32.83444574, 32.83224374, 32.90687042,

32.89939698, 32.85616627, 32.848223, 32.83825122, 32.88979101,

32.8642824, 32.75943454, 32.8096699, 32.82822489, 32.84032485,

32.80854774, 32.80993619, 32.78956582, 32.85264625, 32.802178,

32.817449, 32.811064, 32.795207, 32.746858, 32.820998

],

'Demand\_kg': [3, 4, 2, 0, 8, 7, 4, 9, 10, 6, 7, 12, 3, 5, 6, 5, 3, 13, 12, 3,

14, 10, 4, 34, 6, 6, 3, 4, 20, 5, 6, 5, 3, 15, 2, 6, 3, 4, 3, 2,

6, 5, 9, 3, 3, 4, 6, 4, 4, 0]

})

# 计算两个点之间的距离

def haversine(lon1, lat1, lon2, lat2):

lon1, lat1, lon2, lat2 = map(np.radians, [lon1, lat1, lon2, lat2])

dlon = lon2 - lon1

dlat = lat2 - lat1

a = np.sin(dlat / 2) \*\* 2 + np.cos(lat1) \* np.cos(lat2) \* np.sin(dlon / 2) \*\* 2

c = 2 \* np.arcsin(np.sqrt(a))

r = 6371 # 地球平均半径（公里）

return c \* r

# 公交站点和需求点的经纬度

stations\_lons = stations\_data['Longitude'].values

stations\_lats = stations\_data['Latitude'].values

demands\_lons = demands\_data['Longitude'].values

demands\_lats = demands\_data['Latitude'].values

# 计算所有公交站点到所有需求点的距离

stations\_lons\_matrix, demands\_lons\_matrix = np.meshgrid(stations\_lons, demands\_lons)

stations\_lats\_matrix, demands\_lats\_matrix = np.meshgrid(stations\_lats, demands\_lats)

distances\_matrix = haversine(stations\_lons\_matrix, stations\_lats\_matrix, demands\_lons\_matrix, demands\_lats\_matrix)

distances = {(stations\_data['Station\_ID'][i], demands\_data['Demand\_ID'][j]): distances\_matrix[j, i]

for i in range(len(stations\_data)) for j in range(len(demands\_data))}

# 发车时间表

bus\_schedule = {

'白河至仓上': [6.67, 8.5, 9, 11, 14, 16.5],

'仓上至白河': [6, 7.33, 8.83, 11, 14, 15.83]

}

# 贪心算法优化无人机使用量

def optimize\_drone\_usage():

total\_cost = 0

drone\_tasks = []

remaining\_demands = set(demands\_data['Demand\_ID'])

while remaining\_demands:

best\_station = None

best\_combination = None

best\_combination\_cost = float('inf')

for station\_id in stations\_data['Station\_ID']:

valid\_demands = [d for d in remaining\_demands if distances[(station\_id, d)] <= D\_max\_A and

demands\_data.loc[demands\_data['Demand\_ID'] == d, 'Demand\_kg'].values[0] <= Q\_max\_A]

for i in range(1, len(valid\_demands) + 1):

for combination in combinations(valid\_demands, i):

total\_weight = sum(demands\_data.loc[demands\_data['Demand\_ID'].isin(combination), 'Demand\_kg'])

if total\_weight > Q\_max\_A:

continue

combination\_cost = C\_fixed\_A + 2 \* C\_per\_km\_A \* sum(distances[(station\_id, d)] for d in combination)

if combination\_cost < best\_combination\_cost:

best\_combination = combination

best\_station = station\_id

best\_combination\_cost = combination\_cost

if best\_combination is None:

break

drone\_tasks.append((best\_station, best\_combination, best\_combination\_cost))

total\_cost += best\_combination\_cost

remaining\_demands -= set(best\_combination)

return total\_cost, drone\_tasks

total\_cost, drone\_tasks = optimize\_drone\_usage()

