最佳指挥官

摘要

军事物资的补给是战争中一项非常重要的工作，一场战争陷入较为持久的战斗时则需要调动补给站中的后勤部队快速补充物资，所以研究战争物资调度问题很有意义。

针对问题一，首先将题目要求转化为相应的数学问题，以新建补给站到各个部队的总距离最短表示最大程度给各个部队提供支援，由此可以建立无约束优化模型，自变量即新建补给站的位置坐标。本文采用两种方法求解无约束问题，分别为梯度下降法和鲸鱼优化算法。两种算法求解的结果分别为（10.49，5.05）和（9.0055，4.9992），对比发现，鲸鱼优化算法求解的最优位置正是部队 A122，因此从战争因素考虑将补给站设置在（10.49，5.05）最好，到四个部队的总距离为 15.3737km。

针对问题二，首先，将题目要求的确定飞机调用方案转化为最小化完成所有运输任务的时间，并以此为目标函数，最晚需求时间、飞机超出物资数量要求、飞机总数量为约束条件建立单目标整数规划模型，明确自变量为 6 架飞机 A 和 10 架飞机 B 的调度方案。考虑智能优化算法在本问中对产生新解的要求过高，本问采用贪心算法求解模型二。结合两种飞机的属性，将每次调度均选择距离补给站最远的部队和优先考虑 B 类飞机为贪心方式，求解得到的结果是派遣 4 架 B 飞机去部队 A102、5 架 A 飞机和 1 架 B 飞机去部队 A121、7 架 A 飞机去部队 B121 和 5 架飞机去部队 B404，完成所有任务的运输时间为 102.01 分钟。最后，改动贪心算法的贪心方式对模型进行检验，结果显示，改变后的贪心算法并不能改变最短时间，但是会影响多出物资的数量，证明了模型二是可靠的。

针对问题三，首先，在问题二模型的基础上，引入最小化多出物资为第二个目标函数，调整约束条件超出物资的数量，并考虑飞机运输过程中的损失因素，引入损失因子概念，将不确定的损失因子规定为范围内的期望值，建立多目标整数规划模型。模型三是多目标优化问题，因此贪心算法在模型三中存在很大的限制，本文采用模拟退火算法求解问题三，将最小化完成运输任务的时间转化为约束条件，并考虑新解的产生固定在问题的解空间内，经过 150 次迭代后算法找到的最优解为：派遣 2 架 A 飞机和 3 架 B 飞机去部队 A102、4 架 B 飞机去部队 A122、6 架 A 飞机和 1 架 B 飞机去部队 B121、4 架 A 飞机和 2 架 B 飞机去部队 B404。此外，模拟退火求解的物资总数为 334.3 吨，相比于贪心算法的结果 370.375 吨提高了 10.79%，最短时间仍然可以达到 102.01 分钟。最后，通过调整损失因子的值对参数损失因子进行灵敏度检验，结果显示，当物资损失的程度在期望值的小范围内波动时，模型求解的结果并不会存在很大差异，但是当损失程度过大时，飞机的调度方案将会发生大变化，此时完成所有运输任务的最短时间将会增加 46.98%。

关键词：优化模型 、鲸鱼优化算法、梯度下降法、贪婪算法、模拟退火、灵敏度检验

一、问题重述

2.1 问题背景

近日俄罗斯和乌克兰冲突加剧，即便是在科技极度发达的现代，战争依旧是以人为主体，并没有进入完全机械化的战争状态。正所谓“军事以人为本”，为了保障最基本的人，在战争当中的军事补给便显得十分重要，中国的古代兵法中，早早指明“兵马未动，粮草先先行。”可以说，军事补给在军事战争中起到了举足轻重的作用。在现代战争中，考虑到多方面因素，士兵往往不会一次性带上大量的食物和弹药，倘若陷入了耗时较长的持久战时，便需要调动补给站中的后勤部队，快速为士兵补充物资。

在某处补给站附近，同时有 4 支部队需要物资补给，但是补给站仅有 A、B 两种类型飞机。A 类飞机共有 6 架，每架所能承受的最大载重量为 13 吨，飞行速度为 260km/h；B 类飞机有 10 架，美架所能承受的最大载重量为 20 吨，飞行速度为 260km/h。飞机可以沿着直线向部队发放物资，并且每次只能向一个目的地投放物资，结束后，要返回补给站，补充物资后，才可以前往下一个目的地。

每个部队所能接受的物资数量不能超过所需数量的 150%，在物资装卸的过程中 A 类飞机均需要 20min;B 类飞机装载货物需要 30min;卸载货物需要 40min。

四支部队最晚需求时间如图所示:

部队编号	平面坐标 km	物资需求量 t	最晚需求时间 min
A102	(5,4)	80	450
A122	(9,5)	75	430
B121	(13,6)	90	410
B404	(16,4)	85	400

表格 1　4 支部队的坐标、请求物资的数量和最晚的需求时间表

2.2 问题描述

问题一：想要最大程度给给个部队提供支援，则要在最短时间内将物资运送到部队，在不考虑其他因素的前提之下，距离四个部队的距离总和最小，则时间最短。所以要建立模型，求出四个距离最短的点，该点则为新补给站的最佳修建点；

问题二：想要飞机最好的调用方案，则要求在最短时间内可以将四个部队想要的物资送达;

问题三：由于部队的移动，所能接受的物资数量不能超过所需的 110%，同时由于部队的移动，每为下一个部队提供物资时，要考虑到转移到该部队的上部队，所提供物资应为所有转移到该部队的综合，除此之外也要考虑到在运输过程中所损耗的物资，在这些前提下，调动飞机，在最短时间内完成物资支援；

二、问题分析

本题要求我们作为飞机调度的决策者，解决三个不同情况下的优化问题。解决这类纯优化类问题，解题思路主要分为两步：第一步明确题目的目标函数，找到题目要求的约束条件；第二步采用优化方法求解模型；

对于问题一无约束优化问题，可以借助梯度的方法求解，也可以采用智能优化算法进行求解。对于问题二有约束的整数规划问题，最简单的求解办法就是指

定贪心规则，采用贪婪算法求解。对于问题三，需要考虑约束条件的变化以及双目标的选择，本文采用模拟退火算法进行求解。

3.1 问题一的分析

问题一中假设旧的补给站被敌军炸毁，需要紧急建设新的补给站。不考虑其它因素，仅仅考虑能最大程度给各个部队提供支援，该题的思路分为两个步骤：

1. 确定题目要求的考虑最大程度给各个部队提供支援的意义，将建站问题转化为数学问题。对附件表格中的数据进行分析，我们发现可以将最大程度给各个部队提供支援理解为新建站到四个部队的总距离最小，于是我们就可以建立无约束优化模型；
2. 求解无约束的方法各种各样，最主流的方法就是基于梯度的方法，例如梯度下降法、牛顿法和拟牛顿法等等，不基于梯度的方法例如演化算法和群智能算法等等，常见的有模拟退火、遗传算法和粒子群算法等等，本文选择了一种求解能力相较于常规算法更强的群智能优化算法——鲸鱼算法。并对两种算法求解的结果进行了对比分析，给出了在不同现实情况下的建站方案。

3.2 问题二的分析

在问题二中，由于种种原因，新补给站被修建在坐标为（10，30）的位置，让我们建立数学模型确定飞机的调用方案。

在知道新建补给站的坐标位置后，我们就可以使用给定的坐标位置来调度飞机运送物资了，至于如何调度飞机就是第二问需要确定的。同第一问思路类似，我们必须先找到合适的目标函数和约束条件，从而将飞机调度问题转化为规划问题。本文需要考虑多种因素，比如物资全部运送完成的时间不得超过该部队的最晚需求时间、所有飞机的存在最大载重等等。很明显，这是一个整数规划模型，自变量是 16 架飞机的调度情况。目标函数题目并没有明确给出，但是在考虑到实际情况下将目标函数确定为最小化完成运输时间。解决整数规划的方法有很多，例如模拟退火，贪婪算法等等均可，如何选择最优的解决方法需要考虑题目所给的约束条件，本文采用贪婪算法来求解规划模型，相较于模拟退火算法，贪婪算法更加简单，并且贪婪算法不需要像模拟退火一样每次生成在解空间的解，贪婪算法是逐步迭代的，每一步都取最优的，最后总结果即是最优的。最后，我们对贪婪算法求解的结果进行了检验分析，给出了更加精确的改进方案。

3.3 问题三的分析

问题三是在问题二的基础上加上新的约束条件，即题目明确规定了各个部队需要在收到补给后继续移动前往下一个地点执行任务，因此每个部队所存储的物资量不能超过所需数量的 110%。且在第三问新增加了搬运过程的损耗，且 A 类飞机的损耗和 B 类飞机的损耗不同，这就对模型求解的结果造成了一定的随机性，对于这类带有随机性的问题，我们一般解决的办法有两种。第一种就是将随机性因素转化成固定的因素，从而减小复杂度，但是这样做的结果就是模型的鲁棒性不强。另一种方法就是保持随机性，且每次的决策都需要考虑随机因素，显然这样做将会加大题目的难度。除此之外，在第三问中我们考虑到最优调度不仅仅是时间上最优，调度的结果也需要最优，于是我们引入最小化多出物资为第二个目标函数，建立了多目标整数优化模型。为了简化求解的难度，我们先固定了损耗因子的值，并将最小化运输时间转化为约束条件，从而使双目标模型变为单目标模型，采用模拟退火算法求解模型，最后对模型的损耗因子参数进行了灵敏

