控制与决策
Control and Decision

**基于数据驱动的自适应并行搜索算法求解多星协同调度问题**

吴健, 姚锋, 杜永浩, 陈宇宁, 何磊, 何永明, 罗绥芝

引用本文:
吴健, 姚锋, 杜永浩, 等. 基于数据驱动的自适应并行搜索算法求解多星协同调度问题[J]. 控制与决策, 2024, 39(12): 4064-4072.

在线阅读 View online: https://doi.org/10.13195/j.kzyjc.2023.0946

---

# 基于数据驱动的自适应并行搜索算法求解多星协同调度问题

吴 健[1,2], 姚 锋[1], 杜永浩[1], 陈宇宁[1], 何 磊[1†], 何永明[1], 罗绥芝[3]

(1. 国防科技大学 系统工程学院，长沙 410073；2. 西安电子科技大学 杭州研究院，杭州 311231；3. 湖南师范大学 旅游学院，长沙 410081)

**摘 要:** 针对元启发式算法在求解多星协同调度问题时暴露出的过早或过晚收敛、稳定性较差等问题,提出一种基于数据驱动的自适应并行搜索算法. 首先,根据领域知识设计多个任务分配算子,目的是将多星协同调度问题转化为多个单星任务调度问题. 然后,启动多个线程并行、独立求解各单星任务调度问题. 在算法迭代过程中,各线程依据概率选择不同的邻域操作算子,并且动态更新精英解集和邻域操作算子概率. 接着,对精英解集挖掘频繁模式,提取高质量解中有价值的知识并构造新解. 最后,将单星任务调度的结果反馈给任务分配层,指导算法开展新一轮的任务分配. 仿真实验表明,所提出的算法能够在有限时间内获得高质量的解,在不同的场景下均能表现出良好的适用性和优化效果.

**关键词:** 数据驱动；多星协同调度；自适应并行搜索；单星任务调度；频繁模式挖掘

**中图分类号:** V474；TP18 **文献标志码:** A

**DOI:** 10.13195/j.kzyjc.2023.0946

**引用格式:** 吴健,姚锋,杜永浩,等. 基于数据驱动的自适应并行搜索算法求解多星协同调度问题 [J]. 控制与决策, 2024, 39(12): 4064-4072.

# A data-driven adaptive parallel search algorithm for multiple agile satellites cooperative scheduling problem

*WU Jian*[1,2], *YAO Feng*[1], *DU Yong-hao*[1], *CHEN Yu-ning*[1], *HE Lei*[1†], *HE Yong-ming*[1], *LUO Sui-zhi*[3]

(1. College of Systems Engineering，National University of Defense Technology，Changsha 410073，China；2. Hangzhou Institute of Technology，Xidian University，Hangzhou 311231，China；3. College of Tourism，Hunan Normal University，Changsha 410081，China)

**Abstract:** When solving the multiple agile satellites cooperative scheduling problem, the metaheuristics faces many problems due to their low intelligence, such as premature or late convergence, poor stability, etc. To solve these problems more efficiently, a data-driven adaptive parallel search algorithm is proposed. Firstly, some task allocation operators are designed to based on domain knowledge, with the purpose of transforming the multiple agile satellites cooperative scheduling problem into multiple single-satellite task scheduling subproblems. Then, multiple threads are started to parallelly and independently solve each single-satellite task scheduling problem. During algorithm iterations, each thread selects different neighborhood operators based on probability, and dynamically updates the probability of neighborhood operators and elites. Next, the frequent pattern mining method is applied to extract knowledge from the elites to construct new solutions. Finally, all single-satellite task scheduling results are fed back to the task allocation layer to start a new allocation. The simulation results show that the proposed algorithm can obtain high-quality solutions within a limited time, and has good applicability and optimization effects in different scenarios.

**Keywords:** data-driven；multiple agile satellites cooperative scheduling；adaptive parallel search；single-satellite task scheduling；frequent pattern mining

## 0 引 言

对地观测卫星(以下简称卫星)作为重要的天基信息获取平台,在国土普查、地形勘探、军事情报获取等方面发挥了积极作用[1]. 作为高价值观测平台,高效的卫星任务调度算法不仅能够提升用户的满意度,还能减少卫星资源的消耗. 卫星任务调度是指在

**收稿日期:** 2023-07-06；**录用日期:** 2024-03-11.

**基金项目:** 国家自然科学基金项目 (72201272, 72001212,72201273, 72271240)；湖南省研究生科研创新项目 (CX20210031).

**责任编委:** 侯忠生.

†通讯作者. E-mail: helei@nudt.edu.cn.

不违反卫星使用约束的前提下,针对用户任务需求,生成卫星可执行的任务序列以及具体执行时间,以达到最大化完成任务数量、最大化完成任务收益等目标. 求解多星协同调度问题的难点主要是: 1) 敏捷卫星具备滚动、俯仰、偏航3个方向的自由度,可以提前或延后观测目标,可见时间窗口(visible time window, VTW)长度被大大延长[2],因此需要确定在该VTW下的任务具体执行时间; 2) 对比单颗敏捷卫星仅需要决策任务执行时间,求解多星协同调度问题还要考虑任务应由哪颗卫星执行. 上述两方面共同增加了多星协同调度问题的求解难度.

多星协同调度问题的求解思路主要有一体调度和双层调度这两类. 一体调度是指算法在搜索过程中同时决定任务由哪颗卫星去执行以及卫星执行该任务的时间. 一体调度面向的对象是卫星与目标的VTW[3-7],其优势是能够遍历问题解空间,在时间不受限制的前提下,能够找到问题的最优解;其劣势也比较明显,随着任务数量和卫星规模的扩大,解空间呈指数级增长,更加容易出现“维数灾难”,算法面临“求解质量-求解速度”的矛盾更加突出. 双层调度指首先采用某种方法将任务分配到不同卫星上,每颗卫星再根据所分配的任务开展任务调度. 这种先分配后调度的方式有效缩减了问题规模,结合并行方式,大大节省了算法求解时间. 相较于一体调度,由于无法遍历解空间,双层调度容易丢失优质解. 如果能够提出合理的任务分配方法,再结合高效的单星任务调度算法便能够很好弥补双层调度的劣势.

