第3章

面向单颗敏捷卫星任务调度问题的数据驱动优化算法

由于敏捷卫星任务调度问题的复杂性，即使不考虑多星协同以及成像时可能出现的突发情况，单星调度仍然具有 NP-hard 特点。高效求解单星调度问题，是能够良好解决多星常规任务调度问题和多星应急任务调度问题的关键。因此，本章首先开展面向单星调度问题的数据驱动优化算法研究。

本章围绕卫星任务调度的下层单星调度问题开展研究。首先，对单星调度问题进行抽象化描述，给出问题的相关定义及基本假设，详细分析问题的约束条件，在此基础上建立单星调度问题的数学模型。其次，针对元启发式算法在求解任务调度问题时求解速度慢以及求解效果不稳定等问题，提出一种数据挖掘和元启发式算法结合的求解思路。再次，在该思路指导下，分别在局部搜索算法和进化算法框架中加入频繁模式挖掘，设计 FPAPSA 和 FPAMA。最后，设计多个不同任务规模的实验场景，从求解质量、求解时间以及稳定性方面去客观验证所提出算法的性能，进一步分析算法核心组件对算法整体性能的影响。

3.1 单星调度问题建模

3.1.1 问题假设

在遵循实际卫星任务管控的业务处理流程前提下，为方便开展研究，本章做出以下 5 点假设来简化单星调度问题。

(1) 本书后续所指的任务是卫星过境一次即可执行的元任务。对于大区域目标覆盖、移动目标跟踪以及多频次观测目标等需要卫星多次过境才能完成的复杂成像需求，可以参考 Bunkheila 等[176]与任必虎和贺仁杰[177]介绍的条带切分方法转换成多个元任务去执行。本书在此不研究具体的任务分解方法。

(2) 任务在执行过程中不能被抢占。一旦任务开始执行，不允许中途终止。

(3) 每颗卫星只搭载一个成像载荷，这意味着卫星同时只能执行一个任务。

(4) 每个任务至多执行一次，不可重复执行。

(5) 不考虑卫星特殊使用约束以及云层覆盖等特殊情况。

后续第 4 章、第 5 章也将继续沿用这些基本假设。

3.1.2　数学模型

第 2 章已经对卫星任务管控的业务流程进行了详细描述，接下来开始对单星调度问题的目标函数、约束条件以及决策变量进行数学化表示，并建立问题的数学模型。

1. 目标函数

本章将已调度任务的总收益最大化作为单星调度问题的目标函数。任务 t_i 的收益由优先级 p_i 确定，目标函数表示如下：

$$\max \sum_{i=1}^{N} \sum_{k=1}^{|w_i|} p_i x_{ik} \tag{3.1}$$

式中，N 为任务数量；$|w_i|$ 为任务 t_i 在卫星上的可见时间窗口数量；p_i 为任务 t_i 的收益值。

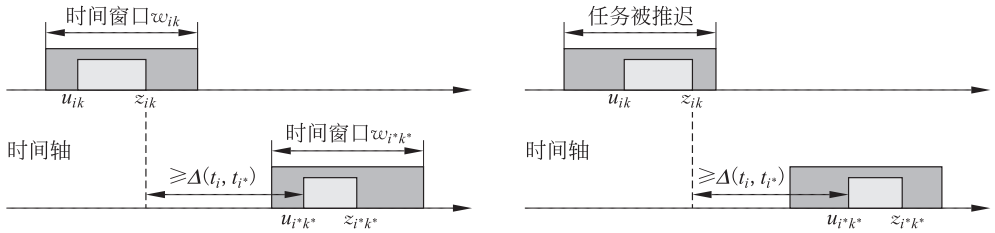
二元变量 x_{ik} 为单星调度问题的决策变量，其取值如下：

$$x_{ik}=\begin{cases}1, & \text{任务 } t_i \text{ 被安排在时间窗口 } w_{ik} \text{ 内执行},\\ 0, & \text{否则},\end{cases} \quad \forall t_i \in T \tag{3.2}$$