度分析。

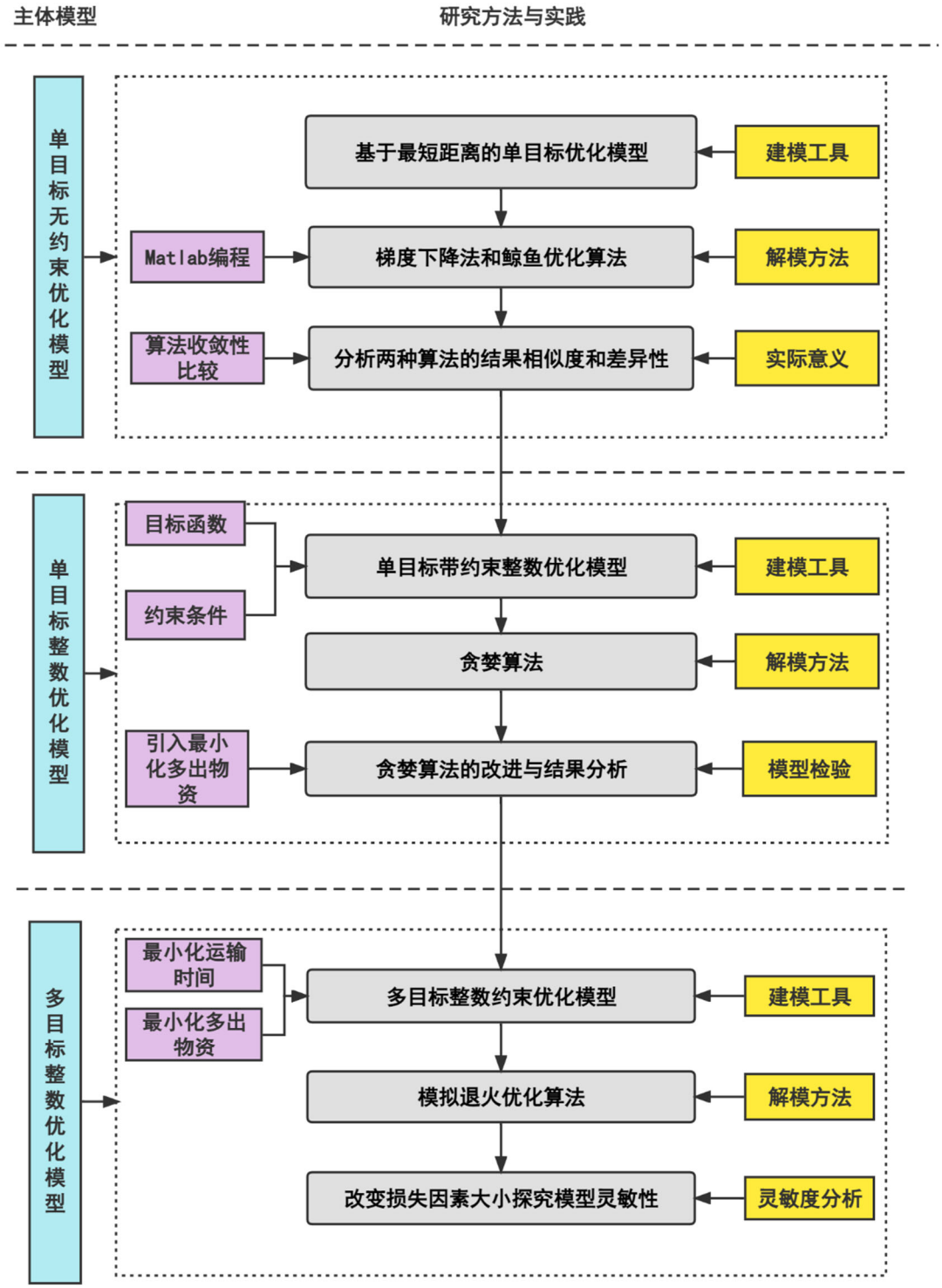


图 1 总体思路图

三、模型假设

一、 假设飞机在运送过程中都是装满出发的，不存在一架飞机未装满就出发的情况；

二、 假设所有的飞机可以同时起飞、同时出发，并且支持并行；

三、 问题三的补给站位置就是第二问的补给站位置；
四、 飞机在飞行过程中不会存在耽误时间，默认用给定速度恒定飞行；

四、符号说明

符号	定义
$(\overline{x},\overline{y})$	补给站的位置坐标
η	梯度下降法的学习率
X_i	第 i 架飞机的调度情况
t_{Aj}	飞机 A 到第 j 个部队的时间
t_{Bj}	飞机 B 到第 j 个部队的时间
s_j	补给站到第 j 个部队的距离
v_A	A 类飞机的飞行速度
v_B	B 类飞机的飞行速度
q_1	A 类飞机的装货时间
q_2	A 类飞机的卸货时间
p_1	B 类飞机的装货时间
p_2	B 类飞机的卸货时间
K_A	A 类飞机的最大承重量
K_B	B 类飞机的最大承重量
T_j	第 j 个部队的最晚需求时间
M_j	第 j 个部队的所需物资数量
α	A 类飞机的损耗因子
β	B 类飞机的损耗因子
L	模型一的目标函数
W	模型二的目标函数

五、模型的建立与求解

5.1 模型一的建立与求解

5.1.1 模型的建立

题目明确指出不考虑其它因素，仅仅考虑能最大程度给各个部队提供支援。在战场上，任何时间都是宝贵的，物资的提供更是战士们的救命稻草，所以选择合适的建站地点非常重要，不考虑其它因素，就不用管是否建站地点在战争区这种特殊情况。于是我们可以将模型简化为所建补给站到剩余四个部队的距离最小，将如何建站问题转化为一个无约束问题，具体来看。
先绘制四个部队的位置坐标如下图：

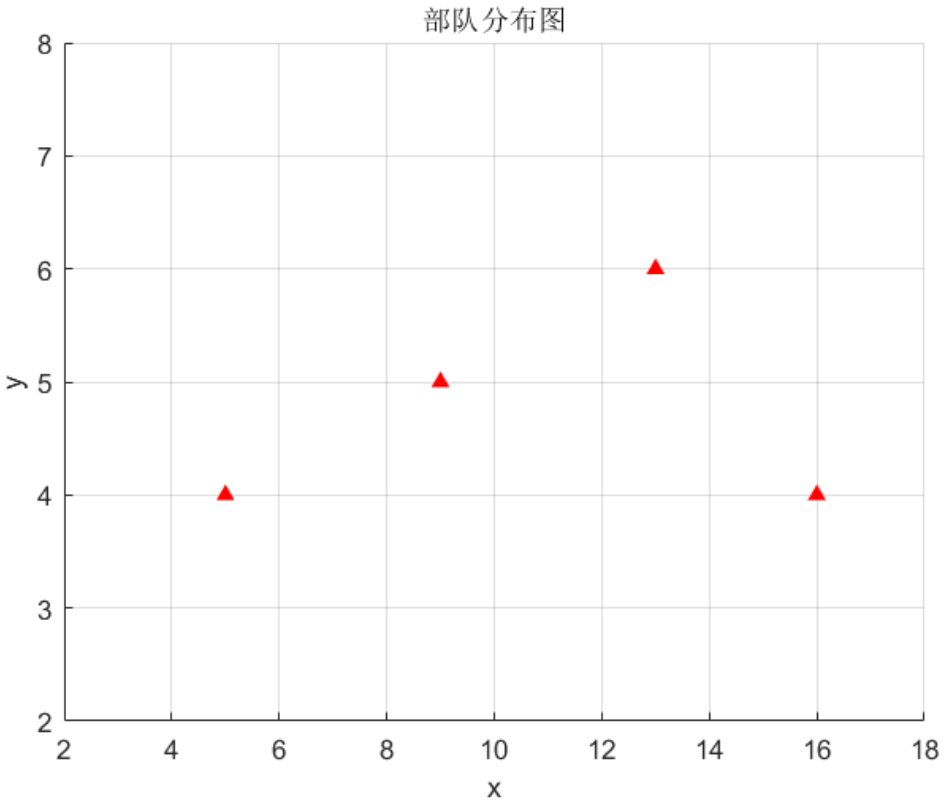


图 2 四支部队的位置分布图

假设新建的补给站的位置坐标为$(\bar{x},\bar{y})$，即我们需要找到最合适的$\bar{x}$和$\bar{y}$，使得补给站到剩余四个部队的距离最小。设 A102、A122、A121、B404 这四个部队的坐标为(x_i,y_i)，其中$i=1,2,3,4$。那么可以得到如下的目标函数：

$$\sum_{i}^{4}\left[\left(x_i-\bar{x}\right)^2+\left(y_i-\bar{y}\right)^2\right]$$

而我们的目的就是最小化目标函数：

$$\min\sum_{i}^{4}\left[\left(x_i-\bar{x}\right)^2+\left(y_i-\bar{y}\right)^2\right]$$

min 的含义是最小化。

5.1.2 模型的求解

5.1.2.1 求解方法的选择

此模型是一个典型的无约束优化问题，求解此类无约束优化问题最常用的方法主要有两种：基于梯度的方法和群智能搜索算法。为了在求解此类模型的同时对比两类方法的有效性，本文采用两类方法中最具有代表性的算法分布求解无约束模型，它们分别是梯度下降法和鲸鱼算法。使用两种方法来求解模型不仅能起到检验结果是否正确的作用，还能互相对比两类算法在解决无约束问题上的优劣性。

5.1.2.2 梯度下降法

梯度下降法[1]（Geadient descent）是一种常用的一阶优化算法，是求解无约束优化问题最简单、最经典的方法之一。要使用梯度下降法，我们得找到目标函数的偏导数，例如在本文中，假设目标函数为L，则目标函数的偏导数分别为：

$$\frac{\partial L}{\partial \bar{x}} = -2\sum_{i}^{4}\left(x_i - \bar{x}\right)$$
$$\frac{\partial L}{\partial \bar{y}} = -2\sum_{i}^{4}\left(y_i - \bar{y}\right)$$