方法层面,精确求解算法和元启发式算法是求解卫星任务调度问题常用的两类方法. 精确求解算法的思路通常是将问题建模成整数规划模型,再通过动态规划[8]、分支定界[9-10]、列生成[11]等方法求解. Peng等[8]提出了一种双向动态规划的迭代局部搜索算法求解卫星任务调度问题,其中动态规划的作用是替代传统资源分配规则,帮助任务在VTW内找到最理想的执行时机. 王沛等[11]将多星协同调度问题分解成集合配置主问题和包含VTW的最短路径子问题,采用Cplex来求解主问题,子问题则采用动态规划方法. 精确求解算法尽管能够求出问题的最优解,但缺点也同样突出,具体是: 1) 需要简化问题的数学模型,从而导致最终方案往往违反实际约束; 2) 仅适用于中、小规模问题,在大规模问题上表现欠佳,甚至无法在可接受时间内给出问题的解. 元启发式算法是基于单个解(种群),通过邻域操作(进化操作)产生新解(新种群),并有选择性地向优质解空间移动的一类通用优化方法. Liu等[12]在传统自适应大邻域搜索算法(adaptive large neighborhood search algorithm, ALNS)基础上,设计专门的破坏算子和排序算子,成功求解了单星任务调度问题. He等[6]在此基础上,提出了自适应任务分配策略,成功将问题从单星任务调度扩展到多星协同调度. 杜永浩等[7]针对卫星通信、导航、遥感以及测控这4类典型任务,提出了统一化建模思路,设计了包含多种策略的元启发式算法. 针对大规模卫星数传任务调度问题,Zhang等[13]在完成任务收益和卫星负载均衡的双重需求下,设计了一种多目标优化算法. Wu等[14]利用遗传算法迭代产生的数据训练神经网络,利用其提升种群中个体的适应度. 元启发式算法尽管能够有效求解复杂优化问题,在可接受时间范围内给出问题的解,但因其低智能性,算法求解效果往往不稳定,容易早熟或过晚收敛.

基于上述分析,本文提出一种基于数据驱动的自适应并行搜索算法(data-driven adaptive parallel search algorithm, DDAPS)求解多星协同调度问题. DDAPS算法采用“任务分配+单星调度”的双层调度框架,核心思想是基于元启发式算法搜索过程中产生的数据,利用数据挖掘方法从数据中提升有价值的知识,进一步利用该知识引导算法后续更高效搜索.

## 1　多星协同调度问题建模

### 1.1　问题描述与基本假设

多星协同调度问题是指给定多颗卫星以及多个观测任务,在不违反资源限制和任务要求的前提下,生成卫星执行的任务序列,确定任务执行的具体时间. 多星协同调度问题已经被证明是NP-Hard问题[6],因而不存在多项式时间内求解的精确算法. 为了更方便研究该问题,本文作出如下假设.

1) 本文所指的任务均是卫星可一次执行的元任务,需要多频次观测的目标、大区域目标、移动目标等复杂任务可通过任务分解、条带划分等方式处理成元任务;

2) 任务在执行过程中不能被抢占,一旦任务开始执行,不允许被中途终止;

3) 每颗卫星只能搭载一个成像载荷,这意味着卫星同时只能执行一个任务;

4) 卫星在实际工作中需要对日充电以及不定时向地面站传输数据,释放存储,因此本文不考虑卫星电量约束和存储约束.

### 1.2　参数设置与模型构建

首先给出多星协同调度问题的相关参数定义,见表1.

**表1 参数定义**

| 参数 | 参数名称 | 参数 | 参数名称 |
|---|---|---|---|
| $t_i$ | 第$i$个任务 | $s_j$ | 第$j$个卫星 |
| $T$ | 任务集合 | $\varphi_{jt}$ | $t_i$在时刻$t$的俯仰角 |
| $N$ | 任务数量 | $W$ | VTW集合 |
| $l_i$ | 任务$l_i$的执行时间 | $w_{ijk}$ | $t_i$在$s_j$上的第$k$个VTW |
| $S$ | 卫星资源集合 | $\lvert w_{ij}\rvert$ | $t_i$在$s_j$上的VTW数量 |
| $M$ | 卫星数量 | $\mathrm{st}_{ijk}$ | $w_{ijk}$的开始时间 |
| $\gamma_{jt}$ | 任务$t_i$在时刻$t$的滚动角 | $\mathrm{et}_{ijk}$ | $w_{ijk}$的结束时间 |
| $\pi_{jt}$ | 任务$t_i$在时刻$t$的俯仰角 | $d_{ijk}$ | $w_{ijk}$的时间长度 |
| $p_i$ | 任务$t_i$的收益值 | | |

多星协同调度问题的决策变量为$(x_{ijk},u_{ijk})$. 其中:$x_{ijk}$为一个二元变量,$x_{ijk}=1$表示任务$t_i$在时间窗口$w_{ijk}$内执行,否则$x_{ijk}=0$;决策变量$u_{ijk}$表示任务$t_i$在时间窗口$w_{ijk}$内的开始执行时间.

构建多星协同调度问题的数学模型如下所示:

$$\max \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{|w_{ij}|} x_{ijk}p_i. \tag{1}$$

$$\sum_{j=1}^{M}\sum_{k=1}^{|w_{ij}|} x_{ijk}\leqslant 1,\ \forall t_i\in T. \tag{2}$$

$$\mathrm{st}_{ijk}\leqslant u_{ijk}<u_{ijk}+l_i\leqslant \mathrm{et}_{ijk},\ \text{if}\ u_{ijk}=1,\ \forall t_i\in T. \tag{3}$$

$$\rho_{x_{ijk}x_{i^*jk^*}}=\begin{cases}1,\ t_i\ \text{is the precursor to}\ t_{i^*};\\ 0,\ \text{otherwise};\end{cases}$$

$$\text{if}\ x_{ijk}=x_{i^*jk^*}=1,\forall t_i,t_{i^*}\in T,\forall s_j\in S. \tag{4}$$

$$u_{ijk}+l_i+\Delta\text{time}(t_i,t_{i^*})\leqslant u_{i^*jk^*},\ \text{if}\ \rho_{x_{ijk}x_{i^*jk^*}}=1. \tag{5}$$

$$\Delta\text{time}(t_i,t_{i^*})=\begin{cases}11.66,\ \Delta g\leqslant 10;\\ 5+\Delta g(t_i,t_{i^*})/v_1,\ 10<\Delta g\leqslant 30;\\ 10+\Delta g(t_i,t_{i^*})/v_2,\ 30<\Delta g\leqslant 60;\\ 16+\Delta g(t_i,t_{i^*})/v_3,\ 60<\Delta g\leqslant 90;\\ 22+\Delta g(t_i,t_{i^*})/v_4,\ \Delta g>90.\end{cases} \tag{6}$$

$$\Delta g(t_i,t_{i^*})=|\gamma^{u_{ijk}}-\gamma^{u_{i^*jk^*}}|+|\pi^{u_{ijk}}-\pi^{u_{i^*jk^*}}|+|\varphi^{u_{ijk}}-\varphi^{u_{i^*jk^*}}|,\ \text{if}\ \rho_{x_{ijk}x_{i^*jk^*}}=1. \tag{7}$$