# 可视化飞行路径

def visualize\_flight\_paths():

fig, ax = plt.subplots()

# 绘制公交站点

ax.scatter(stations\_data['Longitude'], stations\_data['Latitude'], c='blue', label='公交站点')

# 绘制需求点

ax.scatter(demands\_data['Longitude'], demands\_data['Latitude'], c='red', label='需求点')

# 绘制飞行路径

for station, tasks, cost in drone\_tasks:

station\_coords = stations\_data[stations\_data['Station\_ID'] == station][['Longitude', 'Latitude']].values[0]

for demand\_id in tasks:

demand\_coords = demands\_data[demands\_data['Demand\_ID'] == demand\_id][['Longitude', 'Latitude']].values[0]

ax.plot([station\_coords[0], demand\_coords[0]], [station\_coords[1], demand\_coords[1]], c='green')

ax.legend(prop=font)

ax.set\_xlabel('经度', fontproperties=font)

ax.set\_ylabel('纬度', fontproperties=font)

ax.set\_title('飞行路径', fontproperties=font)

plt.show()

visualize\_flight\_paths()

# 打印最优结果

print(f"最小总费用：{total\_cost:.2f} 元")

print("无人机任务分配：")

for i, (station, tasks, cost) in enumerate(drone\_tasks):

print(f"无人机 {i + 1}:")

print(f" 从站点 {station} 起飞")

for demand\_id in tasks:

demand\_weight = demands\_data.loc[demands\_data['Demand\_ID'] == demand\_id, 'Demand\_kg'].values[0]

print(f" 需求点 {demand\_id}，需求重量: {demand\_weight} kg")

print(f" 总费用: {cost:.2f} 元")

print("-----")

**2.第二问**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import matplotlib#需要导入的库

matplotlib.rc("font", family='STXingkai')#family是设置的字体

# 公交站点数据

stations\_data = pd.DataFrame({

'Station\_ID': [1, 2, 3, 4, 5, 6, 7, 8, 9],

'Longitude': [110.125713, 110.08442, 110.029866, 109.962839, 109.956003, 109.920425, 109.839046, 109.823329,

109.767127],

'Latitude': [32.815024, 32.771676, 32.748994, 32.743622, 32.812194, 32.856136, 32.860495, 32.847468, 32.807855]

})

