基于混合蛙跳和遗传规划的跨单元调度方法

贾凌云 [1] 李冬妮 [1] 田云娜 [1, 2]

摘 要 针对运输能力受限条件下的跨单元问题, 提出了一种基于混合蛙跳与遗传规划的超启发式算法. 将改进的混合蛙跳算法作为超启发式算法的高层框架, 为跨单元调度问题搜索启发式规则, 同时利用遗传规划产生可以兼顾多因素的优质规则, 用于扩充超启发式算法的规则集. 实验表明, 提出的算法可以有效地搜索出优异的规则组合, 并且通过遗传规划产生的规则可以在很大程度上改善候选规则集, 提升算法性能.

关键词 跨单元调度, 跨单元运输, 超启发式算法, 混合蛙跳算法, 遗传规划

引用格式 贾凌云, 李冬妮, 田云娜. 基于混合蛙跳和遗传规划的跨单元调度方法. 自动化学报, 2015, **41**(5): 936−948

DOI 10.16383/j.aas.2015.c140455

An Intercell Scheduling Approach Using Shuffled Frog Leaping Algorithm and Genetic Programming

JIA Ling-Yun[1] LI Dong-Ni[1] TIAN Yun-Na[1, 2]

Abstract To deal with the intercell scheduling problem with limited transportation capabilities, a shuffled frog leaping algorithm-based hyperheuristic approach with genetic programming is developed in this paper. The proposed approach develops an improved shuffled frog leaping algorithm to search the combinations of heuristic rules for the addressed problem. Meanwhile, genetic programming is also introduced into the proposed algorithm to generate well-performing heuristic rules as an extension to the predefined candidate heuristic rules. Experimental results show that the improved shuffled frog leaping algorithm is efficient to search the outperforming combinations of the heuristic rules, and that the heuristic rules generated via genetic programming can obviously improve the quality of the candidate rule set, therefore providing better performance.

Key words Intercell scheduling, intercell transportation, hyperheuristic, shuffled frog leaping algorithm, genetic programming

Citation Jia Ling-Yun, Li Dong-Ni, Tian Yun-Na. An intercell scheduling approach using shuffled frog leaping algorithm and genetic programming. *Acta Automatica Sinica*, 2015, **41**(5): 936−948

单元制造系统 (Cellular manufacturing systems, CMS) 作为当今非常有效的生产方式之一, 在制造企业中广泛应用. 它是成组技术 (Group technology, GT) 在制造领域的典型应用, 体现了精益生产的哲理[1]. 在 CMS 中, 理想的情况下, 每个工件的加工都会被封装在一个单元内, 即把加工相似工件的机器划分到一个单元. 然而, 随着复杂产品的工艺越来越复杂, 单个单元的生产能力显得愈发有限, 导致工件的部分工序需要在其他单元的机器上进行, 跨单元调度问题由此产生.

早在上世纪 90 年代初, Garza 和 Smunt[2] 就指出由于跨单元转移难以避免, 理想的 CMS 将难以实施, 须定量分析跨单元转移对生产系统产生的影响, 但大部分研究一直集中在如何进行单元构造[3−4] 及如何进行单元内部管理[5−6] 等问题上. 直到近年来, 随着 CMS 的实施逐渐深入并遇到困难, 跨单元调度这一问题才开始被关注, 相关研究可以分为跨流水单元[7−10] 和跨作业单元两种类型[11−15].

Solimanpur 和 Elmi[7]、Li 和 Murata[11]、Meng 等[12]、Pajoutan 等[13]、李冬妮等[14−15] 均考虑了跨单元转移时间, Golmohammadi 和 Ghodsi[8]、Mosbah 和 Dao[9]、Gholipour 等[10] 忽略了跨单元转移时间. 然而, 上述研究中都隐含着一个假设, 即假设单元间具有充足的运输能力, 因此工件在一个单元内完成相应的工序后, 如果下一道工序需要在其他单元的机器上加工, 经过固定的转移时间后可以到达下一单元.