式(1)是多星协同调度问题的目标函数,表示最大化已调度任务的收益. 该目标函数是卫星任务调度研究中常用的目标函数,也是实际业务中卫星管控方首要关切的指标. 式(2)是任务执行唯一性约束,代表每个任务最多被执行一次,不能重复执行. 式(3)表示任务必须安排在可见时间窗口内执行,这是为了保证任务执行的有效性. 式(4)中的$\rho_{x_{ijk}x_{i^*jk^*}}$是一个二元变量,$\rho_{x_{ijk}x_{i^*jk^*}}=1$表示任务$t_i$是任务$t_{i^*}$的前驱任务,否则为0. 式(5)表示同一颗卫星执行不同任务需要满足转换时间约束,转换时间约束是卫星任务调度问题中最常见也是最核心的约束之一,反映了敏捷卫星任务调度具有时间依赖的特性. 式(6)和(7)为任务之间的姿态转换时间计算公式,其中$v_1\sim v_4$表示4个不同姿态角度的转换速度,$\Delta g$表示任务之间的姿态角度(滚动角、俯仰角以及偏航角)的差值.

## 2 求解算法

针对多星协同调度问题,本文提出了DDAPS算法来求解. 如图1所示,算法采用“任务分配+单星调度”双层调度的整体框架.

### 2.1 任务分配

算法设计了多个与问题特征相关的任务分配算子,目的是将任务快速分配给不同卫星,将多星协同调度问题转换成单星任务调度问题,降低问题规模,缩减算法搜索空间. 本文借鉴了文献[6]设计的多星任务分配算子,并在此基础上增加了负载均衡分配算子和历史分配算子.

随机分配算子:将任务随机安排给有VTW的卫星,目的是增强算法的随机性.

卫星位置分配算子:如图2所示,将任务分配给具有最小姿态角的卫星,目的是减少卫星的姿态转换时间,从而预留更多的时间窗口安排后续任务.

时间窗口数量分配算子:任务在卫星上的VTW越多,任务执行的概率越大. 该算子根据任务在卫星上的VTW数量,优先将任务分配给VTW较多的卫星.

负载均衡分配算子:该算子是根据卫星已被安排的任务数量,优先将任务分配给已安排任务较少的卫星,目的是减少任务的冲突度,提升任务被执行的概率.

历史分配算子:该算子优先选择上一轮未被选择的分配算子,目的是扩大算法的搜索空间,避免陷入局部最优.

最早开始时间分配算子:将任务分配给最早能够执行的卫星,目的是预留足够多的时间资源安排其他任务.

### 2.2 单星调度

本文设计了一种自适应局部搜索算法(adaptive local search algorithm, ALSA)并行求解单星任务调度问题,该算法的主要特点有:1)参考文献[6,15]的研究,算法的邻域操作算子被定义为“破坏+排序+快速插入”. 2)为防止算法陷入局部最优,充分发挥算法的寻优能力,加入了禁忌策略和模拟退火中关于