式中，T 为待调度的任务集合；w_{ik} 为任务 t_i 在卫星上的第 k 个可见时间窗口；w_i 为任务 t_i 在卫星上的可见时间窗口集合。

2. 约束条件

(1) 转换时间约束。姿态转换时间约束是单星调度问题的核心约束之一，目的是确保卫星有足够多的姿态机动时间去执行后续任务，从而反映敏捷卫星具有时间依赖特性。时间依赖是指某个变量随着时间变量的变化而变化。如图 3.1 所示，假设任务 t_i 与任务 t_{i^*} 是同一颗卫星执行的两个相邻任务，并且 $u_{ik}<u_{i^*k^*}$，那么任务 t_i 的结束时间 z_{ik} 与任务 t_{i^*} 的开始时间 $u_{i^*k^*}$ 的间隔不小于转换时间 $\Delta(t_i,t_{i^*})$。



(a) 示意1　　(b) 示意2

图 3.1　转换时间约束示意图

该约束表示如下：

$$u_{ik}+l_i+\Delta(t_i,t_{i^*})\leqslant u_{i^*k^*},\quad \forall t_i,\quad t_{i^*}\in T,$$
$$\text{当 } x_{ik}=x_{i^*k^*}=1 \text{ 和 } u_{ik}<u_{i^*k^*} \text{ 时} \tag{3.3}$$

式中，u_{ik} 为任务 t_i 在可见时间窗口 w_{ik} 的开始时间；l_i 为任务 t_i 的执行时间；$\Delta(t_i,t_{i^*})$为任务 t_i 与任务 t_{i^*} 的转换时间。

结合 Wu 等[8]以及实际工程对转换时间的计算方法，将转换时间描述成与合成转角相关的分段函数，表示如下：

$$\Delta(t_i,t_{i^*})=\begin{cases}11.66, & \Delta g\leqslant 10\\ 5+\dfrac{\Delta g}{a_1}, & 10<\Delta g\leqslant 30\\ 10+\dfrac{\Delta g}{a_2}, & 30<\Delta g\leqslant 60\\ 16+\dfrac{\Delta g}{a_3}, & 60<\Delta g\leqslant 90\\ 22+\dfrac{\Delta g}{a_4}, & \Delta g>90\end{cases} \tag{3.4}$$

式中，a_1、a_2、a_3、a_4 为不同姿态的转换速度；Δg 为任务 t_i 与任务 t_{i^*} 之间的合成转角。

其中，合成转角的计算公式如下：

$$\Delta g=|\gamma^{u_{ik}}-\gamma^{u_{i^*k^*}}|+|\pi^{u_{ik}}-\pi^{u_{i^*k^*}}|+|\varphi^{u_{ik}}-\varphi^{u_{i^*k^*}}|,\quad \forall t_i,t_{i^*}\in T,$$
$$\text{当 } x_{ik}=x_{i^*k^*}=1 \text{ 时} \tag{3.5}$$

式中，$\gamma^{u_{ik}}$ 为卫星处于时刻 u_{ik} 的滚动角；$\pi^{u_{ik}}$ 为卫星处于时刻 u_{ik} 的俯仰角；$\varphi^{u_{ik}}$ 为卫星处于时刻 u_{ik} 的偏航角。

（2）卫星电量、存储约束。卫星在轨工作伴随着对电量的消耗，这种消耗主要来源于两部分，一是卫星姿态机动所耗费的固定电量以及进姿态机动时每一度耗费的电量，二是卫星执行任务所耗费的电量。卫星在轨工作耗费的电量不能超过卫星最大电量，即

$$\sum_{i=1}^{N}\sum_{k=1}^{|w_i|}x_{ik}l_ie_i+\sum p^{\mathrm{s}}+\sum\Delta g p^{\mathrm{a}}\leqslant E,\quad \forall t_i\in T,\quad \text{当 } x_{ik}=1 \text{ 时} \tag{3.6}$$