在梯度下降法中，还需要设置学习率，学习率是一个影响算法搜索速率的参数，一般设置为 0.1 到 0.001 之间，本文设置学习率为 0.001。即得到如下的迭代式：

$$\bar{x} = \bar{x} - \eta\frac{\partial L}{\partial \bar{x}}$$
$$\bar{y} = \bar{y} - \eta\frac{\partial L}{\partial \bar{y}}$$

使用上式对自变量 $\bar{x}$ 和 $\bar{y}$ 进行迭代，知道达到满足的精度要求就停止迭代。

5.1.2.3 鲸鱼算法

鲸鱼算法[2][3]也属于群智能优化算法之一，属于比较有代表性的优化算法，提出的俄时间是 2016 年，在求解无约束问题上效果非常好。

在鲸鱼算法中，每个鲸鱼代表一个可行解，在鲸鱼群进行捕猎过程中，每只鲸鱼可以选择两种捕猎方式，即包围猎物和起泡网捕食，包围猎物又分为两种，分别是向最优的鲸鱼移动和向随机鲸鱼游动。从捕猎方式上来看气泡捕食应该是算法的局部搜索，向着最优的鲸鱼移动应该是加快收敛速度的，随机移动应该是算法的全局搜索。并且第 i 个鲸鱼在选择捕猎方式是围绕捕食还是气泡捕食的概率是相等的，即两者各占 0.5。

假设鲸鱼的种群规模为 N，求解问题的维度为 D（即自变量的个数）。则第 i 只鲸鱼在 D 维空间中的位置为:

$$X_i = (x_i^1, x_i^2, \cdots, x_i^D), i = 1, 2, \cdots N$$

在本文中维度 D 等于 2。其中在 N 个鲸鱼种群中，最优鲸鱼的位置即为猎物的位置，其中每只鲸鱼都会以 0.5 的概率选择围捕捕食或者气泡捕食。下面是这两种捕食方法的详细介绍

✧ **通过包围猎物进行捕食**

➢ **向着最优的鲸鱼进行包围**

如果第 i 只鲸鱼选择了向着最优的鲸鱼游动，那么鲸鱼首先会观察猎物的位置在何处确定好猎物的位置后，其他鲸鱼便向着该鲸鱼的位置游去从而更新自身的位置，所以，游动第一步需要先求解该个体与最优鲸鱼（猎物）之间的距离：

$$D = \left|C \cdot X^*(t) - X(t)\right|$$

式中 t 表示当前迭代的次数，$X^*(t)$ 表示第 i 代中最优鲸鱼的位置（即猎物的位置），$X(t)$ 为第 t 代中某个鲸鱼的位置。并且 $X^*(t)$ 随着迭代而更新，常数 C 为摆动因子，摆动因子 C 的计算方式如下

$$C = 2 * r$$

鲸鱼位置更新公式为：

$$X(t+1) = X^*(t) - AD$$

其中，A 为收敛因子，其计算公式如下：

$$A = 2 \cdot a \cdot r - a$$

r 为[0,1]之间的随机数，a 是随着迭代次数的增加从 2 线性减小到 0。

➢ **随机搜寻鲸鱼进行包围**

鲸鱼在捕食的过程中会跟着伙伴们的位置变化而更新自身位置，也就是说其实鲸鱼在某阶段不会向着最优鲸鱼（猎物移动），他们会向着随机鲸鱼移动。为了确定什么阶段鲸鱼会向着最优鲸鱼移动，采用将 A 的模与 1 进行比较，若 A 的绝对值大于 1，则鲸鱼进行全局搜索，若 A 的模小于 1，鲸鱼向着最优的鲸鱼移动，而从上面的公式中可以看出 A 的取值范围是-a 到 a，而 a 的值又是随着迭代次数而下降的。可以这样说，在算法前期，由于 a 比较大，所以 A 的模大于 1 的可能性比较大，鲸鱼随机游动的可能性比较大，这代表着鲸鱼算法前期的全局搜索能力较大 ，下面来看随机游动的公式：

$$D = \left| C \cdot X_{rand} - X \right| \tag{1}$$

$$X(t+1) = X_{rand} - A \cdot D \tag{2}$$

式中的 X_{rand} 表示当前种群中的随机一只鲸鱼（或者猎物的位置）。

✧ **通过吐气泡进行捕食**

如果鲸鱼选择了气泡捕食，鲸鱼首先计算出自身到最优鲸鱼之间的位置，接着鲸鱼以螺旋形姿势向上游动并吐出大小不等的气泡捕食鱼虾。

$$D' = \left| X^*(t) - X(t) \right| \tag{3}$$

$$X'(t+1) = D' \cdot e^{bl} \cdot \cos(2\pi l) + X^*(t) \tag{4}$$

其中 D 为第 i 只鲸鱼到食物的距离， L 为[-1,1]之间的随机值， b 为螺旋常数。

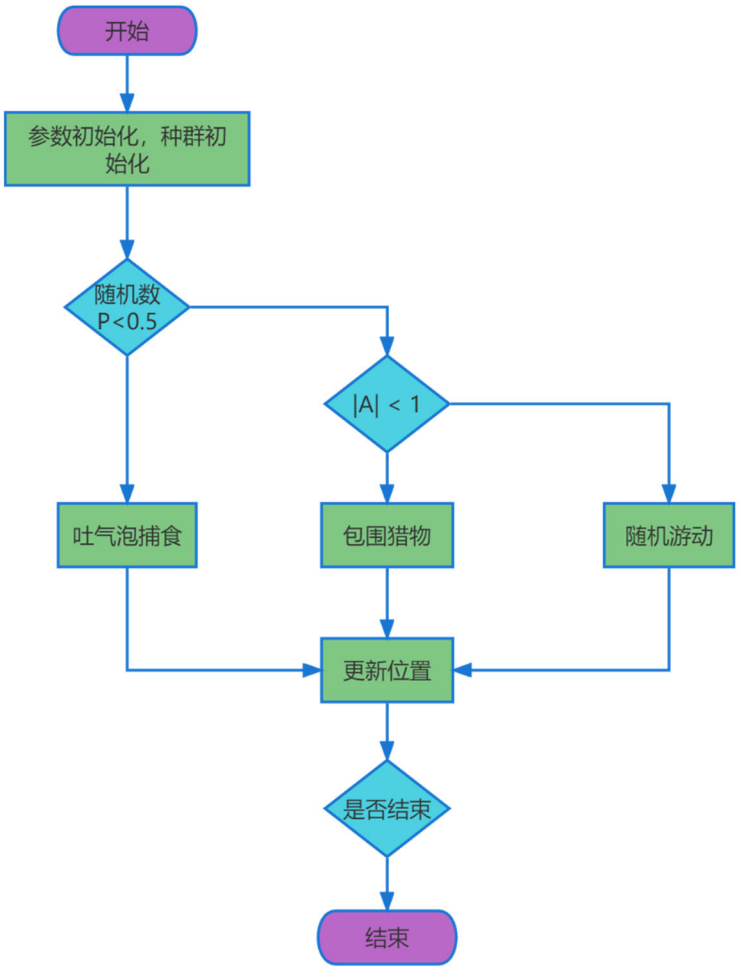


图 3 鲸鱼算法流程图

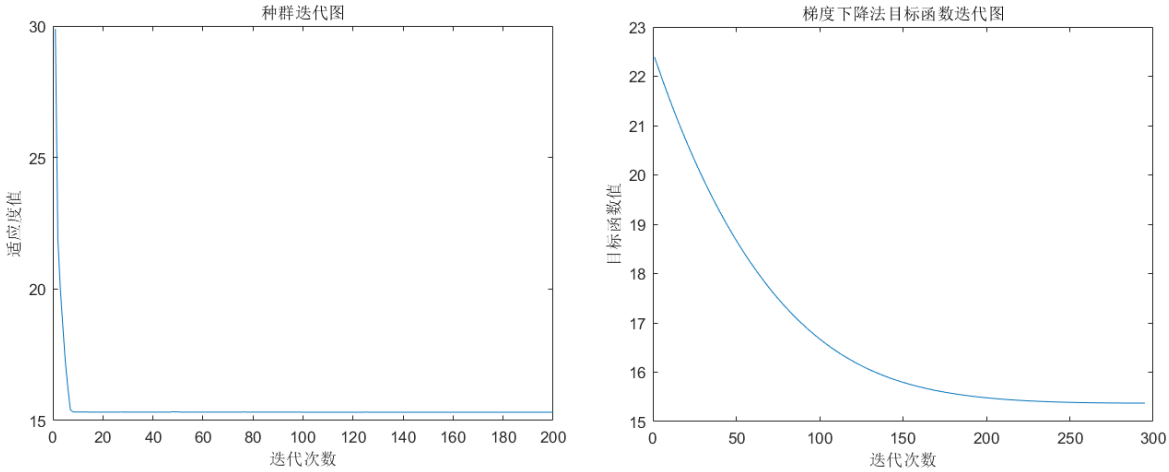
如果第i只鲸鱼选择了包围捕食，那么这只鲸鱼又将选择是进行围绕最优鲸鱼游动还是随机选择一只鲸鱼游动。从字面上也可以看出，随机选择一只鲸鱼应该就是全局搜索了，而围绕最优鲸鱼搜索其实就是趋于最优解，但是我们不知道这个解是不是全局最优，所以在选择两种围捕捕食时，需要考虑算法整体的运行程度。

由于本问的自变量只有一个大于 0 的约束，所以在生成解空间时可以将解空间设置为 0 到 100 内。

5.1.3 模型求解的结果

借助 Matlab 编程，实现最优化问题的梯度下降法和鲸鱼算法的求解。

设置学习率η的值为 0.001，梯度下降法的初值为（8，8）。鲸鱼算法设置种群数量为 100 个，迭代次数 2000，螺旋系数为 3，游动因子初始值为 2，下降方式为线下下降。两种方法求解的结果如下：