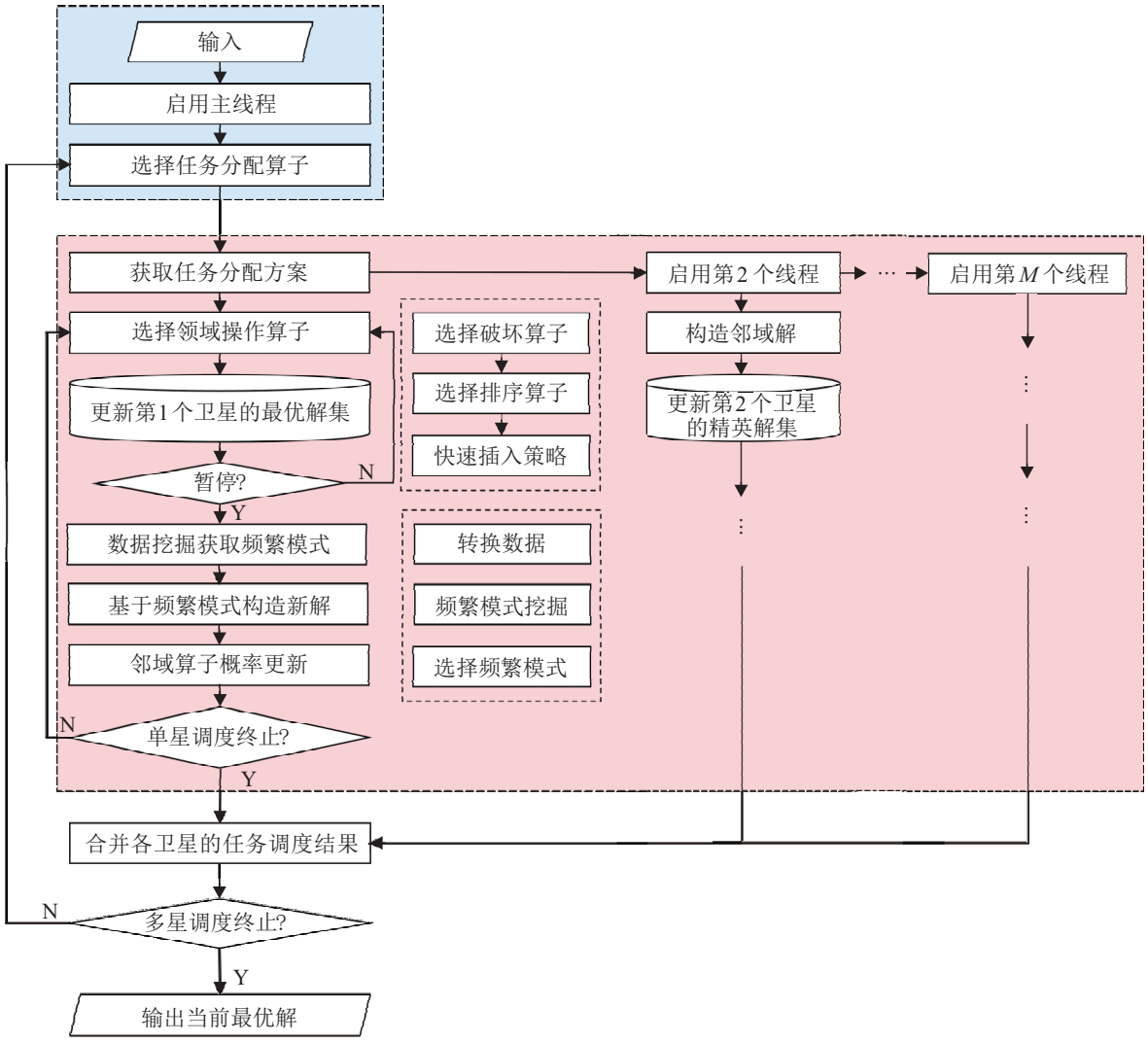


**图 1　算法框架**

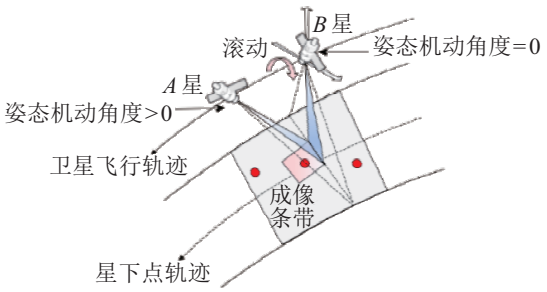


**图 2　卫星位置分配算子示意图**

解的接受准则策略. 禁忌策略是指, 如果用构造的新解替换当前解, 那么本次插入的任务在未来 $t$ 代被禁止删除; 反之, 本次破坏的任务在未来 $t$ 代被禁止破坏. ALSA 算法终止条件有 3 个, 一是达到最大迭代次数, 二是达到连续未提升的代数, 三是所有任务均被全部调度. 满足上述三者之一, 算法随即停止. ALSA 算法伪代码如下所示.

输入: 初始解 $x_o$, 目标函数 $F(x)$, 精英集合长度 $k$, 邻域算子集合 $O$, 精英解集 $U_{\mathrm{TH}}$, 使用的邻域算子集合 $U_o$, 禁忌解集长度 $t$, 降温系数 $c$, 终止条件;

输出: 最优解 $x_b$.

step 1: 初始化算法参数.

step 2: $x \leftarrow x_o, x_b \leftarrow x_o, U_x \leftarrow \varnothing, U_o \leftarrow \varnothing, T \leftarrow F(x_o)$

step 3: while 未达到终止条件 do

step 4:　依据概率从 $O$ 中选择破坏算子 $r$ 和排序算子 $s$

step 5:　构造邻域解 $x_n \leftarrow s(r(x))$

step 6:　if $x_n$ in $U_x$ then

step 7:　　将 $r$ 和 $s$ 记录在 $U_o$

step 8:　else

step 9:　　if $U_x$ 的规模 $< K$ then

step 10:　　将 $r$ 和 $s$ 记录在 $U_o$

step 11:　　将 $x_n$ 记录在 $U_x$

step 12:　else

step 13:　　找到精英解集最劣解 $x_{\mathrm{worst}} \leftarrow \arg\min\{F(x) : x \in U_x\}$

step 14:　　if $F(x_n) \geqslant F(x_{\mathrm{worst}})$ then

step 15: 用$x_n$替换最劣解$x_{\text{worst}}$
step 16: 将$r$和$s$记录在$U_o$
step 17: end if
step 18: end if
step 19: end if
step 20: 更新禁忌解集
step 21: if $F(x_n) \geqslant F(x)$ then
step 22: $x \leftarrow x_n, x_b \leftarrow x_n$
step 23: else
step 24: if random$(0,1) < \exp\{100/T \times (F(x_n)-F(x))/F(x)\}$ then
step 25: $x \leftarrow x_n$
step 26: end if
step 27: $T \leftarrow T \times c$
step 28: 基于频繁模式挖掘的新解构造
step 29: 基于竞争的算子概率自适应更新
step 30: end while

### 2.2.1 邻域操作算子

邻域操作算子的主要含义是:假设目前解的长度为$L_{\text{solution}}$,首先选择某个破坏算子从当前解中删除$N$个任务,并对解中剩余的任务紧前安排;然后将本次删除的任务和剩余未安排的任务按照某种规则进行排序;最后采用快速插入策略依次将剩余全部任务插入到当前解中. 表2和表3分别给出了破坏算子和排序算子的时间复杂度,算子具体含义可参考文献[6, 15].

**表 2 破坏算子时间复杂度**

| 破坏算子 | 时间复杂度 |
|---|---|
| 随机删除 | $o(n)$ |
| 最小收益删除 | $o(n\log n)$ |
| 最小单位收益删除 | $o(n\log n)$ |
| 最大转换时间删除 | $o(n\log n)$ |
| 最大VTW数量删除 | $o(n\log n)$ |
| 最大冲突度删除 | $o(n\log n)$ |
| 最差任务序列删除 | $o(n)$ |

**表 3 排序算子时间复杂度**

| 排序算子 | 时间复杂度 |
|---|---|
| 随机排序 | $o(n)$ |
| 最大收益排序 | $o(n\log n)$ |
| 最大单位收益排序 | $o(n\log n)$ |
| 最小转换时间排序 | $o(n\log n)$ |
| 最少VTW数量排序 | $o(n\log n)$ |
| 最小冲突度排序 | $o(n\log n)$ |
| 最小距离排序 | $o(n\log n)$ |

快速插入策略的基本做法如下.