式中，E 为卫星最大电量；e_i 为卫星执行任务 t_i 时每一秒所耗费的电量；p^{s} 为每次姿态机动卫星所耗费的固定电量；p^{a} 为姿态机动每一度所消费的电量。

卫星拍摄目标并存储相关图像，全部图像占据的存储不能超过卫星最大存储空间，表示如下：

$$\sum_{i=1}^{N}\sum_{k=1}^{|w_i|}x_{ik}l_im_i\leqslant M,\quad \forall t_i\in T,\quad \text{当 } x_{ik}=1 \text{ 时} \tag{3.7}$$

式中，M 为卫星最大存储空间；m_i 为任务 t_i 在单位时间内所占据的存储容量。

(3) 执行唯一性约束。任务最多被执行一次，不可重复执行，该约束表示如下：

$$\sum_{k=1}^{|w_i|} x_{ik} \leqslant 1, \quad \forall t_i \in T \tag{3.8}$$

(4) 观测有效性约束。任务只有在可见时间窗口范围内执行才被视为调度成功，这也反映了调度对象的本质是目标与卫星的可见时间窗口。该约束表示如下：

$$\mathrm{st}_{ik} \leqslant u_{ik} \leqslant u_{ik} + l_i \leqslant \mathrm{et}_{ik}, \quad \forall t_i \in T, \quad \text{当 } x_{ik} = 1 \text{ 时} \tag{3.9}$$

式中，st_{ik} 为可见时间窗口 w_{ik} 的开始时间；et_{ik} 为可见时间窗口 w_{ik} 的结束时间。

(5) 成像质量约束。卫星执行成像任务时，所拍摄的图像需要满足最低成像质量要求。最低成像质量通常是用户或者卫星管控部门根据任务类型来设定。该约束表示如下：

$$q_i \geqslant c_i, \quad \forall t_i \in T \tag{3.10}$$

式中，q_i 为任务 t_i 的成像质量；c_i 为任务 t_i 需要满足的最低成像质量。

任务成像质量 q_i，可通过计算得到[5]：

$$q_i = 10 - 9\,\frac{|2u_{ik} + l_i - 2w_{ik}^*|}{d_{ik} - l_i}, \quad \text{当 } x_{ik} = 1 \text{ 时} \tag{3.11}$$

式中，w_{ik}^* 为任务 t_i 在时间窗口 w_{ik} 中的最佳成像质量时刻点。

最佳成像质量时刻点 w_{ik}^* 可通过计算得到：

$$w_{ik}^* = \frac{\mathrm{st}_{ik} + \mathrm{et}_{ik}}{2}, \quad \text{当 } x_{ik} = 1 \text{ 时} \tag{3.12}$$

3.2　数据挖掘与元启发式结合的算法设计思路

如何设计高效的求解方法，是组合优化问题研究中不变的主题。本书在绪论部分已全面分析了目前求解卫星任务调度问题最常用的精确求解算法、元启发式算法以及机器学习算法。精确求解算法虽然能够从理论上得到问题的最优解，但往往需要简化问题的约束条件，同时该算法仅面向小规模问题，一旦问题规模扩大，将会耗费巨大的计算时间。杜永浩[118]通过实验证明精确算法求解单星调度问题的效率十分有限，CPLEX 在 7 200 s 的计算时间内仅能调度 20 个成像卫星任务，这显然无法满足实际要求。此外，机器学习算法虽然能够在短时间内求出问题的解，但无法保证解的质量，在大规模问题上的表现与专业优化器相比仍有不小的差距[178]。

近年来，元启发式算法已成为求解组合优化问题的有利手段，在传统制造、航空航天以及国防等领域取得了巨大成功。因此，本书采用元启发式算法求解单星调度问题。元启发式算法是一种通过迭代搜索的方式在决策空间中寻找优质解的