从运行迭代图中看出，鲸鱼算法在第 10 代得时候就接近收敛了，但是梯度下降法是不断迭代的，可以看到，两种算法最后找到的最优值都非常接近。目标函数值和最佳站点设置位置如下：

优化方法	梯度下降法	鲸鱼优化算法
目标函数值	15.3737	15.3137
最佳建站地点	(10.49,5.05)	(9.0055,4.9992)

表格 2 两种算法求解结果图

5.1.4 模型结果的分析

在 5.1.3 节的求解结果中我们可以发现，鲸鱼算法和梯度下降法的求解结果存在很小的差距，下面绘制两种算法求解得到的最优补给站站点位置图：

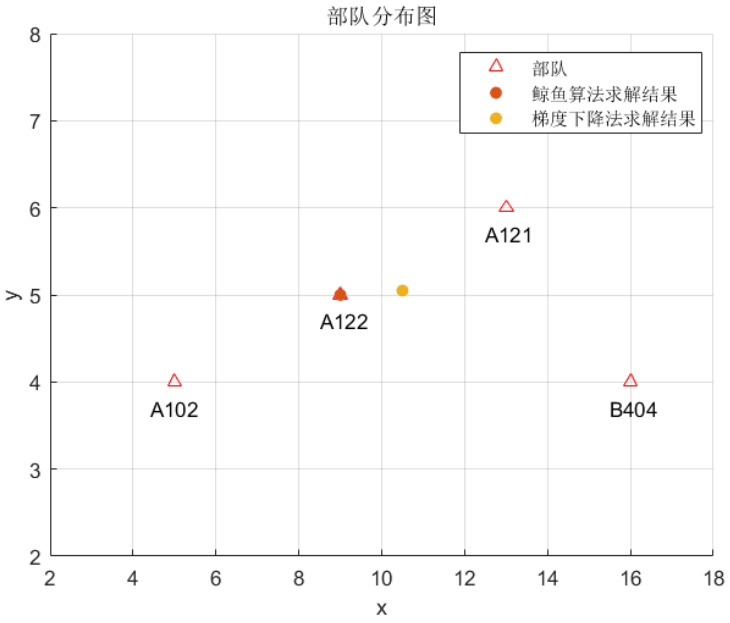


图 4 两种算法求解结果的分布图

从位置图中我们可以看到，当选择优化方法是鲸鱼优化算法时，得到的最佳补给站点位就是部队 A122，而使用梯度下降法作为求解方法时，得到的最佳补给站点是地图上的某一点，而两种方法运行的结果相差非常非常小。这就为我们设置补给站点提供了多种思路，如果考虑现实战争的因素，那么设置站点为(10.49,5.05)可能更好，因为这时的补给站不在部队区，可以避免被轰炸等因素。如果仅仅考虑最快到达各个部队，那就设置补给站站点在部队 A122 处，这样可以最快到达各个部队，从而为各个部队提供及时的物资。

5.2 模型二的建立与求解

5.2.1 模型的建立

问题二要求我们在给定补给站位置情况下，找到最优的飞机调度安排。题目提供的补给站位置为（10，30），我们可以借助 Matlab 绘制出补给站和部队的示意图，结果如下：

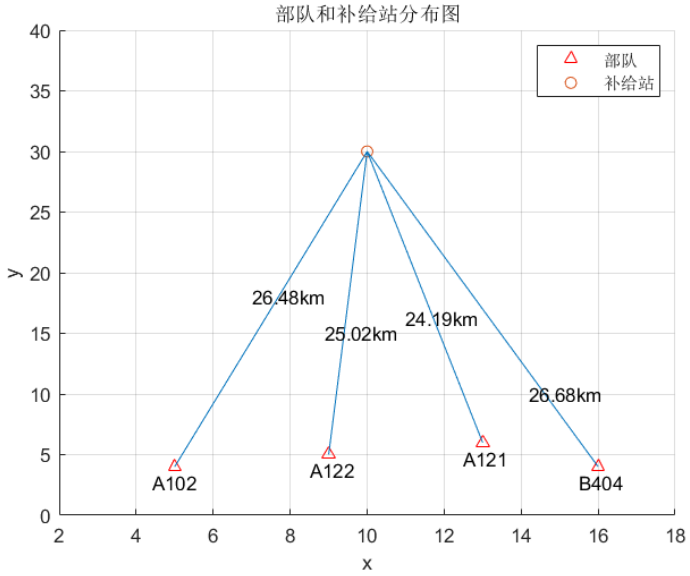


图 5 补给站和部队的分布图

绘制补给站和部队分布图过程中，我们还计算了补给站到所有部队之间的距离，并在图中展示出来。从图中可以看出，补给站在（10，30）位置处到四个部队的距离相差都不是很大。下面给出建模的全过程。

◆ **符号定义**

要想将调度问题转化为数学模型[4]，那就需要找到合适的目标函数。在第一问中我们以建站点到其它四个部队的总距离最小为目标函数来建立模型，但是在这一问中，我们要考虑的因素比第一问要多。首先设 16 架飞机的调度变量为 $X_i=[x_{i1},x_{i2},x_{i3},x_{i4}]$，其中 $i=1,2,\cdots 16$，X_i 表示第 i 架飞机的调度分配情况，也即自变量。假设飞机 A 和飞机 B 从补给站到四个部队所用的时间为 t_{Aj} 和 t_{Bj}，其中 $j=1,2,3,4$，下面是 t_{Aj} 和 t_{Bj} 的计算方式：

$$t_{Aj}=\frac{s_j}{v_A}\cdot 60$$

$$t_{Bj}=\frac{s_j}{v_B}\cdot 60$$

上式中，s_j 表示补给站到第 j 个部队的距离，单位是 km，v_A 和 v_B 是 A B 两种飞机的速度，单位是 km/h，我们在计算式后乘以 60，将小时转化为分钟。其次定义飞机 A 装卸货物的时间分别为 q_1 和 q_2，飞机 B 装载货物的时间为 p_1 和 p_2

◆ **目标函数**

确定本问的目标函数是解决本问的核心问题，在战争中，战事瞬息万变，而物资运送的时间应该算是最重要的，于是我们将最佳调度方案理解为使用此方案得到的最佳运送时间是最短的，于是可以将目标函数定义为：

$$\min\left\{\begin{matrix}\max\{(2x_{ij}-1)\cdot t_{Aj}+2x_{ij}(q_1+q_2),i=1,2,\cdots 6\\(2x_{ij}-1)\cdot t_{Bj}+2x_{ij}(p_1+p_2),i=7,8,\cdots 16\}\end{matrix},j=1,2,3,4\right\}$$

第二问的目的就是使得上式目标函数最小化，如果设上面的目标函数为 W，那么就找到了优化的目标为：

$$\min W$$

为了理解为什么 W 要这么写，我们可以将 max 的部分理解为所有部队完成所有运送的最短时间，而每个部队的完成时间又是和自变量 X 所对应。即给定所有的 $X_i=[x_1,x_2,x_3,x_4]$，都对应着一个时间。由于 x_{ij} 是所用飞机的数量，即当 x_{ij} 等于 2 时，所代表的是在第 j 个部队处派出了两次第 i 架飞机。

◆ **约束条件**

由于本问各个部队的最短时间和所需货物量进行了相应的限制，所以会存在约束条件。

约束一： 每只部队可以接受多于所需要的物资，但接受的物资数量不能超过所需数量的 150%，设第 j 个部队所需要的物资数量为 M_j，那么就有：

$$\sum_{i=1}^{6}x_{ij}\cdot K_A+\sum_{i=7}^{16}x_{ij}\cdot K_B\le M_j\cdot 150\%$$

其中 K_A 和 K_B 是 A B 两种飞机的最大承重量。

约束二：由于每只部队接受全部物资的最长时间是有限制的，我们设第 j 个部队最晚需求时间为 T_j，于是有：

$$(2x_{ij}-1)\cdot t_{Aj}+2x_{ij}(q_1+q_2)<T_j, i=1,2,\cdots 6$$
$$(2x_{ij}-1)\cdot t_{Bj}+2x_{ij}(p_1+p_2)<T_j, i=7,8,\cdots 16$$

上式中其实和目标函数中的式子有些相似，但是这个却是每一架飞机到部队的约束条件，即所有的调度方案都需要满足此约束条件。我们来看如果 x_{ij} 等于 0，我们即默认此飞机的运行时间为 0，当 x_{ij} 等于 1 时，即飞机 i 到部队 j 之间需要飞行一次，那么时间就是 $t_{Aj}+q_1+q_2$；当 x_{ij} 等于 2 时，飞机 i 到部队 j 之间需要飞行二次，那么时间就是 $3\times t_{Aj}+3\times(q_1+q_2)$。由此可以推出对于自变量 x_{ij}，其对应的时间就是 $(2x_{ij}-1)\cdot t_{Bj}+2x_{ij}(p_1+p_2)$。

约束三：我们设置的转运方案必须满足物资需求量，即：

$$M_j \le \sum_{i=1}^{6} x_{ij}\cdot K_A + \sum_{i=7}^{16} x_{ij}\cdot K_B$$

约束四：所有的自变量存在约束，即：

$$x_{ij}\ge 0 \ i=1,2\cdots 16, j=1,2,3,4$$

综上所述，我们可以建立模型为：

$$\min\min\left\{\begin{array}{l}\max\left\{(2x_{ij}-1)\cdot t_{Aj}+2x_{ij}(q_1+q_2), i=1,2,\cdots 6\right.\\ \left.(2x_{ij}-1)\cdot t_{Bj}+2x_{ij}(p_1+p_2), i=7,8,\cdots 16\right\}, j=1,2,3,4\end{array}\right\}$$