1) 计算时间松弛. 首先计算原方案中每个任务的时间松弛. 时间松弛是指一个任务能够被推迟的最晚时间. 由于卫星任务调度问题具有时间依赖特性,后一个任务开始时间受到前一个任务结束时间的影响,因此采取后向传播的方式计算每个任务的时间松弛. 根据时间松弛可计算出每个任务的最晚开始时间$u^{\text{late}}$,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,如下所示:

$$u_i^{\text{late}} = \begin{cases} \min\{(u_{i+1}^{\text{late}} - \Delta\text{time}(t_{i+1}, t_i) - l_i), \text{et}_i\}, \\ \quad 1 \leqslant i < m; \\ \text{et}_i,\ i = m. \end{cases} \tag{8}$$

其中:$m$表示方案中任务的总数;$\text{et}_i$表示任务$t_i$的理论最晚开始时间,由任务$t_i$的最晚时间窗口决定.

2) 选择最优插入位置. 将剩余任务依次插入到破坏的方案中. 假设任务$t_n$插入的位置为任务$t_{i-1}$与任务$t_i$之间,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$u_i^{\text{late}}$,那么该位置可插入;否则,舍弃该位置. 任务插入到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$\Delta t$表示转换时间增量,即$\Delta t = \Delta\text{time}(t_n, t_{i-1}) + \Delta\text{time}(t_i, t_n) - \Delta\text{time}(t_i, t_{i-1})$,选择转换时间增量最小的位置为最优插入位置.

### 2.2.2 基于频繁模式挖掘的新解构造策略

算法在搜索过程中产生大量的中间数据,这些中间数据包含了大量高质量解. 如何利用这些数据挖掘出有效知识,指导算法更高效搜索是本部分的重点. 基于频繁模式挖掘的新解构造策略本质上是挖掘不同精英个体的共性知识,并依据共性知识构造新解来指导算法开展下一轮更高效的搜索. 该策略对应算法的step 28,主要步骤如下.

1) 数据转换. 转换时间约束是敏捷卫星任务调度问题中最重要的约束之一,反映了问题时间依赖的特性,具体是前一个任务的结束时间影响下一个任务是否能够执行以及执行的具体时间. 因此,本文将前一个任务与后一个任务的次序关系定义为卫星任务调度问题的知识,具体见图3.

| | |
|---|---|
| 方案1 | 5,7,4,3,1,9,12,65 |
| 方案2 | 4,6,12,65,3,9,14 |
| 方案3 | 3,7,5,4,11,34 |

⇒

| | |
|---|---|
| 方案1 | [5,7],[7,4],[4,3],[3,1],[1,9],[9,12],[12,65] |
| 方案2 | [4,6],[6,12],[12,65],[65,3],[3,9],[9,14] |
| 方案3 | [3,7],[7,5],[5,4],[4,11],[11,34] |

**图 3 数据转换示意图**

2) 频繁模式挖掘. 频繁模式挖掘是数据分析中常用的手段之一,其目的是挖掘频繁出现在数据集中的模式. 本文采用FP-Growth这一经典频繁模式挖掘方法. 如图4所示,FP-Growth方法通过建立FP-tree

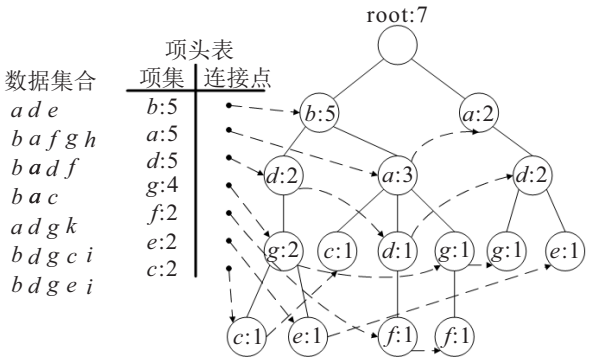


**图 4**　FP-tree的构建过程

减少对数据库的扫描次数从而高效地发现频繁项集. 本文重点不是讨论FP-Growth方法的具体原理, 而是将重点放在如何将其应用于卫星任务调度问题求解. 方法具体细节可参考文献[16].

如图5所示, 本文首先需要将精英解切分成粒度为$s$的子序列. 先验实验表明, 当精英解中的任务数量超过50, FP-Growth方法无法在可接受时间范围内给出结果. 为了使算法适用于大规模任务场景, 本文对精英个体进行了切分, 该方法虽然降低了一定的准确度, 但能够更加快速挖掘频繁项集.