方法。通过研究与实践，元启发式算法尽管能够在可接受的时间范围内求出满意解，但其低智能性导致早熟或过晚收敛、求解效果不稳定等问题。随着问题规模的扩大，元启发式算法的求解性能将面临巨大考验。

基于此，为有效提升元启发式算法的智能性，并引导算法更高效地求解，本书提出了一种数据挖掘与元启发式算法结合的求解思路，在该思路指导下，分别与局部搜索算法和进化算法结合设计了两种新算法。如图 3.2 所示，该方法原理主要是：元启发式算法按照“邻域操作”或“进化操作”的方式在解空间中搜索问题的可行解，数据挖掘则负责从元启发式算法前期搜索产生的数据（高质量解集、低质量解集以及局部最优解等）中，挖掘出有效的“知识”，然后利用这些“知识”引导元启发式算法后续选择算子、构造新解以及更新种群等，提升元启发式算法的求解效率。接下来，本章也将基于该思路，结合卫星任务调度领域知识，设计面向单星调度问题的数据驱动优化算法。

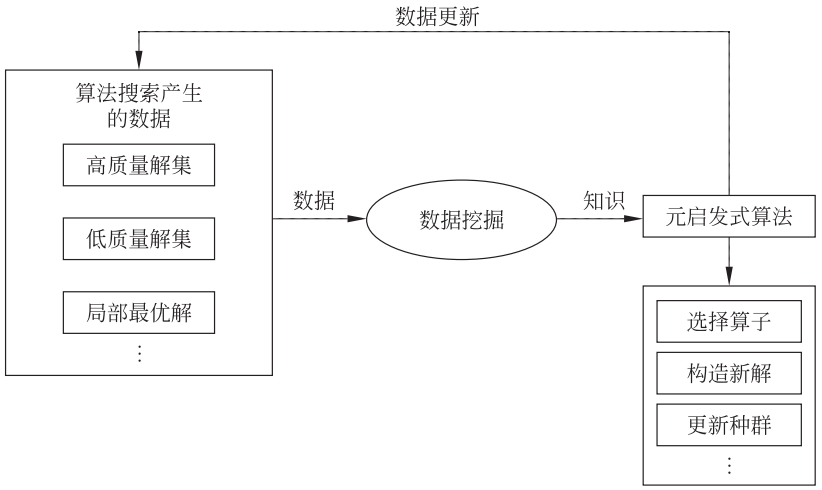


图 3.2 数据挖掘与元启发式算法结合的思路原理图

3.3 基于频繁模式的自适应并行搜索算法

基于上述分析，本书首先采用局部搜索算法与频繁模式挖掘结合，提出了 FPAPSA，该算法包含 4 个部分：①初始解构造策略；②并行搜索策略；③算法算子自适应选择策略；④基于频繁模式的新解构造策略。下面详细分析算法整体框架以及各组件的原理和流程。