# 需求点数据

demands\_data = pd.DataFrame({

'Demand\_ID': range(1, 51),

'Longitude': [

110.1053385, 110.1147032, 110.0862574, 110.0435344, 110.0575508,

110.0386243, 110.0115086, 110.0390602, 110.0246454, 110.0575847,

109.9456331, 109.9612274, 109.94592, 109.9316682, 109.9245376,

109.7087533, 109.7748005, 109.7475891, 109.7534532, 109.783015,

109.7410728, 109.7554844, 109.7147417, 109.8807093, 109.8070677,

109.9054481, 109.8954509, 109.8979229, 109.8942179, 109.8610985,

109.8744682, 109.8338804, 109.870924, 109.8292467, 109.8711312,

109.8813363, 109.978788, 109.8166563, 109.8151216, 109.885638,

109.9890984, 109.9647812, 109.9303732, 109.9401099, 109.944496,

109.979708, 109.976757, 109.94999, 109.973673, 109.967765

],

'Latitude': [

32.77881526, 32.75599834, 32.74905239, 32.74275416, 32.76712584,

32.70855831, 32.72619993, 32.73965997, 32.72360718, 32.76553658,

32.7526657, 32.72286471, 32.70899877, 32.73848444, 32.70740885,

32.7815564, 32.80016336, 32.80903496, 32.85129032, 32.82296929,

32.82914197, 32.80581363, 32.79995734, 32.89696579, 32.79622985,

32.89437141, 32.86724756, 32.83444574, 32.83224374, 32.90687042,

32.89939698, 32.85616627, 32.848223, 32.83825122, 32.88979101,

32.8642824, 32.75943454, 32.8096699, 32.82822489, 32.84032485,

32.80854774, 32.80993619, 32.78956582, 32.85264625, 32.802178,

32.817449, 32.811064, 32.795207, 32.746858, 32.820998

],

'Delivery\_kg': [3, 4, 2, 0, 8, 7, 4, 9, 10, 6, 7, 12, 3, 5, 6, 5, 3, 13, 12, 3,

14, 10, 4, 34, 6, 6, 3, 4, 20, 5, 6, 5, 3, 15, 2, 6, 3, 4, 3, 2,

6, 5, 9, 3, 3, 4, 6, 4, 4, 0],

'Pickup\_kg': [0, 0, 5, 3, 0, 0, 3, 0, 0, 5, 0, 0, 0, 2, 0, 4, 0, 0, 0, 6, 0, 0, 0, 10, 0, 0, 0, 7, 0, 0, 0, 0, 8, 0,

0, 0, 0, 0, 6, 0, 0, 14, 0, 0, 0, 13, 0, 0, 0, 9]

})

# 无人机参数

drone\_params = {

'A': {'D\_max': 27, 'Q\_max': 9, 'C\_fixed': 80, 'C\_per\_km': 0.8},

'B': {'D\_max': 16, 'Q\_max': 30, 'C\_fixed': 150, 'C\_per\_km': 1.5},

'C': {'D\_max': 18, 'Q\_max': 50, 'C\_fixed': 200, 'C\_per\_km': 2.5}

}

# 公交车参数

bus\_speed = 35 # 公交车速度（km/h）

# 发车时间表

bus\_schedule = {

'白河至仓上': [6.67, 8.5, 9, 11, 14, 16.5],

'仓上至白河': [6, 7.33, 8.83, 11, 14, 15.83]

}

# 计算两个点之间的距离（Haversine公式）

def haversine(lon1, lat1, lon2, lat2):

lon1, lat1, lon2, lat2 = map(np.radians, [lon1, lat1, lon2, lat2])

dlon = lon2 - lon1

dlat = lat2 - lat1

a = np.sin(dlat / 2) \*\* 2 + np.cos(lat1) \* np.cos(lat2) \* np.sin(dlon / 2) \*\* 2

c = 2 \* np.arcsin(np.sqrt(a))

r = 6371 # 地球平均半径（公里）

return c \* r

# 将公交站点和需求点的经纬度转换为NumPy数组

stations\_lons = stations\_data['Longitude'].values

stations\_lats = stations\_data['Latitude'].values

demands\_lons = demands\_data['Longitude'].values

demands\_lats = demands\_data['Latitude'].values

# 利用广播机制计算所有公交站点到所有需求点的距离

stations\_lons\_matrix, demands\_lons\_matrix = np.meshgrid(stations\_lons, demands\_lons)

stations\_lats\_matrix, demands\_lats\_matrix = np.meshgrid(stations\_lats, demands\_lats)

distances\_matrix = haversine(stations\_lons\_matrix, stations\_lats\_matrix, demands\_lons\_matrix, demands\_lats\_matrix)

# 将距离矩阵转换为字典形式

distances = {(stations\_data['Station\_ID'][i], demands\_data['Demand\_ID'][j]): distances\_matrix[j, i]

for i in range(len(stations\_data)) for j in range(len(demands\_data))}

# 问题3：三种类型无人机的最小净成本协同配送方案，考虑取货收入

# 问题3：三种类型无人机的最小成本协同配送方案，不考虑取货收入

def allocate\_tasks\_and\_calculate\_costs(): # 函数名修改以反映逻辑变化

total\_cost = 0

task\_allocation = [] # 不再包含总收入和取货收入

demand\_visited = set()

station\_drone\_usage = {station\_id: {'A': 0, 'B': 0, 'C': 0} for station\_id in stations\_data['Station\_ID']}

for demand\_id, demand in demands\_data.iterrows():

if demand\_id in demand\_visited:

continue

best\_cost = float('inf')

best\_station = None

best\_drone = None

for drone\_type, params in drone\_params.items():