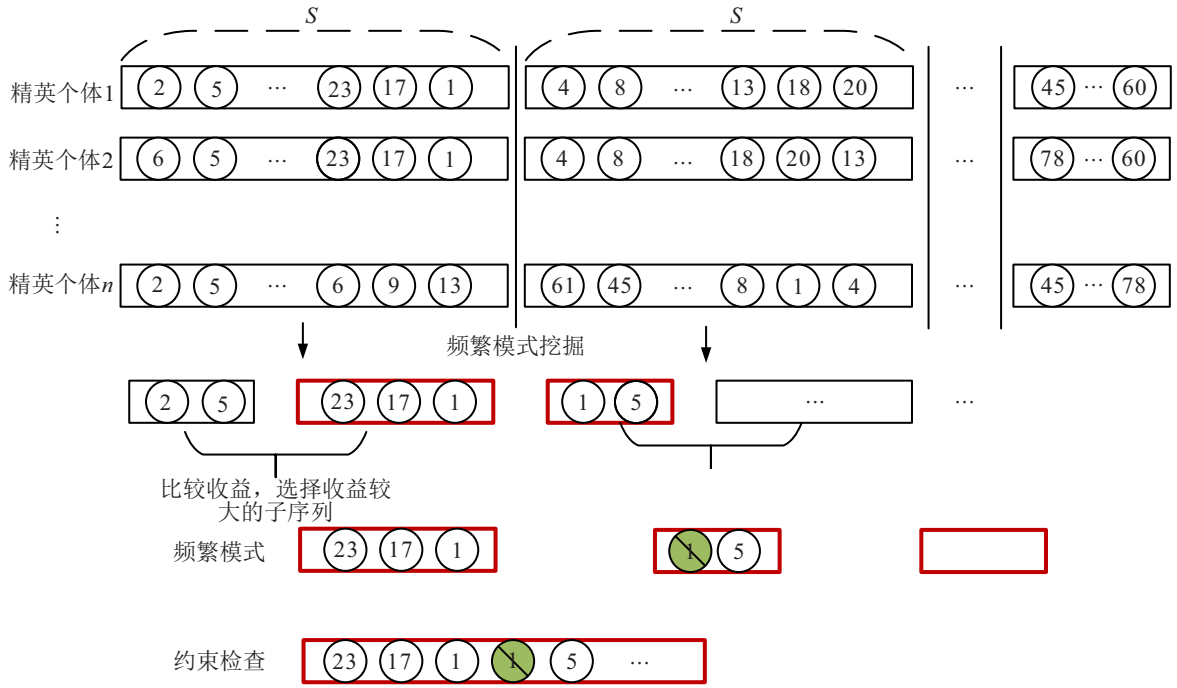


**图 5**　频繁模式挖掘在卫星任务调度问题上的应用

3) 基于精英个体的新解构造. 通过FP-Growth方法挖掘出来多个频繁模式, 选择收益值较大的子序列, 同时进行任务唯一性约束检查, 产生任务调度方案$x^o$. 接着从精英个体集合中选择收益值最大的个体$x^{\text{best}}$来引导新解的构造, 具体是: 首先找到$x^{\text{best}}$中没有出现在$x^o$的任务, 按顺序将其加入到待插入任务集合$T_{\text{insert}}$; 然后, 将剩余全部任务按照收益值降序加入到$T_{\text{insert}}$; 最后, 采用快速插入策略将$T_{\text{insert}}$中的任务依次插入到$x^o$.

### 2.2.3　基于竞争的算子概率自适应更新

基于竞争的算子概率自适应更新是DDAPS算法的最后一部分, 对应算法step 30. 该部分负责在算法迭代搜索过程中, 通过计算算子的贡献度来动态更新算子的选择概率, 提升算法的自适应能力. 各线程负责更新各单星任务调度算法中破坏算子和排序算子的概率.

各单星任务调度算法中的破坏算子和排序算子概率更新方式如下: 假设在线程TH中, 精英解集$U_{\text{TH}}$的大小为$\text{num}_{\text{TH}}$, 其中第$i$个破坏算子贡献$\text{num}_R^i$个精英个体, 第$j$个排序算子贡献$\text{num}_S^j$个精英个体. 根据下式分别求得第$i$个破坏算子和第$j$个排序算子在本轮的贡献度:

$$c_R^i = \text{num}_R^i / \text{num}_{\text{TH}}, \tag{9}$$

$$c_S^j = \text{num}_S^i / \text{num}_{\text{TH}}. \tag{10}$$

第$i$个破坏算子在本轮的概率$p_R^i$和第$j$个排序算子在本轮的概率$p_S^j$可由下式更新:

$$p_R^i = p_R^{*i} \times \omega + c_R^i \times (1-\omega), \tag{11}$$

$$p_S^i = p_S^{*i} \times \omega + c_S^i \times (1-\omega). \tag{12}$$

其中: $p_R^{*i}$和$p_S^{*j}$表示上一轮的破坏算子和排序算子的概率; $\omega$表示上一轮的破坏算子和排序算子的所占比重, 也被称为历史概率所占比重.

在算法最开始时,所有破坏算子概率设置为相同的值,所有排序算子概率同样设置为相同的值.

## 3 实验分析

本文设计了多个多星调度场景来验证所提出算法的有效性. 为创造公平的实验环境,算法和对比算法均使用 Python 3.7.6 编写,算法运行环境为 Intel (R) Core (TM) i7-8700 CPU 3.30 GHz, Windows Server 2010 系统与 16 GB 内存.

### 3.1 实验场景和算法参数设置

由于多星任务调度问题并没有公开测试数据集,本文使用 Liu 等[12] 描述的方法来生成实验场景. 所有场景的调度周期均为 2013-04-20 T 00 : 00 : 00 到 2013-04-20 T 23 : 59 : 59. 卫星轨道可由轨道 6 根数确定,参数见表 4,仿真场景见图 6. 除轨道参数,每颗卫星其余属性相同,$v_1 = 1.5, v_2 = 2, v_3 = 2.5, v_4 = 3$. DDAPS 算法参数设置见表 5,其中 $|T|$ 是分配给该卫星的任务数量.

表 4 卫星轨道参数

| ID | 半长轴/km | 偏心率 | 倾角 | 近地点角 | 升交点赤经 | 真近点角 |
|---|---|---|---|---|---|---|
| 1 | 7 200 | 0.000 627 | 96.576 | 0 | 175.72 | 0.075 |
| 2 | 7 200 | 0.000 627 | 96.576 | 0 | 145.72 | 30.075 |
| 3 | 7 200 | 0.000 627 | 96.576 | 0 | 115.72 | 60.075 |
| 4 | 7 200 | 0.000 627 | 96.576 | 0 | 85.72 | 90.075 |
| 5 | 7 200 | 0.000 627 | 96.576 | 0 | 55.72 | 120.075 |

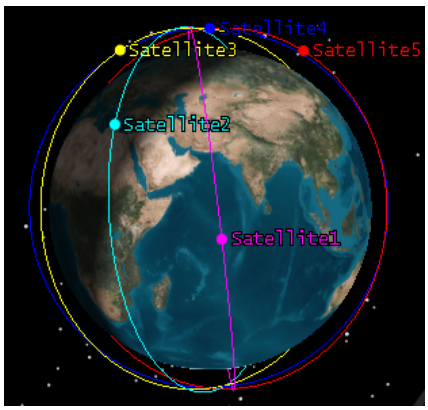


图 6 仿真场景

表 5 算法参数设置

| 参数 | 参数数值 |
|---|---|
| 任务分配次数 | 10 |
| 单星任务调度最大迭代次数 | 100 |
| 最大未提升代数 | 30 |
| 破坏任务数量 | 10%$\|T\|$ |
| 降温系数 | $1\,000^{1/1\,000}$ |
| 禁忌列表长度 | 10%$\|T\|$ |
| 精英解集规模 | 20 |
| 用于频繁模式挖掘的精英个体数量 | 10 |
| 最小支持度% | 20 |
| 算子历史概率所占比重% | 50 |

### 3.2 实验结果分析

为了更好表现本文所提出算法的优越性,本文为其设置 3 种对比算法,分别是基于任务自适应任务分配的 ALNS 算法 (A-ALNS)[6]、基于任务随机分配的 ALNS/I 算法 (R-ALNS/I)[12] 以及基于任务随机分配的改进遗传算法 (R-GA). 由于本文设计的 DDAPS 算法采用的是“任务分配+单星调度”双层调度的求解框架,这 3 种对比算法也是基于该框架. 实验共统计两个指标: 一个是收益率,用被调度任务的收益与全部任务收益的比值来表示,反映算法的求解质量; 二是算法的运行时间,用来反映算法的求解速度.