3.3.1 算法框架

FPAPSA 算法框架如图 3.3 所示。

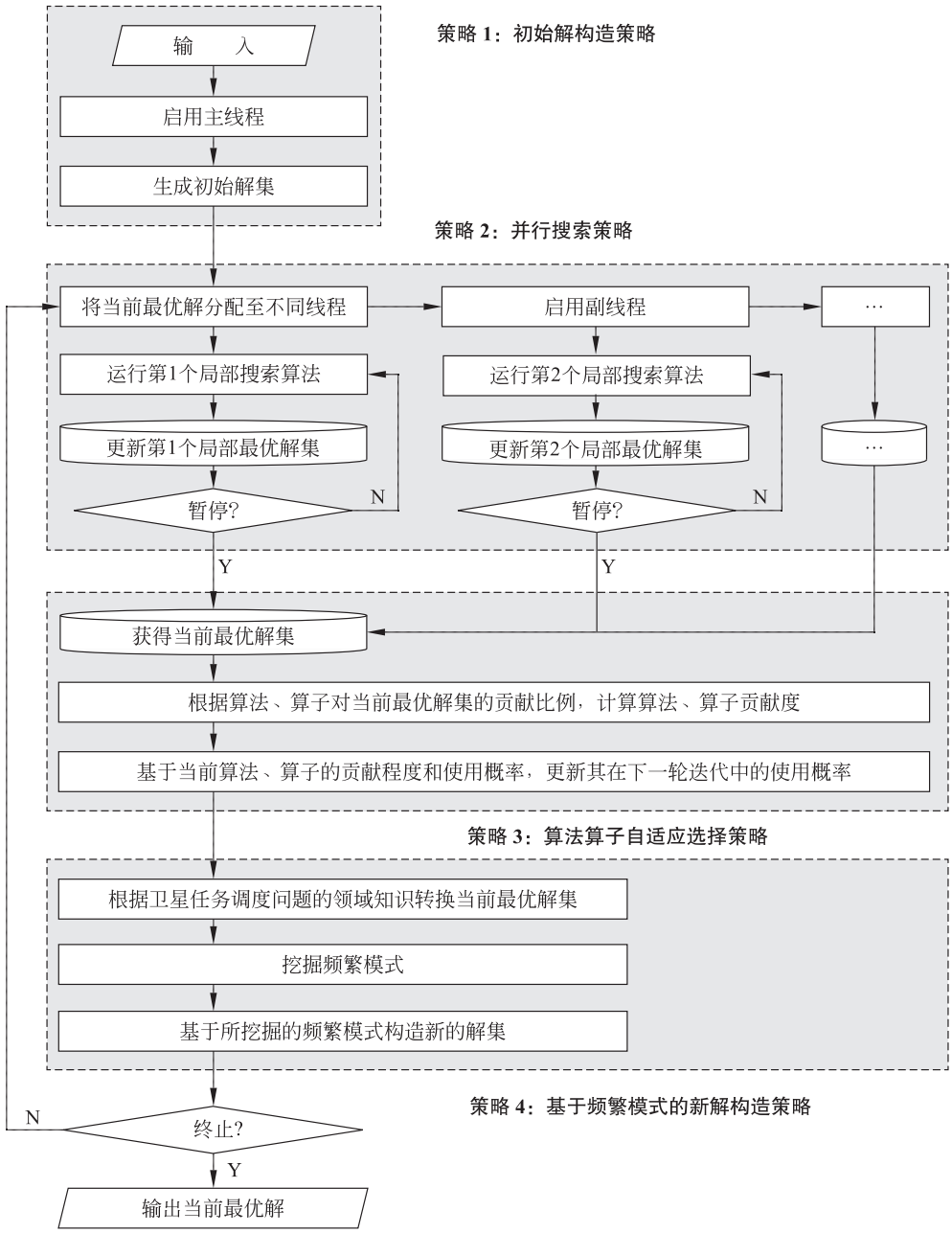


图 3.3　FPAPSA 框架

（1）初始解构造策略。初始解构造策略是 FPAPSA 的第 1 部分。高质量的初始解能够给算法提供一个高起点，引导算法快速进入可行域空间，提升算法的早期搜索效率。FPAPSA 采取“启发式排序规则＋紧前安排”的方法，首先按照启发式

规则对任务进行排序；其次为每个任务安排最早的执行时间，剔除无法完成的任务，生成初始解集。

(2) 并行搜索策略。并行搜索策略是 FPAPSA 的第 2 部分。FPAPSA 内集成了多个局部搜索算法，不同局部搜索算法以并行方式运行，独立更新最优解集，以空间换时间，提升算法局部寻优能力。值得注意的是，该并行搜索策略不仅能够算法并行，即为单颗卫星任务调度问题同时提供不同的局部搜索算法，也能够问题并行，即支持多颗卫星同时开展任务调度，具有优秀的扩展性，本书在第 4 章和第 5 章将会利用并行搜索策略同时求解不同单星调度问题。