D\_max = params['D\_max']

Q\_max = params['Q\_max']

C\_fixed = params['C\_fixed']

C\_per\_km = params['C\_per\_km']

for station\_id, station in stations\_data.iterrows():

station\_id = station['Station\_ID']

distance = distances[(station\_id, demand['Demand\_ID'])]

if distance <= D\_max and demand['Delivery\_kg'] <= Q\_max:

cost = C\_fixed if station\_drone\_usage[station\_id][drone\_type] == 0 else 0

cost += 2 \* distance \* C\_per\_km # 往返距离

if cost < best\_cost:

best\_cost = cost

best\_station = station\_id

best\_drone = drone\_type

if best\_station is not None and best\_drone is not None:

total\_cost += best\_cost

task\_allocation.append((demand['Demand\_ID'], best\_station, best\_cost, best\_drone))

station\_drone\_usage[best\_station][best\_drone] += 1

demand\_visited.add(demand\_id)

return total\_cost, task\_allocation # 返回值中不再包含总收入

total\_cost, task\_allocation = allocate\_tasks\_and\_calculate\_costs() # 移除了 total\_revenue

def generate\_flight\_schedule\_with\_pickup(task\_allocation):

wait\_time = 5 / 60 # 等待时间（小时）

flight\_schedule = []

for direction, times in bus\_schedule.items():

for time in times:

for task in task\_allocation:

demand\_id, station\_id, net\_cost, drone\_type = task

arrival\_time = time

distance = distances[(station\_id, demand\_id)]

flight\_time = distance / (bus\_speed / 2)

start\_time = arrival\_time + wait\_time

end\_time = start\_time + flight\_time

flight\_schedule.append({

'Demand\_ID': demand\_id,

'Drone\_Type': drone\_type,

'Station\_ID': station\_id,

'Flight\_Start\_Time': start\_time,

'Flight\_End\_Time': end\_time

})

return flight\_schedule

flight\_schedule\_with\_pickup = generate\_flight\_schedule\_with\_pickup(task\_allocation)

# 任务分配信息

def print\_task\_allocation(task\_allocation):

allocation\_by\_station = {}

for task in task\_allocation:

demand\_id, station\_id, net\_cost, drone\_type = task

if station\_id not in allocation\_by\_station:

allocation\_by\_station[station\_id] = {}

if drone\_type not in allocation\_by\_station[station\_id]:

allocation\_by\_station[station\_id][drone\_type] = []

allocation\_by\_station[station\_id][drone\_type].append(demand\_id)

print("无人机任务分配:")

for station\_id, drones in allocation\_by\_station.items():

for drone\_type, demands in drones.items():

print(f" 公交站点 {station\_id}: 需求点 {demands} -> 无人机类型 {drone\_type}")

# 路线和时刻表信息

def print\_route\_schedule():

print("路线: 白河至仓上")

for time in bus\_schedule['白河至仓上']:

arrival\_times = time + (distances\_matrix[:, 0] / bus\_speed)

print(f" 出发: {time}h")

print(f" 到达时间: {arrival\_times}")

print("路线: 仓上至白河")

for time in bus\_schedule['仓上至白河']:

arrival\_times = time + (distances\_matrix[:, 0] / bus\_speed)

print(f" 出发: {time}h")

print(f" 到达时间: {arrival\_times}")

# 可视化飞行路径

def visualize\_flight\_paths(task\_allocation, title):

fig, ax = plt.subplots()

# 绘制公交站点

ax.scatter(stations\_data['Longitude'], stations\_data['Latitude'], c='red', label='公交车站')

