总的来说，算法是一个遗传算法套了局部搜索算法。

初始化：随机生成初始种群，每个染色体是一个任务点到车辆的分配方案。我们会为工作线程分配初始化任务来加快初始化。

1.遗传算法核心

- **选择**：轮盘赌选择更优的父代。
- **交叉**：通过单点交叉生成新的子代。
- **变异**：随机修改染色体，增加多样性。

每代种群数量 n，总共迭代m代。

```
//init
population := ∅
for i : 1 to populationSize
    population := population ∪ {generateInitialSolution}
    //generateInitialSolution 是纯随机生成

//遗传算法
for 1 to generations // 枚举世代
    for i : 1 to populationSize // 枚举种群
        localsearch() //通过局部搜索来修复不可行种群并优化种群
    for i : 1 to populationSize / 2 //保留一半结果最优的解作为下一世代
        population := population / {theWorstSolution}
    heredityAndVariation()
    //遗传：将剩下的两两子代进行单点交叉从而生成两个新个体，从而再次将种群数量填补回来
    //变异：将所有种群内的解以极低的概率进行随机染色体变异，从而避免陷入局部最优
```

2.局部搜索

首先对于每辆车因为其任务点确定，直接跑优化算法进行规划路线。

目前的想法：

①

我们先对所有的无限容量油量车进行路径规划，然后在对无人机规划（因为无人机可以依附在车上进行车机协同，所以先确定车辆路径）

对于无限容量油量车我们直接可以使用现成的TSP求解器或是建模成ILP算法，直接用商用求解器。

对于无人机，目前还没想到用什么算法会好一点，目前我是直接使用最邻近法进行规划路径，同时会在规划途中去枚举所有的无限容量油量车辆当前路径，看看是否可以通过车机协同实现优化。

②

全部车辆路径规划完成后，进行一些局部搜索的优化，我们会每次寻找代价最高的车辆，枚举删除哪个任务点后会使得代价变得最小。将这个任务点移除后，枚举其他车辆并添加到最优车辆上。

对于无人机，由于删除某任务点会导致时间发生变化，后续的一个车机协同的任务点，车辆的运行轨迹和之前会不同就无法使用，所以目前没想好怎么更好的优化无人机，目前的想法只能是删除每个点后重新将整个无人机路径重新规划一遍。

```
for Si in unlimitedCapacityFuelVehicle
    Si := TSP(Si) // 使用TSP求解器规划路径
while not reach the designatedTime
    Si := the worst unlimitedCapacityFuelVehicle//寻找代价最大的车辆
    taski := the worst task in Car Si //寻找最差的任务点
    min := ∞
    for Sj in unlimitedCapacityFuelVehicle
        if min > value(Sj ∪ {taski}) - value(Sj)
            min := value(Sj ∪ {taski}) - value(Sj)
            bestCar = Sj
    bestCar := bestCar ∪ {taski}
    Si = Si / {taski}
for Si in UAV
    Si := nearestNeighborFunctuion(Si)
while not reach the designatedTime
    Si := the worst UAV//寻找代价最大的无人机
    taski := the worst task in UAV Si //寻找最差的任务点
    min := ∞
    for Sj in UAV
        if min > value(Sj ∪ {taski}) - value(Sj)
            min := value(Sj ∪ {taski}) - value(Sj)
            bestUAV = Sj
    bestUAV := bestUAV ∪ {taski}
    Si = Si / {taski}
```

3.插入新的任务点后

由于一个人任务点失败相当于加入新的任务点，所以任务点失败也归于此类

①

我们先将新加入的任务点枚举的直接插入到某一个交通工具的任务序列中。这样就可以生成车数量个新解

之后我们将所有的未完成任务点全部使用原先的初始化方案生成另外的解来扩充到populationSize

之后进行正常的遗传和变异算法。

```
//init
population := ∅
for i : 1 to populationSize / 2
    population := population ∪ {generateInitialSolution}
    //generateInitialSolution 是纯随机生成
for i : 1 to populationSize / 2
    population := population ∪ {InsertIntoOriginalPath}
//遗传算法
for 1 to generations // 枚举世代
    for i : 1 to populationSize // 枚举种群
        localsearch() //通过局部搜索来修复不可行种群并优化种群
    for i : 1 to populationSize / 2 //保留一半结果最优的解作为下一世代
        population := population / {theWorstSolution}
    heredityAndVariation()
    //遗传：将剩下的两两子代进行单点交叉从而生成两个新个体，从而再次将种群数量填补回来
    //变异：将所有种群内的解以极低的概率进行随机染色体变异，从而避免陷入局部最优
```