(3) 算法算子自适应选择策略。算法算子自适应选择策略是 FPAPSA 主循环的第 3 部分。该策略负责定期评估各算法以及算子的表现，动态更新使用概率，提升算法的自适应能力。

(4) 基于频繁模式的新解构造策略。基于频繁模式的新解构造策略是 FPAPSA 的最后一部分。它负责利用频繁模式挖掘方法从最优解集中提取任务执行的次序关系，利用其构造新解，引导算法开展更高效地搜索。

3.3.2 初始解构造策略

采用“启发式排序规则＋紧前安排”的方法产生初始解集。本章根据领域知识设计了多个排序规则。

(1) 排序规则 1：按照优先级降序对任务进行排序。

(2) 排序规则 2：按照执行时长升序对任务进行排序。

(3) 排序规则 3：按照可见时间窗口数量升序对任务进行排序。

(4) 排序规则 4：对任务随机排序。

首先，随机选择以上排序规则生成任务的初始排序方案。其次，根据紧前安排的策略为每个任务安排最早的、满足约束的时间窗口和执行时刻，剔除无法完成的任务，生成初始解集(其中，初始解集的规模与线程数量相同)。

3.3.3 并行搜索策略

并行搜索策略是 FPAPSA 内循环的第 1 部分。如图 3.4 所示[118]，并行搜索策略集成了多个算法和算子，各算法独立运行。在算法迭代过程中，每个线程采取先进先出的原则维护一个最优解集[118]。并行搜索策略的作用体现在以下 3 方面：①以空间换时间，短时间内充分发挥算法的寻优能力；②通过比较合并各线程维护的最优解集，为后续计算各算法、算子的贡献度提供依据；③最优解集同时服务后续基于频繁模式的新解构造策略，为后续频繁模式挖掘提供高质量数据。