收稿日期 2014-06-23 录用日期 2014-11-17
Manuscript received June 23, 2014; accepted November 17, 2014
国家自然科学基金 (71401014), 北京市自然科学基金 (4122069) 资助
Supported by National Natural Science Foundation of China (71401014) and Natural Science Foundation of Beijing (4122069)
本文责任编委 赵千川
Recommended by Associate Editor ZHAO Qian-Chuan
1. 北京理工大学计算机学院智能信息技术北京市重点实验室 北京 100081 2. 延安大学数学与计算机科学学院 延安 716000
1. Beijing Key Laboratory of Intelligent Information Technology, School of Computer Science, Beijing Institute of Technology, Beijing 100081 2. College of Mathematics and Computer Science, Yan′an University, Yan′an 716000

但事实上这一假设很难得到满足. 在实际的生产过程中, 单元与单元之间的距离可能很远, 需要借助运输工具进行跨单元运输, 而运输工具作为一种资源, 数量和容量都是有限的, 也就是说, 当运输工具不可用时, 工件必须等待.

基于上述分析, 本文考虑运输能力受限的跨单元调度问题, 以最小化总加权延迟为目标, 提出了一种基于混合蛙跳算法 (Shuffled frog leaping algorithm, SFLA) 与遗传规划 (Genetic programming, GP) 的超启发式算法 (SFLA-based hyperheuristic approach with GP, SHAG). SHAG 算法的基本流程可以描述为: 1) 利用 GP 生成规则, 扩充备选规则集; 2) 用改进的 SFLA 搜索备选规则集中的规则; 3) 引入时间窗的概念, 利用 GP 生成合适的规则以计算每个批次等待的时间窗.

1　问题描述

本文考虑的问题模型由多个生产单元组成. 其中每个单元均有一台运输工具, 负责将本单元的需要跨单元转移的工件运送至其他单元.

1.1　问题假设

本文考虑的运输能力受限的跨单元调度问题基于以下假设:

1) 所有工件在零时刻释放.

2) 每台机器同一时刻只能处理一个工件.

3) 工件具有多道工序, 至少需要在两个不同的单元里完成.

4) 由于机器的加工能力重叠, 工件存在柔性路径. 但对于任一工序, 在同一单元内至多存在一台可加工的机器.

5) 考虑跨单元转移, 忽略工件在单元内的转移.

6) 所有运输工具均是一致的, 每个单元有且仅有一辆运输工具, 且运输工具存在容量限制.

7) 运输工具一次运输由多个工件组成的一个批次, 同一批次中各个工件的目的单元可能不同. 运输工具按照一定顺序访问各个目的单元并卸载对应的工件.

8) 运输工具仅在其所属单元装载工件, 从本单元出发后不在沿途各单元装载工件, 运输工具仅在工件被运送至目的单元后才能卸载该工件. 装载和卸载工件的时间忽略不计.

9) 一旦所有工件均完成运输, 运输工具立刻返回其所属单元.

10) 其他条件, 例如准备时间、机器抢占、机器损坏等不在本文考虑范围内.

1.2　符号列表

问题模型中用到的符号定义如下.

1) 索引

$i = 1, \cdots, N$: 工件索引
$j = 1, \cdots, o_i$: 工件 i 的工序索引
$c = 1, \cdots, C$: 单元及其运输工具索引
$m = 1, \cdots, M$: 机器索引
$b = 1, \cdots, B_c$: 运输工具 c 上的批次索引
$t = 1, \cdots, T$: 时间

2) 系统变量

$o_{i,j}$: 工件 i 的第 j 道工序
o_i: 工件 i 的工序总数
$p_{i,j,m}$: $o_{i,j}$ 在机器 m 上的加工时间 ($p_{i,j,m} = 0$ 表示 $o_{i,j}$ 不能在机器 m 上加工)
sz_i: 工件 i 的体积
w_i: 工件 i 的交货期
v: 运输工具容量
$d_{c,c'}$: 单元 c 和 c' 之间所需转移时间
$p_{i,j}$: $o_{i,j}$ 实际加工时间
$f_{i,j}$: $o_{i,j}$ 的完工时间

$$l_{m,c} = \begin{cases} 1, & \text{机器 } m \text{ 位于单元 } c \\ 0, & \text{其他} \end{cases}$$