②

当进行局部搜索阶段也会进行调整，因为不是所有的解都是可行解了（必须保证原先的任务点的完成时间要小于之前计算的总时间）所以需要调整。

对于每个交通工具我们首先将所有具有时间限制的任务点计算完毕，如果规划完后解不可行，枚举并删除影响时间最大的点直到解可行(并将删除的点记录下来)。规划完所有的解后，将不可行的解，枚举的插入到每个车辆中，如果仍然不可行则放弃该解将其设定为不可行。之后在插入没有时间限制的新点，在保证解仍然可行的前提下直接枚举的插入到行动序列中（因为最差情况是送完之前的解在送新点，所以总有办法能保证仍为可行解）。

```
//localsearch
deletedTasks =  ∅
for Si in unlimitedCapacityFuelVehicle
    Si := TSP({taski | taski is time limit tasks in Si})
    // 使用TSP求解器规划路径将所有具有时间限制的车辆规划给Si
    while Si is not a feasible solution
        taskj := the worst task in Si
        Si := Si / {taskj}
        deletedTasks = deleted_Tasks ∪ {taskj}
for taski in deletedTasks
   for Sj in unlimitedCapacityFuelVehicle
        if Sj ∪ {taski} is feasible
            Sj := Sj ∪ {taski}
            break
    if No vehicles can be inserted // 如果没有车辆可以被插入
        value[solution] += ∞
        return false
for Si in unlimitedCapacityFuelVehicle
    Si := insert({taski | taski is no time limit tasks in Si})


while not reach the designatedTime
    Si := the worst unlimitedCapacityFuelVehicle//寻找代价最大的车辆
    taski := the worst task in Car Si //寻找最差的任务点
    min := ∞
    bestCar := -1
    for Sj in unlimitedCapacityFuelVehicle
        if min > value(Sj ∪ {taski}) - value(Sj)&Sj ∪ {taski} is feasible
            min := value(Sj ∪ {taski}) - value(Sj)
            bestCar = Sj
    if bestCar != -1
        bestCar := bestCar ∪ {taski}
        Si = Si / {taski}

for Si in UAV
    Si := TSP({taski | taski is time limit tasks in Si})
    // 使用TSP求解器规划路径将所有具有时间限制的车辆规划给Si
    while Si is not a feasible solution
        taskj := the worst task in Si
        Si := Si / {taskj}
        deletedTasks = deleted_Tasks ∪ {taskj}
for taski in deletedTasks
   for Sj in unlimitedCapacityFuelVehicle
        if Sj ∪ {taski} is feasible
            Sj := Sj ∪ {taski}
            break
    if No vehicles can be inserted // 如果没有车辆可以被插入
        return false
for Si in UAV
    Si := insert({taski | taski is no time limit tasks in Si})
```

```
while not reach the designatedTime
    Si := the worst UAV//寻找代价最大的车辆
    taski := the worst task in UAV Si //寻找最差的任务点
    min := ∞
    bestUAV := -1
    for Sj in unlimitedCapacityFuelVehicle
        if min > value(Sj ∪ {taski}) - value(Sj)&Sj ∪ {taski} is feasible
            min := value(Sj ∪ {taski}) - value(Sj)
            bestCar = Sj
    if bestCar != -1
        bestUAV := bestUAV ∪ {taski}
        Si = Si / {taski}
```

后续需要改进的点

① 无人机的路径规划可以采取更好的算法。

② 需要设计一个更好的局部搜索的框架，目前搜索的方式过于暴力没有考虑陷入局部最优或是陷入循环的解决方案，可以增加格局检测策略等局部搜索优化方案

③ 无人机和车辆必须分开，需要设计一个不需要分开计算的方法，但目前完全没想法

④ 在插入新任务时，有没有什么更可行的路径规划方案避免出现更多的不可行解。

⑤ 再插入新任务时目前没法很好的考虑到突发事件（堵车，某条路不同时间段的速率不同）

⑥ 局部搜索和遗传之间的比例设置