值得注意的是，在本章中，并行策略被设计成“算法并行”，即各线程针对同一个问题选择不同的局部搜索算法。在后续第 4 章、第 5 章中，并行策略被设计成

“问题并行”，即同一个算法同时求解多个单星调度问题。

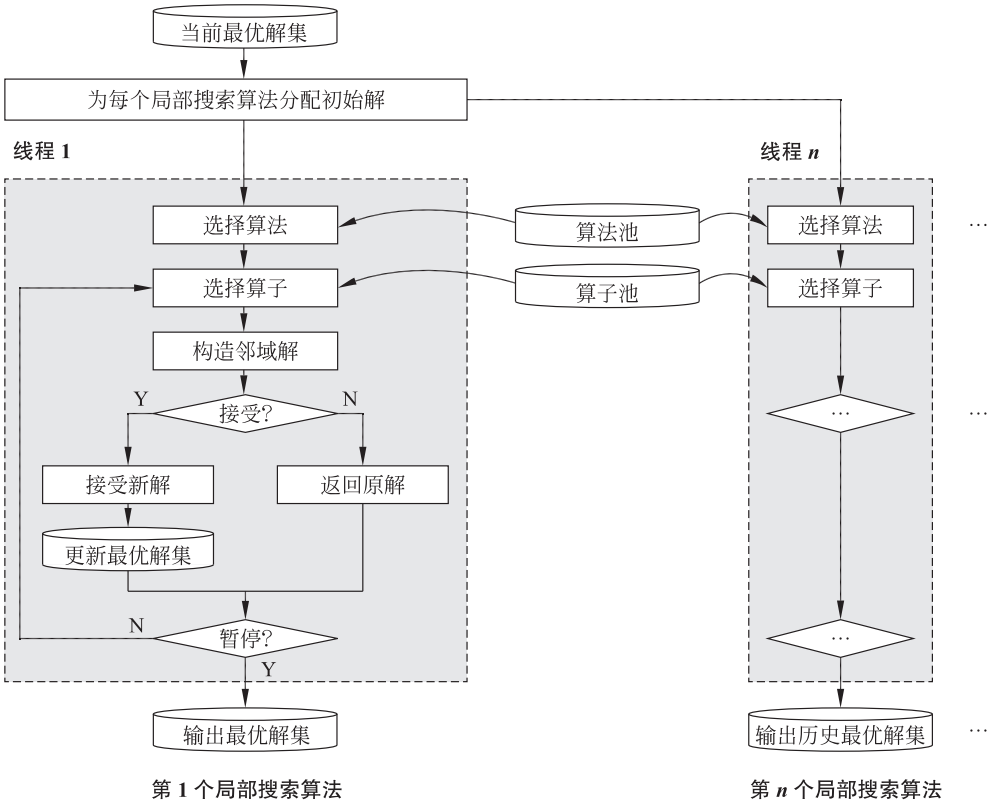


图 3.4　并行搜索策略示意图

1. 算子池

在算法每一次构造邻域解的过程中，算子池将提供可用的邻域构造算子。本书采取“删除＋排序＋快速插入”的方式产生新解，主要步骤描述如下。

（1）步骤 1：依概率选择删除算子。给定当前解，依概率选择某个删除算子从当前解中删除 P_{d} 个任务。采用紧前安排的方法调整破坏的解，为破坏的解中每个任务安排最早的执行时间。

（2）步骤 2：依概率选择排序算子。合并未被调度的任务以及删除的任务，依概率选择某个排序算子对剩余任务进行排序。

（3）步骤 3：采用快速插入方法安排剩余任务。将排序好的任务依次序插入当前解中，某个任务一旦能够被插入当前解中，那么采用紧前安排的方式确定任务的执行时间。若任务无法插入当前解中，那么就舍弃该任务。

上述步骤为一次邻域操作，产生相应的邻域解。下面详细介绍删除算子、排序算子以及快速插入方法。

本章借鉴 He 等[90]以及 Liu 等[12]的研究，设计了 7 种删除算子和 7 种排序算子。本章只介绍算子的原理和时间复杂度，每个算法的具体含义可查阅上述研究，本章在此不再赘述。

7 种删除算子分别如下。

(1) 随机删除算子。随机删除 P_d 个任务，时间复杂度为 $O(n)$。

(2) 最小收益删除算子。删除 P_d 个收益值较低的任务，目的是优先完成收益较高的任务，时间复杂度为 $O(n\log n)$。

(3) 最小单位收益删除算子。删除 P_d 个单位收益较低的任务。任务 t_i 的单位收益可由公式 p_i/l_i 计算得到。与最小收益删除算子目的相同的是，该算子是从单位收益的维度去刻画任务的价值，同样是为了优先完成高价值的任务。时间复杂度为 $O(n\log n)$。

(4) 最大时间窗口数量删除算子。删除 P_d 个可见时间窗口数量较多的任务。任务在卫星上的时间窗口数量越多，后续能被调度的可能性也就越大，因此优先执行时间窗口数量较少的任务。时间复杂度为 $O(n\log n)$。

(5) 最大转换时间删除算子。删除 P_d 个转换时间较长的任务。由于卫星任务调度需要在一定周期内完成，转换时间较长的任务会压缩后续任务执行的时间。时间复杂度为 $O(n\log n)$。

(6) 最大冲突度删除算子。删除 P_d 个冲突度较大的任务。时间复杂度为 $O(n\log n)$。

(7) 最差路径删除算子。将连续 P_d 个任务当作一个路径，删除单位收益最小的路径。假设 $[t_i, t_{i+1}, \cdots, t_{i+P_d-1}]$ 为某个路径，其单位收益可由公式 $\dfrac{\sum p}{z_{i+P_d-1}-u_i}$ 计算得到，其中，$\sum p$ 为该路径上所有任务收益之和。时间复杂度为 $O(n\log n)$。