# 绘制需求点

ax.scatter(demands\_data['Longitude'], demands\_data['Latitude'], c='blue', label='需求点')

# 绘制飞行路径

for task in task\_allocation:

demand\_id, station\_id, net\_cost, drone\_type = task

demand = demands\_data.loc[demands\_data['Demand\_ID'] == demand\_id].iloc[0]

station = stations\_data.loc[stations\_data['Station\_ID'] == station\_id].iloc[0]

ax.plot([station['Longitude'], demand['Longitude']], [station['Latitude'], demand['Latitude']],

label=f'无人机类型 {drone\_type}')

ax.legend()

ax.set\_xlabel('经度')

ax.set\_ylabel('纬度')

ax.set\_title(title)

plt.show()

# 可视化问题3的飞行路径

visualize\_flight\_paths(task\_allocation, '问题 2-飞行路径')

# 打印任务分配和路线时刻表

print\_task\_allocation(task\_allocation)

print\_route\_schedule()

print(f"问题2最小净成本：{total\_cost} 元")

**3.第三问**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.pyplot as plt

from matplotlib import rcParams

rcParams['font.sans-serif'] = ['SimHei']

rcParams['axes.unicode\_minus'] = False

# 公交站点数据

stations\_data = pd.DataFrame({

'Station\_ID': [1, 2, 3, 4, 5, 6, 7, 8, 9],

'Longitude': [110.125713, 110.08442, 110.029866, 109.962839, 109.956003, 109.920425, 109.839046, 109.823329,

109.767127],

'Latitude': [32.815024, 32.771676, 32.748994, 32.743622, 32.812194, 32.856136, 32.860495, 32.847468, 32.807855]

})

# 需求点数据

demands\_data = pd.DataFrame({

'Demand\_ID': range(1, 51),

'Longitude': [

110.1053385, 110.1147032, 110.0862574, 110.0435344, 110.0575508,

110.0386243, 110.0115086, 110.0390602, 110.0246454, 110.0575847,

109.9456331, 109.9612274, 109.94592, 109.9316682, 109.9245376,

109.7087533, 109.7748005, 109.7475891, 109.7534532, 109.783015,

109.7410728, 109.7554844, 109.7147417, 109.8807093, 109.8070677,

109.9054481, 109.8954509, 109.8979229, 109.8942179, 109.8610985,

109.8744682, 109.8338804, 109.870924, 109.8292467, 109.8711312,

109.8813363, 109.978788, 109.8166563, 109.8151216, 109.885638,

109.9890984, 109.9647812, 109.9303732, 109.9401099, 109.944496,

109.979708, 109.976757, 109.94999, 109.973673, 109.967765

],

'Latitude': [

32.77881526, 32.75599834, 32.74905239, 32.74275416, 32.76712584,

32.70855831, 32.72619993, 32.73965997, 32.72360718, 32.76553658,

32.7526657, 32.72286471, 32.70899877, 32.73848444, 32.70740885,

32.7815564, 32.80016336, 32.80903496, 32.85129032, 32.82296929,

32.82914197, 32.80581363, 32.79995734, 32.89696579, 32.79622985,

32.89437141, 32.86724756, 32.83444574, 32.83224374, 32.90687042,

32.89939698, 32.85616627, 32.848223, 32.83825122, 32.88979101,

32.8642824, 32.75943454, 32.8096699, 32.82822489, 32.84032485,

32.80854774, 32.80993619, 32.78956582, 32.85264625, 32.802178,

32.817449, 32.811064, 32.795207, 32.746858, 32.820998

],

'Delivery\_kg': [3, 4, 2, 0, 8, 7, 4, 9, 10, 6, 7, 12, 3, 5, 6, 5, 3, 13, 12, 3,

14, 10, 4, 34, 6, 6, 3, 4, 20, 5, 6, 5, 3, 15, 2, 6, 3, 4, 3, 2,

6, 5, 9, 3, 3, 4, 6, 4, 4, 0],

'Pickup\_kg': [0, 0, 5, 3, 0, 0, 3, 0, 0, 5, 0, 0, 0, 2, 0, 4, 0, 0, 0, 6, 0, 0, 0, 10, 0, 0, 0, 7, 0, 0, 0, 0, 8, 0,