3) 决策变量

$t_{i,j}$: $o_{i,j}$ 完成后所花费的转移时间
$dc_{i,j}$: $o_{i,j}$ 完成后转移的目的单元
$s_{i,j}$: $o_{i,j}$ 开始时间

$$e_{i,j} = \begin{cases} 1, & o_{i,j+1} \text{ 和 } o_{i,j} \text{ 被分派至不同单元} \\ 0, & \text{其他} \end{cases}$$

$$x_{i,j,m} = \begin{cases} 1, & o_{i,j} \text{ 被分派至机器 } m \\ 0, & \text{其他} \end{cases}$$

$$y_{i,j,t} = \begin{cases} 1, & o_{i,j} \text{ 开始加工时刻为 } t \\ 0, & \text{其他} \end{cases}$$

$$z_{c,b,t} = \begin{cases} 1, & \text{小车 } c \text{ 的批次 } b \text{ 在 } t \text{ 时刻开始运输} \\ 0, & \text{其他} \end{cases}$$

$$u_{i,j,c,b,q} = \begin{cases} 1, & o_{i,j} \text{ 完工后被分派至小车 } c \text{ 的} \\ & \text{批次 } b \text{ 的第 } q \text{ 个进行运输} \\ 0, & \text{其他} \end{cases}$$

1.3　目标函数及约束条件

本文所考虑的问题模型的调度目标为总加权延时, 如式 (1) 所示.

$$Minimize\ TWT \tag{1}$$

其中, TWT 为总加权延时, 且 $TWT = \sum_{i=0}^{N} w_i \times \max\{f_i - d_i, 0\}$.

问题约束如下:

$$\sum_{m=1}^{M} x_{i,j,m} = 1,\ \forall i,j \tag{2}$$

$$\sum_{m=1}^{M} x_{i,j,m} p_{i,j,m} > 0,\ \forall i,j \tag{3}$$

$$p_{i,j} = \sum_{m=1}^{M} x_{i,j,m} p_{i,j,m},\ \forall i,j \tag{4}$$

$$\sum_{t=1}^{T} y_{i,j,t} = 1,\ \forall i,j \tag{5}$$

$$s_{i,j} = \sum_{t=1}^{T} y_{i,j,t} t,\ \forall i,j \tag{6}$$

$$f_{i,j} = s_{i,j} + p_{i,j},\ \forall i,j \tag{7}$$

$$x_{i,j,m} y_{i,j,t} \sum_{i'=1}^{N} \sum_{j'=1}^{O_i} \sum_{t'=t}^{t+p_{i,j}-1} x_{i',j',m} y_{i',j',t'} = 0,\ \forall i,j,t,m \tag{8}$$

$$\sum_{c=1}^{C} \sum_{b=1}^{B} \sum_{t=1}^{T} z_{c,b,t} = 1 \tag{9}$$

$$\sum_{c=1}^{C} \sum_{b=1}^{B} \sum_{q=1}^{Q} u_{i,j,c,b,q} = e_{i,j},\ \forall i,j \tag{10}$$

$$\sum_{b=1}^{B} \sum_{q=1}^{Q} u_{i,j,c,b,q} = \sum_{m=1}^{M} x_{i,j,m} l_{m,c},\ \forall i,j,c,e = 1 \tag{11}$$

$$\sum_{i=1}^{N} \sum_{j=1}^{O_i} \sum_{q=1}^{Q} u_{i,j,c,b,q} s z_i \le v,\ \forall c,b \tag{12}$$

$$u_{i,j,c,b,q} u_{i',j',c,b,q} dc_{i,j} = u_{i,j,c,b,q} u_{i',j',c,b,q} dc_{i',j'},\ \forall i,j,i',j',c,b,q \tag{13}$$

$$dc_{i,j} = \sum_{m=1}^{M} l_{m,c} x_{i,j,m} c,\ \forall i,j \tag{14}$$

$$t_{i,j} \ge u_{i,j,c,b,q+1} u_{i',j',c,b,q} (t_{i'j'} + d_{dc_{i,j},dc_{i',j'}}),\ \forall i,j,i',j',c,b,q \tag{15}$$

$$t_{i,j} \ge u_{i,j,c,b,q} d_{dc_{i,j},c},\ \forall i,j,c,b \tag{16}$$

$$\sum_{t=1}^{T} z_{c,b,t} t \ge u_{i,j,c,b,q} f_{i,j},\ \forall i,j,c,b,q \tag{17}$$

$$s_{i,j+1} \ge \max\left\{ f_{i,j}, e_{i,j} \times \left(\sum_{c=1}^{C} \sum_{b=1}^{B} \sum_{q=1}^{Q} \sum_{t=1}^{T} z_{c,b,t} t + t_{i,j} \right) \right\},\ \forall i,j \tag{18}$$

$$\sum_{t=1}^{T} z_{c,b+1,t} t \ge \sum_{t=1}^{T} z_{c,b,t} t + t_{i,j} + u_{i,j,c,b,q} d_{dc_{i,j},c},\ \forall i,j,c,b,q \tag{19}$$