每次构造新解时，本轮所删除的任务和未调度的任务将进入候选任务池，接着排序算子按照某种规则对候选任务池中的任务进行排序。各排序算子的含义如下。

(1) 随机排序算子。对候选池中的任务随机排序。时间复杂度为 $O(n)$。

(2) 收益排序(profit-soft，PS)算子。对候选池中的任务按照收益值从高到低排序。时间复杂度为 $O(n\log n)$。

(3) 最大单位 PS 算子。对候选池中的任务按照单位收益值从高到低排序。时间复杂度为 $O(n\log n)$。

(4) 最小冲突度排序算子。对候选池中的任务按照冲突度从小到大排序。时间复杂度为 $O(n\log n)$。

(5) 最小时间窗口数量排序算子。对候选池中的任务按照可见时间窗口数量从小到大排序。时间复杂度为 $O(n\log n)$。

(6) 最小转换时间排序算子。对候选池中的任务按照转换时间从小到大排序。时间复杂度为 $O(n\log n)$。

(7) 最小距离排序算子。对候选池中的任务按照到当前解的距离从小到大排序。距离通过任务与当前解中任务的最短转换时间来表示。时间复杂度为 $O(n\log n)$。

候选池中的任务排序完成后,采用快速插入方法依次将任务插入当前解中,具体步骤如下。

(1) 步骤 1:计算任务的时间松弛量。时间松弛量是指一个任务能够被推迟的最晚时间量。敏捷卫星任务调度问题具有时间依赖的特性,后一个任务的开始时间受到前一个任务执行情况的影响,因此采取后向传播的方式计算每个任务的时间松弛。根据时间松弛量可计算出每个任务的最晚开始时间,其中最后一个任务的最晚开始时间仅由时间窗口的最晚开始时间决定,其具体计算公式如下:

$$u_i^{\text{late}}=\begin{cases}\min\{(u_{i+1}^{\text{late}}-\Delta(t_i,t_{i+1})-l_i),\text{et}_i\}, & 1\leqslant i<m\\ \text{et}_i, & i=m\end{cases} \tag{3.13}$$

式中,et_i 为任务 t_i 的理论最晚开始时间;m 为当前解包含任务的数量;u_i^{late} 为任务 t_i 的最晚开始时间。

(2) 步骤 2:选择最优插入位置。将剩余任务依次插入破坏的方案中。假设任务 t_n 插入的位置为任务 t_{i-1} 与任务 t_i,首先根据 t_{i-1} 的结束时间紧前安排任务 t_n,再依据任务 t_n 计算任务 t_i 的最早开始时间,如果任务 t_i 最早开始时间早于 u_i^{late},那么该位置可插入;否则,舍弃该位置。任务插入不同位置会导致方案中任务整体转换时间产生不同增量,用 Δt 表示转换时间增量,即 $\Delta t=\text{TransitionTime}(t_n,t_{i-1})+\text{TransitionTime}(t_i,t_n)-\text{TransitionTime}(t_i,t_{i-1})$,选择转换时间增量最小的位置为最优插入位置。

通过上述“破坏+排序+快速插入”方法产生新解,即完成一次邻域操作。下面介绍 FPAPSA 中集成的不同局部搜索算法。

2. 算法池

本章为并行搜索策略提供 3 种改进后的局部搜索算法,分别是爬山算法,TS 算法和 SA 算法。

算法 3.1、算法 3.2 和算法 3.3 分别给出了上述 3 种算法的伪代码。

在算法 3.1 中,算法第 3 行表示算法须根据算子池中各算子使用概率,采用“轮盘赌”的方式选择某个删除算子和排序算子,进而构造邻域解。后续算法也将沿用该算子选择方式。算法第 5～18 行表示更新最优解集以及对应的操作算子集合。首先,判断构造出的新解 X' 是否已经存在最优解集中,如果存在,那么按照先进先出的方式更新操作算子集合;如果未存在,判断当前最优解集长度达到最大长度 K,一旦达到最大长度,那么比较新解 X' 与最优解集中最劣解 X^{worst} 的收益值,