约束条件为：

$$\begin{cases}\sum_{i=1}^{6} x_{ij}\cdot K_A+\sum_{i=7}^{16} x_{ij}\cdot K_B \le M_j\cdot 150\%, j=1,2,3,4\\ M_j \le \sum_{i=1}^{6} x_{ij}\cdot K_A+\sum_{i=7}^{16} x_{ij}\cdot K_B, j=1,2,3,4\\ (2x_{ij}-1)\cdot t_{Aj}+2x_{ij}(q_1+q_2)<T_j, i=1,2,\cdots 6, j=1,2,3,4\\ (2x_{ij}-1)\cdot t_{Bj}+2x_{ij}(p_1+p_2)<T_j, i=7,8,\cdots 16, j=1,2,3,4\\ x_{ij}\ge 0\end{cases}$$

5.2.2 模型的求解

为了找到求解上述模型的最优方法，我们需要先对现有的数据进行分析。

从补给站和部队的分布图中看出，补给站到四个部队的距离分别为：26.18km、25.02km、24.19km、26.68km。可以看出，部队 B404 是距离补给站最远的部队。飞机 A 的速度为 260km/h，飞机 B 的速度为 50km/h，从这里能看出，飞机 B 的运行速度太慢，如果使用飞机 B 从补给站去往某一个部队的话，加上卸货和载货的时间共为 100 分钟左右，如果使用飞机 A 从补给站去往某一个部队的话，加上载货和卸货的时间总过才为 46 分钟，可以看出，如果追求时间的最小化，那么选择 A 飞机比 B 飞机具有更大的效益。但是题目将 A 飞机和 B 飞机的数量进行了限制，A 飞机只有 6 架，B 飞机有 10 架。

如果所有飞机都派出进行运送的话，第一趟可以承载 278 吨的物资，而所有的部队物资需求量和也才只有 330 吨。这样的话，我们可以设计一种基于贪心思想[5][6]的求解方法，具体来看。

（1） 考虑所有的 B 飞机和部队，向距离补给站最远的部队派出 B 飞机，直到该部队满足物资要求。

（2） 在排完所有 B 飞机后，考虑所有的 A 飞机和未满足物资要求的部队，向未满足物资要求且距离补给站最远的部队派出 A 飞机，直到该部队满足物资要求。

（3） 在派出第一批所有的 A B 飞机后，派遣首先完成第一批运送任务的飞机去未满足物资要求的部队。

（4） 直到所有部队完成运送任务，结束派遣。

上述的方法是比较典型的贪心思想，我们调用了所有的飞机，尽量在最短时间内完成所有的运送任务。贪心选择是每次都选择距离补给站最远的部队，这样做得到的结果可能不是最优的。

采用贪心算法而不是智能优化算法的优点是，贪婪算法在处理约束条件时比智能优化算法更便利，我们在做贪心选择时就可以考虑约束条件，而智能优化算法必须将约束条件考虑进新解的产生中，这样做会造成很大的无用解，从而减小算法的运行效率。

5.2.3 模型求解的结果

借助 matlab 编程，利用贪心算法求解得到结果。

此结果显示，如果按照贪心算法找到的解为最短时间：102.01 分钟，即只需要 102.01 分钟就能完成所有的运送方案，下面是 16 架飞机的调度情况：

调度情况	A102	A122	B121	B404
飞机 A	0	5	7	0
飞机 B	4	1	0	5
多出的物资	0	10	1	15
运送的时间	101.77	100.02	96.93	102.01

将模型求解的结果进行可视化如下:

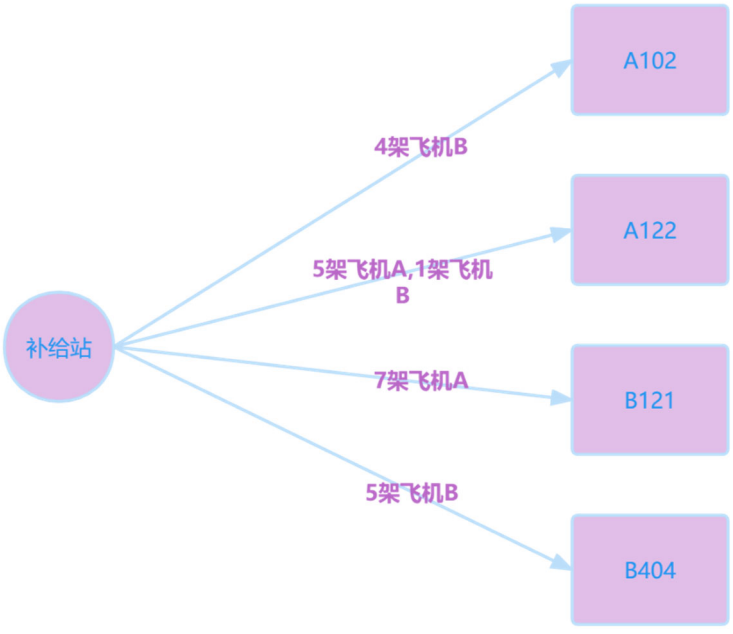


图 6 飞机调度方案示意图

上述的 7 架 A 飞机并不是指同时派出 7 架 A 飞机，而是使用 7 架 A 飞机，即存

在有 A 飞机成功完成运输并返回补给站后再次执行任务。

5.2.3 模型结果的分析

从贪心算法求解的结果上看，从补给站调度的飞机有两种，派遣 4 架 B 飞机去部队 A102、派遣 5 架 A 飞机和一架 B 飞机去部队 A122、派遣 7 架 A 飞机去部队 B121、派遣 5 架 B 飞机去部队 B404。注意贪婪算法运行的结果中所有的 B 飞机只派遣过一次，而多架的 A 飞机是派送过多次的。至于为什么这样，原因可能是 A 飞机的速度远远大于 B 飞机的速度。对多出的物资进行分析，部队 A102 做到没有多余物资，而部队 B404 多出了 15 吨的物资，而这个多出的物资正好小于 150%的范围，所以模型求解的结果是合理的。

5.2.4 模型的检验

由于本模型的解决方法是贪心思想，而贪心算法最大的缺点就是可能陷入一个局部最小值中，而为了验证本问的结果是否是可接受结果范围内，这一节从多出的物资出发，减小多出的物资，从而观察花费的最小时间是否有变化。为了减小多出的物资，我们可以修改 B404 部队的调度方案，从 5 架 B 飞机转变为 1 架 A 飞机和 4 架 B 飞机，由于每次可以使用的飞机数量有明确的规定，所以在改变部队 B404 的调度方案时，其它部队的调度方案也相应的发生变化。具体结果如下：

调度情况	A102	A122	B121	B404
飞机 A	0	3	7	1
飞机 B	4	2	0	4
多出的物资	0	4	1	8

物资全部运送完成的时间为 102.01min，相比于原贪心算法求解的结果不变，但是使用改进后的贪心算法可以让多出的物资减少，这说明贪心算法找到的解并不一定就是最优解，其只是一个局部的最优解，但是我们建立的模型却是合理的。

5.3 模型三的建立与求解

5.3.1 模型的建立

问题三并没有给出补给站的设定位置，而问题三考虑的却是转运方案。所以我们先基于假设三认定在第三问中补给站的设定位置为（10，30）。

在第三问中对飞机 A 和飞机 B 有着损耗的约束，即每种飞机都有着对应的运输损耗，为了量化损耗的影响，我们假设飞机 A 和飞机 B 一个损耗因子为 α 和 β，其中 α 和 β 有固定的范围，范围如下：

$$\alpha \in [0.05, 0.10]$$
$$\beta \in [0.03, 0.07]$$

同第二问类似，在第三问中只让我们确定转运方案，并没有给出最优转运方案的定义。但根据第二问模型求解的结果可以看出，最小运输时间其实和飞机 B 有着很大的关联，而与飞机 A 的关系不大。不同的转运方案对减小最短运输时间影响不大，但是对多出物资数量影响却很大，因此，在问题三中，我们可以在模型二的基础上引入一个新的目标函数，即最小化多出物资：

$$\min \sum_{j=1}^{4} \left(\sum_{i=1}^{6} x_{ij} \cdot K_A (1-\alpha) + \sum_{i=7}^{16} x_{ij} \cdot K_B (1-\beta) - M_j \right)$$

约束条件会存在变化，在第二问中，我们只考虑多出的物资小于所需物资的

150%，这应该是一个非常宽松的约束，并不会对模型的结果造成很大的约束，但是在第三问中，题目明确要求了将 150%缩小到 110%，于是第二问的约束条件变为:

$$\sum_{i=1}^{6} x_{ij} \cdot K_A(1-\alpha) + \sum_{i=7}^{16} x_{ij} \cdot K_B(1-\beta) \le M_j \cdot 110\%, j = 1,2,3,4$$

此外，我们还需要约束每个部队的运输任务成功完成，如下：

$$M_j \le \sum_{i=1}^{6} x_{ij} \cdot K_A(1-\alpha) + \sum_{i=7}^{16} x_{ij} \cdot K_B(1-\beta), j = 1,2,3,4$$

初次之外，第三问的模型和第二问的模型类似，于是我们可以总结得到第三问的双目标优化模型[8]。

目标函数：

$$\min \min \left\{ \begin{array}{l} \max\{(2x_{ij}-1)\cdot t_{Aj} + 2x_{ij}(q_1+q_2), i = 1,2,\cdots 6 \\ (2x_{ij}-1)\cdot t_{Bj} + 2x_{ij}(p_1+p_2), i = 7,8,\cdots 16\}, j = 1,2,3,4 \end{array} \right\}$$

$$\min \sum_{j=1}^{4} \left(\sum_{i=1}^{6} x_{ij} \cdot K_A(1-\alpha) + \sum_{i=7}^{16} x_{ij} \cdot K_B(1-\beta) - M_j \right)$$