约束 (2) 和 (3) 表示每道工序仅能被分派至一台机器且该机器可以加工该工序; 约束 (4) 给出了工序实际加工时间的定义; 约束 (5) 表示每道工序都需加工且只能加工一次; 约束 (6) 和 (7) 分别给出了工序开始时间和完工时间的定义; 约束 (8) 表明一台机器同一时间仅能加工一道工序; 约束 (9) 表示每个批次均需被运输且只能被运输一次; 约束 (10) 表明每道工序完工后仅能被分派至一个运输工具上的一个批次; 约束 (11) 表示每道工序完工后只能被所在单元的运输工具运输; 约束 (12) 说明同一批次内工件的体积总和不得超过运输工具容量; 约束 (13) 表示仅有目的单元相同的工件可以被同时在同一单元卸载; 约束 (14) 给出了一次运输的目的单元的定义; 约束 (15) 表示任一工件在运输工具完成当前批次中排在前面所有的工件的运输、且到达该工件目的单元之后才能被卸载 (批次中的首个工件除外); 约束 (16) 表明任意批次的首个工件只有在运输工具到达其目的单元后才能被卸载; 约束 (17) 说明同一批次中所有工件均完成前一道工序的加工后才能开始运输; 约束 (18) 表明任意工序只有在其前一道工序完成加工, 并被运输至对应单元 (如有跨单元转移发生) 后才能开始加工; 约束 (19) 表示运输工具运输完一个批次中所有工件并返回起始单元后才能开始下一批次的运输.

2 算法设计

SFLA 模拟青蛙寻找食物的过程, 是一种新型群智能进化算法. 它是由 Eusuff 和 Lansey[16] 于 2003 年首次提出, 并应用在求解水资源网络的管径选择和管网扩张问题中. Xu 等[17] 将混合蛙跳算法应用在多目标混合流水车间调度问题上. 基于上述成功应用, 本文提出一种基于 SFLA 与 GP 的 SHAG 算法, 解决运输能力受限的跨单元调度问题.

2.1 SHAG 算法概述

SHAG 算法将 SFLA 应用在搜索启发式规则上, 其中, 启发式规则集不仅包含预定义的、生产中常用的规则, 还包含利用 GP 产生的一些优质规则. GP 产生规则的过程可以进行离线计算, 实际计算时只需 SFLA 算法进行规则选择的计算, 这样不仅大大提高了计算效率, 也保证了解的质量.

本文提出运输能力受限的跨单元调度问题可以分为四个子问题, 即工序分派子问题、工序排序子问题、运输组批子问题和路径决策子问题.

1) 工序分派是指每道工序都在可选择加工的机器里找到一台机器进行加工.

2) 工序排序是指为每台机器的缓冲区里的工件进行排序, 以确定机器下一个加工的工件.

3) 运输组批是指决策运输工具下一批次中要运送哪些工件. SHAG 算法在运输工具空闲时通过时间窗规则得到一个时间窗的值, 并将在时间窗内所有可运输的工件按顺序装载入运输工具直至运输工具满载, 来完成运输组批的决策.

4) 路径决策是指为每个单元里需要运送到其他单元进行加工的工件进行排序, 以确定运输工具运送工件的先后顺序.

因此, SHAG 分为两个步骤, 首先是针对四个子问题分别利用 GP 产生优质规则, 与预设的调度规则共同组成候选规则集, 而后再利用 SFLA 算法在候选规则集中搜索出优秀的规则组合, 即为每个工件指定一个分派规则、为每台机器指定一个排序规则、为每台运输工具指定一个路径决策规则和时间窗规则, 进而根据求得的规则组合产生调度解.

2.2 GP 算法与规则生成

2.2.1 GP 算法的整体流程

实际生产中常见的简单规则通常只考虑到了工件、机器或者运输工具的单一因素 (例如工件交货期、机器加工时间、机器负载等), 而本文考虑的问题模型较为复杂, 每一时刻工件、机器或者运输工具都存在多个不同的属性. 为了充分利用工件、机器、运输工具的状态信息, 使得包含这些信息的规则更有价值, 产生高质量的调度规则, 本文利用 GP 生成启发式规则.

本文的 GP 算法是基于 ECJ20 所开发, ECJ (Evolutionary computation journal) 是一个关于进化计算的开源库[18]. 其中包含初始化、交叉变异、适应度评价等操作, 其程序框图如图 1 所示.