如表 6 所示,在算法收益率方面, DDAPS 算法在全部的实验场景中都取得了最佳的收益率,综合求解质量最佳. 主要原因在于 DDAPS 算法集成了多个与问题特征相关的任务分配算子,并且主循环采用了局部搜索算法的架构,充分发挥算法的寻优能力,有效寻找优质解. 同时,加入数据驱动机制,能够有效利用局部搜索算法产生的中间数据,挖掘任务与任务之间的关联关系,引导算法后续搜索方向,提升优化效果. 另外,算法会定期更新邻域构造算子的选择概率,满足算法对自适应的要求. 尽管 A-ALNS 算法和 R-ALNS/I 算法的求解表现不如 DDAPS 算法,但仍能够获得较高质量的解,并且与 DDAPS 算法的差距较小. 这说明上述两种算法同样能够很好地求解多星

表 6 算法收益率对比结果

| 场景 ID | 任务数量 | 卫星规模 | DDAPS/% | A-ALNS/% | R-ALNS/I/% | R-GA/% |
|---|---|---|---|---|---|---|
| 1 | 600 | 3 | **98.00** | 97.23 | 95.70 | 26.34 |
| 2 | 700 | 3 | **96.34** | 93.34 | 92.96 | 22.15 |
| 3 | 800 | 3 | **94.48** | 89.86 | 93.42 | 19.41 |
| 4 | 800 | 4 | **99.98** | 99.82 | 99.83 | 25.65 |
| 5 | 900 | 3 | **92.02** | 89.63 | 89.29 | 17.81 |
| 6 | 900 | 4 | **99.94** | 99.60 | 97.77 | 21.75 |
| 7 | 1 000 | 3 | **88.93** | 86.97 | 87.31 | 15.06 |
| 8 | 1 000 | 4 | **98.33** | 97.49 | 97.85 | 19.92 |
| 9 | 1 000 | 5 | **99.03** | 98.76 | 98.07 | 25.21 |

协同调度问题. 求解质量最差的算法是R-GA算法, 之所以R-GA算法的求解质量不佳,原因在于多星协同调度问题约束复杂,通过交叉、变异等进化操作产生的个体大多是不可行解,违反约束.

表7展示了DDAPS算法、A-ALNS算法和R-ALNS/I算法的求解时间,由于R-GA算法的求解质量与其余算法差距较大,本文不再统计其求解时间. DDAPS算法的求解时间明显优于A-ALNS算法和R-ALNS/I算法,这受益于基于数据挖掘的新解构造策略. 该策略通过挖掘高质量解中不同任务之间的次序关系(这些次序关系反映了问题的特点),根据所挖掘的次序关系构造新解,引导算法搜索方向,避免无效搜索,从而提升整个算法的求解速度.

**表7　算法运行时间对比结果　　单位**: s

| 场景ID | DDAPS | A-ALNS | R-ALNS/I |
|---|---|---|---|
| 1 | **161.23** | 320.64 | 325.10 |
| 2 | **261.72** | 549.71 | 399.69 |
| 3 | **348.87** | 780.36 | 386.34 |
| 4 | **228.32** | 508.82 | 254.19 |
| 5 | **385.96** | 696.54 | 521.63 |
| 6 | **282.31** | 508.07 | 489.23 |
| 7 | **471.22** | 892.90 | 509.89 |
| 8 | **409.56** | 737.89 | 534.13 |
| 9 | **337.65** | 546.67 | 390.22 |

### 3.3　DDAPS算法 vs Cplex

为进一步验证DDAPS算法的优越性,本文尝试将其与求解器Cplex进行比较,用Cplex求出的最优解作为上界去衡量DDAPS算法的求解效果. 然而,在上述设计的多星协同调度实验场景中,Cplex均无法在可接受的时间范围内求出问题的解. 为此,本文采用以下两种做法降低问题的求解难度:一是利用文献介绍的姿态角度拟合的方法,将多星协同调度问题建模成复杂度更低的混合整数规划模型[6]. 尽管该方法存在一些误差,但能够避免枚举全部的VTW. 二是降低任务数量与卫星规模,并且缩短调度周期,将调度周期从2013-04-20 T 00 : 00 : 00～23 : 59 : 59缩短至2013-04-20 T 00 : 00 : 00～12 : 00 : 00,从而减少卫星对目标的VTW数量. 对比结果如表8所示.

从表8的结果来看,DDAPS算法在小规模场景下能够取得与Cplex相同的最优结果. 随着任务数量和卫星规模的扩大,Cplex无法在可接受时间范围内求出问题的解,而DDAPS算法在中、大规模场景下依旧表现出优秀的求解效果.

**表8　小规模场景下DDAPS算法与Cplex对比实验结果**

| 类型 | 任务数量 | 卫星规模 | DDAPS/% | Cplex/% | 差距/% |
|---|---|---|---|---|---|
| 区域分布 | 25 | 2 | 93.84 | 93.84 | 0 |
| | 50 | 2 | 96.79 | 96.79 | 0 |
| | 75 | 2 | 93.98 | 93.98 | 0 |
| 全球分布 | 25 | 2 | 95.33 | 95.33 | 0 |
| | 50 | 2 | 96.17 | 96.17 | 0 |
| | 75 | 2 | 96.12 | 96.12 | 0 |

## 4　结　论

针对多星协同调度问题,本文提出了一种基于数据驱动的自适应并行搜索算法DDAPS,该算法采用的是“任务分配+单星调度”的双层调度框架,主要贡献有:

1) 提出了一种数据挖掘与元启发式算法结合的求解新思路. 该思路充分利用元启发式算法搜索过程中产生的中间数据,利用数据挖掘发现与问题相关的知识,基于所挖掘的知识构造新解,进一步引导元启发式算法后续更高效的搜索.