0, 0, 0, 0, 6, 0, 0, 14, 0, 0, 0, 13, 0, 0, 0, 9]

})

# 无人机参数

drone\_params = {

'A': {'D\_max': 27, 'Q\_max': 9, 'C\_fixed': 80, 'C\_per\_km': 0.8},

'B': {'D\_max': 16, 'Q\_max': 30, 'C\_fixed': 150, 'C\_per\_km': 1.5},

'C': {'D\_max': 18, 'Q\_max': 50, 'C\_fixed': 200, 'C\_per\_km': 2.5}

}

# 公交车参数

bus\_speed = 35 # 公交车速度（km/h）

# 发车时间表

bus\_schedule = {

'白河至仓上': [6.67, 8.5, 9, 11, 14, 16.5],

'仓上至白河': [6, 7.33, 8.83, 11, 14, 15.83]

}

# 计算两个点之间的距离

def haversine(lon1, lat1, lon2, lat2):

lon1, lat1, lon2, lat2 = map(np.radians, [lon1, lat1, lon2, lat2])

dlon = lon2 - lon1

dlat = lat2 - lat1

a = np.sin(dlat / 2) \*\* 2 + np.cos(lat1) \* np.cos(lat2) \* np.sin(dlon / 2) \*\* 2

c = 2 \* np.arcsin(np.sqrt(a))

r = 6371

return c \* r

# 公交站点和需求点的经纬度

stations\_lons = stations\_data['Longitude'].values

stations\_lats = stations\_data['Latitude'].values

demands\_lons = demands\_data['Longitude'].values

demands\_lats = demands\_data['Latitude'].values

# 计算所有公交站点到所有需求点的距离

stations\_lons\_matrix, demands\_lons\_matrix = np.meshgrid(stations\_lons, demands\_lons)

stations\_lats\_matrix, demands\_lats\_matrix = np.meshgrid(stations\_lats, demands\_lats)

distances\_matrix = haversine(stations\_lons\_matrix, stations\_lats\_matrix, demands\_lons\_matrix, demands\_lats\_matrix)

distances = {(stations\_data['Station\_ID'][i], demands\_data['Demand\_ID'][j]): distances\_matrix[j, i]

for i in range(len(stations\_data)) for j in range(len(demands\_data))}

def allocate\_tasks\_and\_calculate\_costs\_with\_pickup():

total\_cost = 0

total\_revenue = 0

task\_allocation = []

demand\_visited = set()

station\_drone\_usage = {station\_id: {'A': 0, 'B': 0, 'C': 0} for station\_id in stations\_data['Station\_ID']}

for demand\_id, demand in demands\_data.iterrows():

if demand\_id in demand\_visited:

continue

best\_net\_cost = float('inf')

best\_station = None

best\_drone = None

for drone\_type, params in drone\_params.items():

D\_max = params['D\_max']

Q\_max = params['Q\_max']

C\_fixed = params['C\_fixed']

C\_per\_km = params['C\_per\_km']

for station\_id, station in stations\_data.iterrows():

station\_id = station['Station\_ID']

distance = distances[(station\_id, demand['Demand\_ID'])]

if distance <= D\_max and demand['Delivery\_kg'] + demand['Pickup\_kg'] <= Q\_max:

cost = C\_fixed if station\_drone\_usage[station\_id][drone\_type] == 0 else 0

cost += 2 \* distance \* C\_per\_km # 往返距离

revenue = demand['Pickup\_kg'] \* 0.5

net\_cost = cost - revenue

if net\_cost < best\_net\_cost:

best\_net\_cost = net\_cost

best\_station = station\_id

best\_drone = drone\_type

if best\_station is not None and best\_drone is not None:

total\_cost += best\_net\_cost

total\_revenue += demand['Pickup\_kg'] \* 0.5

task\_allocation.append((demand['Demand\_ID'], best\_station, best\_net\_cost, best\_drone))

station\_drone\_usage[best\_station][best\_drone] += 1

demand\_visited.add(demand\_id)

return total\_cost, total\_revenue, task\_allocation

total\_cost\_with\_pickup, total\_revenue, task\_allocation\_with\_pickup = allocate\_tasks\_and\_calculate\_costs\_with\_pickup()

def generate\_flight\_schedule\_with\_pickup(task\_allocation):

wait\_time = 5 / 60 # 等待时间（小时）

flight\_schedule = []

for direction, times in bus\_schedule.items():

for time in times:

for task in task\_allocation:

demand\_id, station\_id, net\_cost, drone\_type = task

arrival\_time = time

distance = distances[(station\_id, demand\_id)]

flight\_time = distance / (bus\_speed / 2)

start\_time = arrival\_time + wait\_time

end\_time = start\_time + flight\_time

flight\_schedule.append({

'Demand\_ID': demand\_id,

'Drone\_Type': drone\_type,

'Station\_ID': station\_id,

'Flight\_Start\_Time': start\_time,

'Flight\_End\_Time': end\_time

})

return flight\_schedule

flight\_schedule\_with\_pickup = generate\_flight\_schedule\_with\_pickup(task\_allocation\_with\_pickup)

# 任务分配

def print\_task\_allocation(task\_allocation):

allocation\_by\_station = {}

for task in task\_allocation:

demand\_id, station\_id, net\_cost, drone\_type = task

if station\_id not in allocation\_by\_station:

allocation\_by\_station[station\_id] = {}

if drone\_type not in allocation\_by\_station[station\_id]:

allocation\_by\_station[station\_id][drone\_type] = []

allocation\_by\_station[station\_id][drone\_type].append(demand\_id)

print("无人机任务分配:")

for station\_id, drones in allocation\_by\_station.items():

for drone\_type, demands in drones.items():

print(f" 公交站点 {station\_id}: 需求点 {demands} -> 无人机类型 {drone\_type}")

# 路线和时刻表信息

def print\_route\_schedule():

print("路线: 白河至仓上")

for time in bus\_schedule['白河至仓上']:

arrival\_times = time + (distances\_matrix[:, 0] / bus\_speed)

print(f" 出发: {time}h")

print(f" 到达时间: {arrival\_times}")

print("路线: 仓上至白河")

for time in bus\_schedule['仓上至白河']:

arrival\_times = time + (distances\_matrix[:, 0] / bus\_speed)

print(f" 出发: {time}h")

print(f" 到达时间: {arrival\_times}")

# 飞行路径

def visualize\_flight\_paths(task\_allocation, title):

fig, ax = plt.subplots()

ax.scatter(stations\_data['Longitude'], stations\_data['Latitude'], c='red', label='公交车站')

ax.scatter(demands\_data['Longitude'], demands\_data['Latitude'], c='blue', label='需求点')

for task in task\_allocation:

demand\_id, station\_id, net\_cost, drone\_type = task

demand = demands\_data.loc[demands\_data['Demand\_ID'] == demand\_id].iloc[0]

station = stations\_data.loc[stations\_data['Station\_ID'] == station\_id].iloc[0]

ax.plot([station['Longitude'], demand['Longitude']], [station['Latitude'], demand['Latitude']],

label=f'无人机类型 {drone\_type}')

ax.legend()

ax.set\_xlabel('经度')

ax.set\_ylabel('纬度')

ax.set\_title(title)

plt.show()

#问题3的飞行路径

visualize\_flight\_paths(task\_allocation\_with\_pickup, '问题 3-飞行路径(有取货)')