2.2.2 规则表示

本文 GP 算法中, 规则的表示采用二叉树状数据结构, 这样可以更方便地产生对各种输入因素进行各种操作的规则. 树的叶子节点表示工件、机器或者运输工具的某种属性, 非叶子结点表示对属性进行的操作, 有 +、−、×、/ 等.

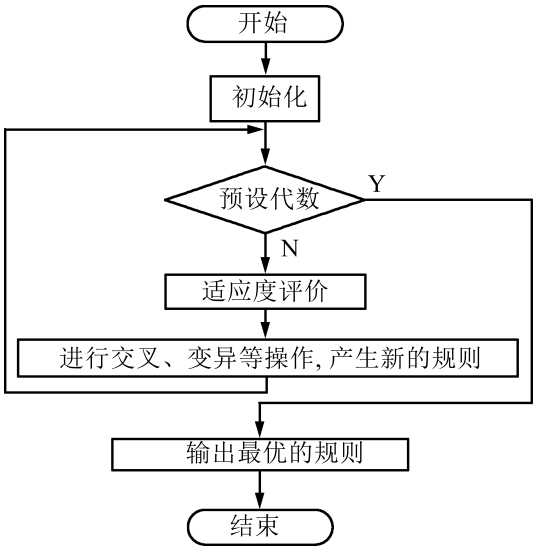


图 1　GP 算法流程图

Fig. 1　Algorithm flowchart of GP

例如, 图 2 表示的式 (20) 就是一个路径决策的启发式规则. 运输工具在决定运输工件的先后顺序时, 根据式 (20) 代入各个工件的属性值会得到各个工件对应的函数值. 这个值可以看成是工件需要运输的紧迫程度, 运输工具根据每个工件对应的函数值大小来决定运输的先后顺序. 值越大, 工件运输的优先级越高.

$$\frac{W}{PT+DD}+AT \tag{20}$$

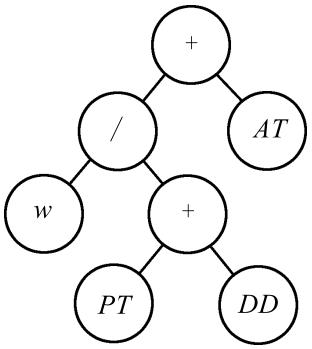


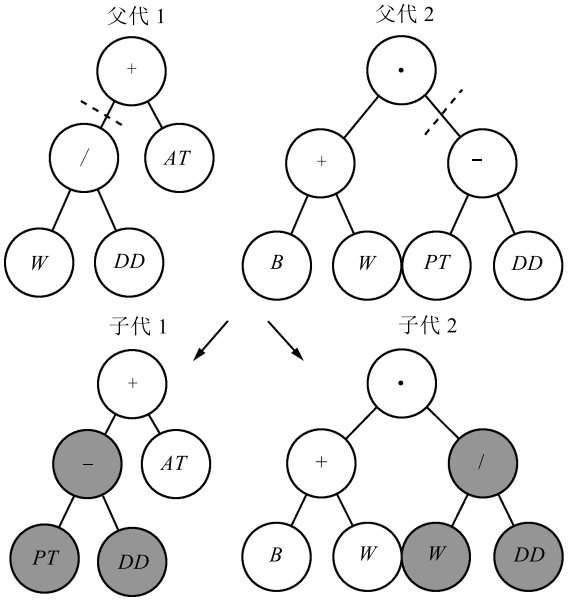
图 2　规则表示

Fig. 2　Representation of the rule

在初始化构建种群时, 随机选择端点集中的元素作为叶子节点, 函数集中的元素作为非叶子节点, 按此规则生成如图 2 中表示规则的二叉树.

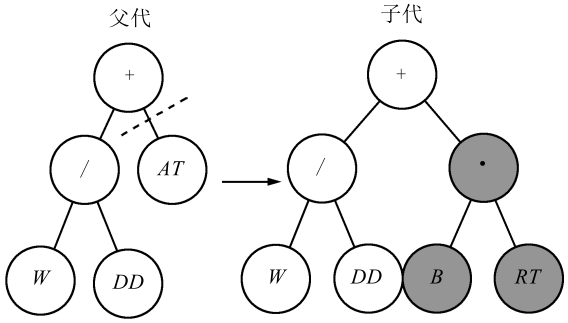
2.2.3 GP 的基本操作

GP 通过交叉、变异等操作产生新的规则. 操作均应满足语法约束, 例如除数不能为 0. 对于交叉操作, GP 算法使用子树交叉[19], 通过选中两个较优质的个体, 组合它们的子树来产生下一代新的个体; 对于变异操作, GP 算法使用子树变异[19], 即随机选择一个个体的节点, 再随机选择另一个个体的子树, 用此子树替代以原个体的节点为根节点的子树. 这两个操作实例如图 3 所示.