2) 成功将新算法应用于求解多星协同调度问题. 在任务分配层,算法设计了多个与问题特征相关的任务分配算子,能够快速产生质量较高的任务分配方案. 在单星调度层,为充分提升算法整体求解速度,本文以“问题”并行的方式同时求解多个单星任务调度问题,基于搜索过程中的数据,利用数据挖掘方法发现不同任务之间的次序关系,既保证了求解速度,也保证了求解质量. 实验结果表明,DDAPS算法能够有效解决多星协同调度问题,在小规模问题上甚至能够获得问题的最优解.

未来的研究可从以下两方面展开:

1) 尝试将该算法与更多的机器学习方法进行结合. 以算子选择为例,目前算法是依据贡献度去计算算子的选择概率,该方法虽然能够满足算法对自适应的要求,但是总体稍显粗糙. 下一步可以结合强化学习去研究不同算子的选择策略.

2) 从多个维度扩展多星协同调度问题. 例如,本文仅考虑了调度任务的总收益,未来可加入卫星负载均衡等其他目标,将问题从单目标优化问题扩展到多目标优化问题.

## 参考文献(References)

[1] Wu J, Chen Y N, He Y M, et al. Survey on autonomous task scheduling technology for earth observation satellites[J]. Journal of Systems Engineering and Electronics, 2022, 33(6): 1176-1189.

[2] Du Y H, Wang T, Xin B, et al. A data-driven parallel

scheduling approach for multiple agile earth observation satellites[J]. IEEE Transactions on Evolutionary Computation, 2020, 24(4): 679-693.

[3] Jang J, Choi J, Bae H J, et al. Image collection planning for Korea multi-purpose satellite-2[J]. European Journal of Operational Research, 2013, 230(1): 190-199.

[4] 杜永浩, 邢立宁, 姚锋, 等. 航天器任务调度模型、算法与通用求解技术综述[J]. 自动化学报, 2021, 47(12): 2715-2741.
(Du Y H, Xing L N, Yao F, et al. Survey on models, algorithms and general techniques for spacecraft mission scheduling[J]. Acta Automatica Sinica, 2021, 47(12): 2715-2741.)

[5] Wu J, Lu F, Zhang J W, et al. Design of task priority model and algorithm for imaging observation problem[J]. Journal of Systems Engineering and Electronics, 2020, 31(2): 321-334.

[6] He L, Liu X L, Laporte G, et al. An improved adaptive large neighborhood search algorithm for multiple agile satellites scheduling[J]. Computers & Operations Research, 2018, 100: 12-25.

[7] 杜永浩, 邢立宁, 陈盈果, 等. 卫星任务调度统一化建模与多策略协同求解方法[J]. 控制与决策, 2019, 34(9): 1847-1856.
(Du Y H, Xing L N, Chen Y G, et al. Unified modeling and multi-strategy collaborative optimization for satellite task scheduling[J]. Control and Decision, 2019, 34(9):1847-1856.)

[8] Peng G S, Dewil R, Verbeeck C, et al. Agile earth observation satellite scheduling: An orienteering problem with time-dependent profits and travel times[J]. Computers & Operations Research, 2019, 111: 84-98.

[9] Chen X Y, Reinelt G, Dai G M, et al. A mixed integer linear programming model for multi-satellite scheduling[J]. European Journal of Operational Research, 2019, 275(2): 694-707.

[10] Valicka C G, Garcia D, Staid A, et al. Mixed-integer programming models for optimal constellation scheduling given cloud cover uncertainty[J]. European Journal of Operational Research, 2019, 275(2): 431-445.

[11] 王沛, 谭跃进. 多星联合对地观测调度问题的列生成算法[J]. 系统工程理论与实践, 2011, 31(10): 1932-1939.
(Wang P, Tan Y J. Column generation for the earth observation satellites scheduling problem[J]. Systems Engineering — Theory & Practice, 2011, 31(10): 1932-1939.)

[12] Liu X L, Laporte G, Chen Y W, et al. An adaptive large neighborhood search metaheuristic for agile satellite scheduling with time-dependent transition time[J]. Computers & Operations Research, 2017, 86: 41-53.

[13] Zhang J W, Xing L N, Peng G S, et al. A large-scale multiobjective satellite data transmission scheduling algorithm based on SVM+NSGA-II[J]. Swarm and Evolutionary Computation, 2019, 50: 100560.

[14] Wu J, Song B Y, Zhang G T, et al. A data-driven improved genetic algorithm for agile earth observation satellite scheduling with time-dependent transition time[J]. Computers & Industrial Engineering, 2022, 174: 108823.

[15] Wu J, Yao F, Song Y J, et al. Frequent pattern-based parallel search approach for time-dependent agile earth observation satellite scheduling[J]. Information Sciences, 2023, 636: 118924.

[16] Anas S, Rumui N, Roy A, et al. Comparison of apriori algorithm and FP-growth in managing store transaction data[J]. International Journal of Computer and Information System: IJCIS, 2022, 3(4): 158-162.

## 作者简介

吴健(1994−), 男, 博士生, 主要研究方向为数据驱动优化算法和卫星任务调度, E-mail: jianwwwu@126.com;

姚锋(1978−), 男, 研究员, 博士, 主要研究方向为航天资源调度和军事人力资源调度, E-mail: yaofeng@nudt.edu.cn;

杜永浩(1993−), 男, 副研究员, 博士, 主要研究方向为大规模资源调度和智能优化算法设计, E-mail: duyonghao15@163.com;

陈宇宁(1988−), 男, 副教授, 博士, 主要研究方向为组合优化问题和机器学习算法, E-mail: 451480978@qq.com;

何磊(1991−), 男, 副教授, 博士, 主要研究方向为精确求解算法和不确定性优化, E-mail: helei@nudt.edu.cn;

何永明(1992−), 男, 副教授, 博士, 主要研究方向为强化学习理论及其在航天资源管控中的应用, E-mail: heyongming10@hotmail.com;

罗绥芝(1993−), 女, 讲师, 博士, 主要研究方向为数据驱动的决策支持, E-mail: suizhiluo@163.com.