约束条件：

$$\left\{ \begin{array}{l} \sum_{i=1}^{6} x_{ij} \cdot K_A(1-\alpha) + \sum_{i=7}^{16} x_{ij} \cdot K_B(1-\beta) \le M_j \cdot 110\%, j = 1,2,3,4 \\ M_j \le \sum_{i=1}^{6} x_{ij} \cdot K_A(1-\alpha) + \sum_{i=7}^{16} x_{ij} \cdot K_B(1-\beta), j = 1,2,3,4 \\ (2x_{ij}-1)\cdot t_{Aj} + 2x_{ij}(q_1+q_2) < T_j, i = 1,2,\cdots 6, j = 1,2,3,4 \\ (2x_{ij}-1)\cdot t_{Bj} + 2x_{ij}(p_1+p_2) < T_j, i = 7,8,\cdots 16, j = 1,2,3,4 \\ x_{ij} \ge 0 \\ \alpha \in [0.05, 0.10], \beta \in [0.03, 0.07] \end{array} \right.$$

5.3.2 模型的求解

在 5.2.2 中我们详细介绍了求解模型二的贪婪算法，但是在模型二中并没有损耗率限制，而且在模型二中，对超出物资的最大限制为 150%。与之不同的是，在模型三中，对超出物资的最大限制减小为 110%，并且增加了损耗因素。但是求解模型的方法思想并没有错，设置损耗因子的值分别为 0.075 和 0.05。使用 5.2.2 中的贪婪方法求解模型三得到如下的结果：

调度情况	A102	A122	B121	B404
飞机 A	0	7	8	0
飞机 B	5	0	0	5
多出的物资	15	9.175	6.20	10
最大限度	8	7.5	9	8.5

上表展示了 5.2.2 中贪婪算法求解模型三的结果，从数据中我们发现，部队 A102、部队 A122、部队 B404 均存在超额运输，很明显，如果采用 5.2.2 的贪婪算法求解模型是肯定不能满足约束条件的，所以我们需要改进该算法。

一般的，在 5.2.2 的贪婪算法是，先优先调度全部的飞机 B 参与运输任务，之所以这么做是因为飞机 B 的飞行速度非常慢，相比于飞机 A 要慢 5 倍，而且飞

机 B 的装卸时长均要比飞机的装卸时长要大，而飞机 B 的承载能力只比飞机 A 的承载能力大的程度并不多，所以从最短时间上来考虑，那么飞机 A 的运输效率肯定比飞机 B 要大。而在问题三中，题目对两种飞机的运输货物进行了限制，即两种飞机在运输过程中都存在一定的损失，而且对最大限度进行了约束，那么再用贪婪算法来求解就会带来很大麻烦。值得一提的是，在本问中使用模拟退火来求解优化模型又有很大的优势。下面是模拟退火[9]的详细步骤：

（1） 设置初始温度及参数，将飞机 A 和飞机 B 的调度方案作为解，这里的初始化解必须满足解空间条件，设置退火系数为 0.90。

（2）开始进入循环

1） 产生新的解：

在产生新解的步骤中，我们采用了随机扰动的方式来生成新解，再使用建立的数学模型求得此解下的两个目标函数值，并记录下来。

2） 判断是否达到新解的迭代次数，达到则将扰动得到的最优解记为新解

3） 使用 Metropolis 准则，根据扰动产生的新解 X_{new} 的能量值 $E(X_{new})$ 与扰动前的解 X_{old} 的能量值 $E(X_{old})$ 和当前温度 t 来判断是否接受新的解，如若接受新的解，则用新的解代替旧解，其中关于接受概率的 Metropolis 准则为

$$p=\begin{cases}1 & E(X_{new})\le E(X_{old})\\ e^{-\frac{E(X_{new})-E(X_{old})}{T}} & E(X_{new})>E(X_{old})\end{cases}$$

（3）判断温度是否达到温度阈值，若未达到则继续进行外循环

对于该模拟退火算法需要进一步叙述，由于模拟退火本质上也是一个概率算法，并且要想实现该算法最重要的两个步骤就是新解的产生和适应度函数的计算。由于本问有两个目标函数，但是这两个目标函数之间存在很大关系，那就是在飞机 A 不影响飞机 B 的决定性情况下，最小化运输时间该目标函数的值不会发生很大的变化。因此我们可以将最小化运输时间转化为约束条件，而将多目标规划转化为单目标规划，这样就可以用模拟退火来计算适应度值。

至于新解的产生，必须要在题目要求的解空间中生成新解，也即考虑最大时间、完成任务量，飞机数量等等因素。

下面是模拟退火的流程图：

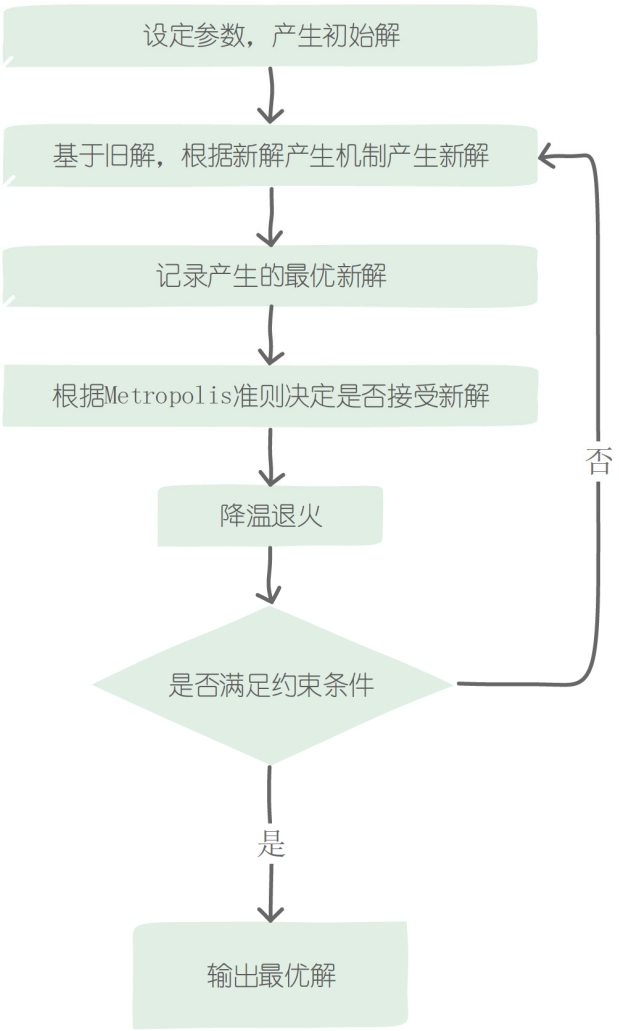


图 7 模拟退火算法流程图

5.3.3 模型求解的结果

借助 Matlab 编程求解得到模型三的最优解为：

调度情况	A102	A122	B121	B404
飞机 A	2	0	6	4
飞机 B	3	4	1	2
多出的物资	1.05	1	1.15	1.1
最大限度	8	7.5	9	8.5

其中最优的目标函数值是 334.3 吨，下面是模拟退火算法的迭代图：

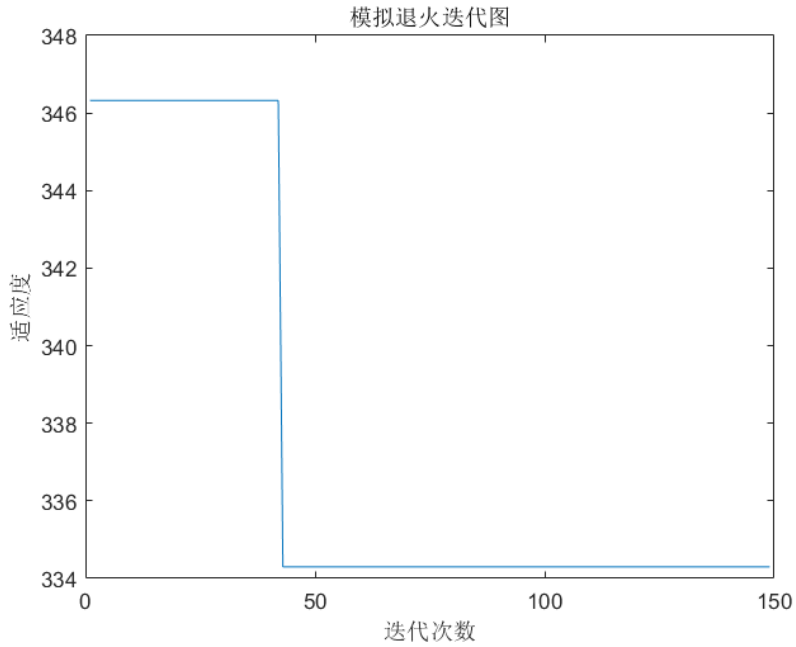


图 8 模拟退火算法迭代图

调度飞机的平面示意图：

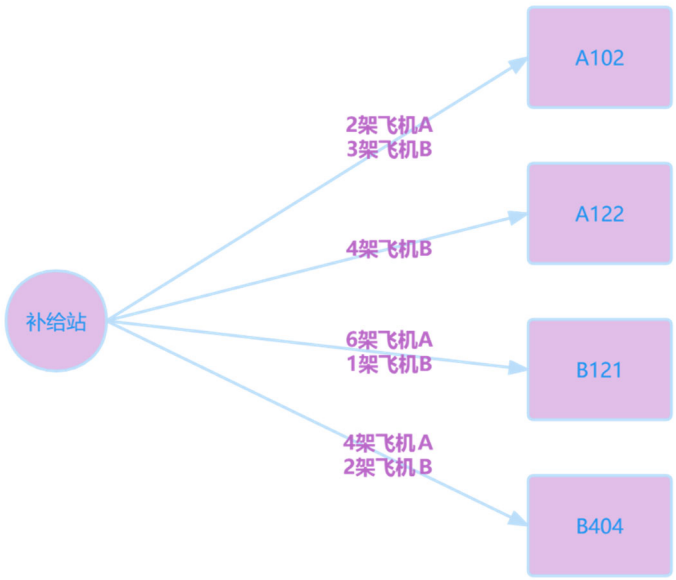


图 9 问题三的飞机调度图

5.3.4 模型结果的分析

分析模型的结果我们发现，四个部队的多出的物资分别为 1.05、1、1.15、1.1，相较于原贪心算法求解的结果：15、9.175、6.20、10 提升了很多。原贪心算法求解的最优适应度函数值为：370.375 吨；模拟退火求解的最优适应度函数值为 334.3 吨；提升了 10.79%。并且最短时间任然可以达到 102.01min。

5.3.5 模型灵敏度分析

由于损失因素存在不确定性，所以我们需要检测模型三对参数损失因子的灵敏度情况。损失因子参数分别为α和β，它们的取值范围分别是 5%-10%和 3%-7%。

（一） 令α和β均为最小值：5%和 3%，使用模拟退火算法运行得到的结果如下：

调度情况	A102	A122	B121	B404
飞机 A	2	3	6	1
飞机 B	3	2	1	4
多出的物资	2.90	0.85	5.25	3.2
最大限度	8	7.5	9	8.5

其中，最优适应度为 342.2 吨，时间为 102.01 分钟。

（二）令α和β均为最大值：10%和 7%，使用模拟退火算法运行得到的结果如下：

调度情况	A102	A122	B121	B404
飞机 A	4	5	3	1
飞机 B	2	1	3	4
多出的物资	6.1	6.9	0.9	5.9
最大限度	8	7.5	9	8.5

其中，最有适应度为 338.1 吨，时间为 149.93 分钟。

从 102.1 分钟变为 149.93 分钟，其中最主要的原因就是损失因素带来的影响，后者的损失因素增加到最大，可以发现飞机A的派遣数量超过了 12 次，达到了 13 次，所以飞机A的往返次数就要增多，这样的话，总的时间就会增大，但是都是满足题目要求的时间限制的。

6.1 模型的优点

（1）问题一采用两种不同的算法求解模型，不仅能对比出两种算法在处理相同问题时的效率，更能使结果相互对比，验证模型的合理性；

（2）问题二以最小化完成所有运输目标所用时间出发建立单目标规划模型，这样考虑比较符合现实情况，并且采用的贪婪算法从效率上要比智能优化算法要更好；

6.2 模型的缺点

（1）问题二和问题三在考虑最优调度方案上，只考虑到最小化运输时间，并没有考虑到其它的一些目标，这样考虑并不周全；

参考文献

[1]刘颖超,张纪元.梯度下降法[J].南京理工大学学报(自然科学版),1993(02):12-16+22.

[2]于航,王子谦,雷振宇,高尚策.面向特征选择问题的差分鲸鱼优化算法[J].电子设计工程,2021,29(21):12-17+22.

[3]褚鼎立,陈红,王旭光.基于自适应权重和模拟退火的鲸鱼优化算法[J].电子学报,2019,47(05):992-999.

[4]曹策俊,李从东.基于数学规划的应急组织指派优化问题综述[J].系统仿真学报,2021,33(01):1-12.

[5]董海,王瀚鹏.基于种群迭代贪婪算法无等待流水车间调度[J/OL].控制工程:1-12[2022-03-20].

[6]刘卿君. 基于贪婪算法的最小（2,m）-连通控制集问题研究[D].河北师范大

学,2020.
[7]《运筹学》教材编写组. 运筹学（修订版）. 清华大学出版社，1990.
[8]肖晓伟,肖迪,林锦国,肖玉峰.多目标优化问题的研究概述[J].计算机应用研究,2011,28(03):805-808+827.
[9]陈华根,吴健生,王家林,陈冰.模拟退火算法机理研究[J].同济大学学报(自然科学版),2004(06):802-805.

附录

环境	Windows 10；CPU：Intel i7；GPU：NVIDIA GEFORCE 1650
语言	Matlab2021a
文件列表：	
Code:	Question1.m Question2.m Question3.m gradient_descent.m find_X.m draw.m fit.m SA.m
Figure	部队分布图.png 部队和补给站分布图.png 部队和补给站分布图(带权).png 第一问结果.png 飞机调度示意图.png 飞机调度示意图 2.png 流程图.png 鲸鱼算法.png 模拟退火.png 梯度下降法.png 总思路图.png

代码在下一页

```
clc
clear

% 绘图函数
% draw

% 求最优位置

% 鲸鱼算法
% 参数初始化
a = 2; % 初始游动因子的值，后续会不断缩减
MAX_iter = 200; % 迭代次数
iter = 1; % 初始化迭代次数
b = 3; % 螺旋系数
number = 100; % 定义鲸鱼的数量
fitt = zeros(number,1); % 初始化适应度
x = zeros(number,2); % 初始化种群
l_min = -100; %自变量下界
l_max = 100; % 自变量上界
for i = 1:number % 初始化种群和适应度
    x(i,1) = l_min + rand(1) * (l_max - l_min);
    x(i,2) = l_min + rand(1) * (l_max - l_min);
    fitt(i) = fit(x(i,:));
end
fit_best = min(fitt); % 记录最优适应度
ind = find(fitt == min(fitt),1);
X_best = x(ind,:); % 记录最优位置
X_BEST = []; % 记录全局
FITT = []; % 记录全局
%% 迭代开始
while iter <= MAX_iter
    for i = 1:number
        if rand() < 0.5 % 采用游动捕食
            a = 2 * (1 - iter / MAX_iter); % 更新游动因子
            r = rand(1,2); % 随机数
            C = 2 * r; % 计算摆动因子
            A =  2 * r * a - a; % 计算收敛因子
            if sqrt(A(1)^2 + A(2)^2) < 1 % 采取包围捕食
                D = abs(C .* X_best - x(i,:)); % 计算最优距离
                x(i,:) = X_best - A .* D; % 这里是位置更新
                if x(i,1) < l_min || x(i,1) > l_max % 判断是否超出边界
```

```
                    x(i,1) = l_min + rand(1) * (l_max - l_min);

                end
                if x(i,2) < l_min || x(i,2) > l_max
                    x(i,2) = l_min + rand(1) * (l_max - l_min);

                end
            else % 采取随机游动
                rand_ind = unidrnd(20); % 随机抽取鲸鱼
                D = abs(C .* x(rand_ind,:) - x(i,:));
                x(i,:) = x(rand_ind,:) - A .* D;
                if x(i,1) < l_min || x(i,1) > l_max % 判断是否超出边界
                    x(i,1) = l_min + rand(1) * (l_max - l_min);

                end
                if x(i,2) < l_min || x(i,2) > l_max
                    x(i,2) = l_min + rand(1) * (l_max - l_min);

                end
            end
        else % 采用气泡捕食
            D_1 = abs(X_best - x(i,:));
            l = unifrnd(-1,1); % 随机数
            x(i,:) = D_1 * exp(b * l) * cos(2 * pi * l) + X_best;
            if x(i,1) < l_min || x(i,1) > l_max % 判断是否超出边界
                x(i,1) = l_min + rand(1) * (l_max - l_min);
            end
            if x(i,2) < l_min || x(i,2) > l_max
                x(i,2) = l_min + rand(1) * (l_max - l_min);
            end
        end
        fitt(i) = fit(x(i,:)); % 每更新一次位置，求解一次适应度
    end
    disp(['当前迭代次数为：',num2str(iter)])
    % 迭代完一次更新数据
    fit_best = min(fitt); % 更新当前代数中种群的最优解
    ind = find(fitt == min(fitt),1);
    X_best = x(ind,:); % 更新最优位置
    X_BEST = [X_BEST;X_best];
```

```
77.     FITT = [FITT;fit_best];
78.     iter = iter + 1;
79.end
80.Fit_Best = min(FITT);
81.ind_best = find(FITT == min(FITT),1);
82.X_BE = X_BEST(ind_best,:);
83.disp(['找到的最优解为：',num2str(Fit_Best)])
84.disp(['最优解对应的 X：',num2str(X_BE)])
85.figure(2)
86.plot(1:MAX_iter,FITT)
87.title('种群迭代图')
88.xlabel('迭代次数')
89.ylabel('适应度值')
1.function [] = draw()
2.% 第一题绘图
3.D_X = [5,9,13,16];
4.D_Y = [4,5,6,4];
5.figure(1)
6.scatter(D_X,D_Y,'^r')
7.grid on
8.axis([2,18,2,8])
9.ylabel('y')
10.xlabel('x')
11.title('部队分布图')
12.% 加图例
13.text(4.42,3.7,'A102')
14.text(8.5,4.7,'A122')
15.text(12.5,5.7,'A121')
16.text(15.5,3.7,'B404')
17.legend('部队')
18.end
1.function [fitness] = fit(x)
2.% 计算适应度
3.D_X = [5,9,13,16];
4.D_Y = [4,5,6,4];
5.fitness = 0;
6.for i=1:length(D_X)
7.     fitness = fitness + sqrt( (D_X(i)-x(1))^2 + (D_Y(i)-
   x(2))^2 );
8.end
9.
1.clc;
```

```
2.clear;
3.% 梯度下降法求最小值
4.alpha = 0.001; % 步长
5.x_1 = 8; % 初始化 x
6.x_2 = 8; % 初始化 y
7.epsilon = 0.00001; % 迭代终止精度
8.tic
9.D_X = [5,9,13,16];
10.D_Y = [4,5,6,4];
11.E = [];
12.while 1
13.    last_x = x_1; % 记录上一次 x
14.    last_y = x_2; % 记录上一次 y
15.
16.    x_1 = x_1 + alpha * 2 * sum(D_X-x_1); % 梯度下降法求解 x_1
17.    x_2 = x_2 + alpha * 2 * sum(D_Y-x_2);% 梯度下降法求解 x_2
18.    if (abs(fit([x_1,x_2])-fit([last_x,last_y])) < epsilon) % 判
   断
19.        break;
20.    end
21.    E = [E;fit([x_1,x_2])];
22.end
23.z = abs(fit([x_1,x_2]));
24.disp(['最小值为: ',num2str(z)])
25.disp([x_1,x_2])
26.
27.plot(E)
28.title('梯度下降法目标函数迭代图')
29.xlabel('迭代次数')
30.ylabel('目标函数值')
31.toc
1.clc
2.clear
3.
4.% 部队地点
5.D_X = [5,9,13,16];
6.D_Y = [4,5,6,4];
7.
8.% 计算补给站和每个部队之间的距离
9.XY = [10,30];
10.D = zeros(4,1);
11.for i = 1:length(D_X)
```

```
    D(i) = sqrt((D_X(i) - XY(1))^2 + (D_Y(i) - XY(2))^2);
end
disp('补给站到每个部队之间的距离为：')
disp(num2str(D))

% 第二题绘图
figure(2)
scatter(D_X,D_Y,'^r')
hold on
scatter(XY(1),XY(2),'o')
grid on
axis([2,18,0,40])
ylabel('y')
xlabel('x')
title('部队和补给站分布图')
% 加图例
text(4.42,2.7,'A102')
text(8.5,3.7,'A122')
text(12.5,4.7,'A121')
text(15.5,2.7,'B404')
% 绘制连线
for i = 1:length(D_X)
    line([XY(1),D_X(i)],[XY(2),D_Y(i)])
end
% 绘制连线数值
text(7,18,'26.48km')
text(8.9,15,'25.02km')
text(11,16.3,'24.19km')
text(14.2,10,'26.68km')
legend('部队','补给站')

% 贪心算法
% 定义飞机的属性
Plane_A.load = 13;
Plane_A.speed = 260;
Plane_A.number = 6;
Plane_A.up = 20;
Plane_A.down = 20;
Plane_B.load = 20;
Plane_B.speed = 50;
Plane_B.number = 10;
Plane_B.up = 30;
```

```
54.Plane_B.down = 40;
55.depot_stations_D = D; % 补给站与部队之间的距离
56.troop_M = [80,75,90,85]; % 部队的最大承重量
57.troop_M_ed = zeros(4,1);
58.troop_T = [450,430,410,400]; % 最晚需求时间
59.time = 0;
60.X = zeros(2,4); % 解空间
61.A_number = 0; % 定义额外需要的飞机 A
62.while 1
63.    % 先考虑飞机 B
64.    if Plane_B.number > 0
65.        [~,choose_index] = max(depot_stations_D); % 获得当前派遣
   的飞机目标
66.        if troop_M(choose_index) > 0
67.            X(2,choose_index) = X(2,choose_index) + 1;
68.            Plane_B.number = Plane_B.number - 1;
69.            troop_M(choose_index) = troop_M(choose_index) - Plan
   e_B.load;
70.            if troop_M(choose_index) <= 0
71.                troop_M_ed(choose_index) = troop_M(choose_index)
   ;
72.                troop_M(choose_index) = 0;
73.                depot_stations_D(choose_index) = 0;
74.            end
75.        end
76.    else % 考虑完飞机 B，考虑飞机 A
77.        if Plane_A.number > 0
78.            [~,choose_index] = max(depot_stations_D); % 获得当前
   派遣的飞机目标
79.            if troop_M(choose_index) > 0
80.                X(1,choose_index) = X(1,choose_index) + 1;
81.                Plane_A.number = Plane_A.number - 1;
82.                troop_M(choose_index) = troop_M(choose_index) -
   Plane_A.load;
83.                if troop_M(choose_index) <= 0
84.                    troop_M_ed(choose_index) = troop_M(choose_in
   dex);
85.                    troop_M(choose_index) = 0;
86.                    depot_stations_D(choose_index) = 0;
87.                end
88.            end
89.        end
```

```
    end

    % 如果第一批飞机派遣完成，接着派遣第二批
    if Plane_A.number == 0 && Plane_B.number == 0
        [~,choose_index] = max(depot_stations_D); % 获得当前派遣的飞机目标
        if troop_M(choose_index) > 0
            X(1,choose_index) = X(1,choose_index) + 1;
            A_number = A_number + 1;
            troop_M(choose_index) = troop_M(choose_index) - Plane_A.load;
            if troop_M(choose_index)<=0
                    troop_M_ed(choose_index) = troop_M(choose_index);
                    troop_M(choose_index) = 0;
                    break
                end
            end
        end
    end
    % 依靠结果计算时间
    Plane_A.Time = D/Plane_A.speed*60; % A 飞机从补给站到部队的时间
    Plane_B.Time = D/Plane_B.speed*60; % A 飞机从补给站到部队的时间
    Time_sum = Plane_A.Time(3)*3 + 2 * (Plane_A.up+Plane_A.down);
    disp(['找到的最快时间为：',num2str(Time_sum)])
%% 模拟退火算法实现
clc;clear


% 部队地点
D_X = [5,9,13,16];
D_Y = [4,5,6,4];

% 计算补给站和每个部队之间的距离
XY = [10,30];
D = zeros(4,1);
for i = 1:length(D_X)
    D(i) = sqrt((D_X(i) - XY(1))^2 + (D_Y(i) - XY(2))^2);
end

% 定义飞机的属性
```

```
Plane_A.load = 13;
Plane_A.speed = 260;
Plane_A.number = 6;
Plane_A.up = 20;
Plane_A.down = 20;
Plane_A.loss = 0.1;
Plane_B.load = 20;
Plane_B.speed = 50;
Plane_B.number = 10;
Plane_B.up = 30;
Plane_B.down = 40;
Plane_B.loss = 0.07;
depot_stations_D = D; % 补给站与部队之间的距离
troop_M = [80,75,90,85]; % 部队的最大承重量
troop_M_max = troop_M * 1.1;
troop_M_ed = zeros(4,1);
troop_T = [450,430,410,400]; % 最晚需求时间
Plane_B.load = Plane_B.load * (1 - Plane_B.loss);
Plane_A.load = Plane_A.load * (1 - Plane_A.loss);
X = zeros(2,4);
% 计算两个飞机到部队的时间
Plane_A.Time = D/Plane_A.speed*60; % A 飞机从补给站到部队的时间
Plane_B.Time = D/Plane_B.speed*60; % A 飞机从补给站到部队的时间

%% 参数初始化
T = 100;    % 初始温度
T_min = 5; % 温度阈值
Markov  = 1;  % 每个温度下的迭代次数
alfa = 0.98;  % 温度衰减系数
[x,Load_results] = find_X(Plane_A,Plane_B,troop_M_max,troop_M);
fitt = sum(Load_results); % 计算其适应度
fitt_min = fitt; % 将初始适应度设为最小
x_min = x; % 将初始位置记为最小位置
FITT = []; % 记录中间变量
X = []; % 记录中间变量
%% 迭代开始
tic % 计时函数
while T > T_min
    for i = 1:Markov
        [x_new,Load_results] = find_X(Plane_A,Plane_B,troop_M_max,troop_M);
        fitt_new = sum(Load_results); % 计算新解的适应度
```

```
58.         if fitt_new < fitt % 因为是找最小值，如果新解小于最小解，则更新解
59.             fitt = fitt_new;
60.             x = x_new;
61.         else % 如果不小于，则随机选择
62.             p = exp(-(fitt_new - fitt)/T); % 接受概率，算法核心部分
63.             if rand < p % 如果随机数小于随机概率，则接受新解
64.                 fitt = fitt_new;
65.                 x = x_new;
66.             end
67.         end
68.         if fitt < fitt_min % 检查是否小于历史最优
69.             fitt_min = fitt;
70.             x_min = x_new;
71.         end
72.%          disp(['当前代数为：',num2str(i)])
73.     end
74.     FITT = [FITT fitt_min];
75.     X_min = [x_min(1,:),x_min(2,:)];
76.     X = [X;X_min];
77.     T = T * alfa; % 温度下降
78.     disp(['当前温度为：',num2str(T)])
79.end
80.toc
81.%% 结果可视化
82.[fit_best,ind_best] = min(FITT);
83.x_best = X(ind_best,:);
84.disp(['模拟退火算法求解的最优解为：',num2str(fit_best)])
85.disp(['模拟退火算法求解的最优位置为：',num2str(x_best)])
86.
87.figure(2)
88.plot(FITT);
89.xlabel('迭代次数');
90.ylabel('适应度');
91.title('模拟退火迭代图')
1.function [X,load_sum] = find_X(Plane_A,Plane_B,troop_M_max,troop_M)
2.%FIND_X 生成新解
3.while 1
4.     temp1 = randsrc(1,4,0:8);
5.     while 1
```

```
    temp2 = randsrc(1,4,0:4);
    if sum(temp2) == 10
        break
    end
     end
     X(1,:) = temp1;
     X(2,:) = temp2;
     AB_load = [Plane_A.load,Plane_B.load];
     load_sum = AB_load*X;
     if (sum(load_sum <= troop_M_max) == 4) && (sum(load_sum > tr
   oop_M) == 4)
         break
     end
end
end
```