(a) 子树交叉

(a) Crossover



(b) 子树变异

(b) Mutation

图 3 交叉变异图

Fig. 3 The graph of crossover and mutation

2.2.4 GP 参数设置

本文 GP 的参数设置如表 1 所示, 设置参考了第 2.2.1 节中提到的开源库 ECJ20[18] 中 GP 的参数设置, 已经有学者在相关算法上使用了 ECJ20 的标准参数[20].

表 1 GP 参数

Table 1 Parameters of GP

参数	描述
种群大小	1 024
交叉概率	90 %
变异概率	10 %
进化代数	51
选择算子	锦标赛 (大小为 17)
初始化	Ramped half-and-half
交叉最大深度	17
端点集	表 2, 表 3, 表 4, 表 5
函数集	+, −, ×, /, !
适应度	见第 2.2.5 节

表 1 中 GP 的初始化为 Ramped half-and-half[19], 表示初始时有一半的二叉树采用全深度初始化, 即所有二叉树都具有相同的最大设定深度, 另一半采用增长深度初始化, 即二叉树的深度可以不相同, 但是不能超过最大设定深度.

2.2.5 适应度函数

由于本文的目标函数值是总加权延迟, 不同规则计算出来的函数值总是差异较大, 不适合直接用作 GP 的适应度, 因此, 引入一个计算适应度的函数. 如式 (21) 所示.

$$dex(\lambda, Ins_i) = \frac{obj(\lambda, Ins_i) - Ref(Ins_i)}{Ref(Ins_i)} \quad (21)$$

其中, $obj(\lambda, Ins_i)$ 表示对于实例 Ins_i 应用规则 λ 计算出来的目标函数值, 是实例 $Ref(Ins_i)$ 的参考目标函数值. 本文采用第 3.2 节参数分析实验所求得的解作为该参考目标函数值.

2.2.6 GP 的端点集

端点集就是上文提到的表示 GP 规则的二叉树的叶子节点的集合, 它表示参与产生 GP 规则的工件、机器或者运输工具属性的集合, 因此在很大程度上决定了 GP 所产生规则的优劣. 由于本文问题模型分为四个子问题, 这里了列出四个子问题的端点集.

工序分派、工序排序、路径决策、运输工具组批 (时间窗) 的端点集分别如表 2～5 所示.

表 2、表 3、表 5 作为端点集时, 可生成的规则的例子分别如图 4、图 5、图 6 所示, 表 4 所对应的例子如图 2 所示.

表 2　工序分派的端点集

Table 2　Terminal set of part assignment

标识	描述
BN	当前机器缓冲区工件个数
DD	工件工期
PT	机器的加工时间
TT	所需要的运输时间
VR	运输工具返回的时间
WT	可能需要等待的时间
$\#$	服从 $[0,1]$ 均匀分布的常量

表 3　工序排序的端点集

Table 3　Terminal set of part sequencing

标识	描述
ROP	工件剩余工序数
DD	工件工期
PT	机器的加工时间
RT	工件的剩余加工时间
WT	工件已经等待的时间
W	工件的权重
$\#$	服从 $[0,1]$ 均匀分布的常量

表 4　路径决策的端点集

Table 4　Terminal set of vehicle routing

标识	描述
AT	工件到达缓冲区的时间
DD	工件工期
PT	工件下一道工序需要加工的时间
ROP	工件剩余工序数
TT	所需要的运输时间
WT	工件等待的时间
W	工件权重
$\#$	服从 $[0,1]$ 均匀分布的常量

2.3　SFLA 与规则选择

2.3.1　SFLA 的整体流程

本文对基本混合蛙跳算法进行了改进, 按相似度划分种群以加快算法的收敛速度, 同时对局部最优和全局最优进行随机扰动和 Path-relinking, 在一定程度上避免了算法陷入局部最优解.

表 5　运输工具组批的端点集

Table 5　Terminal set of batch formation for vehicles

标识	描述
BN	当前缓冲区工件数量
CN	当前运输工具已经装载的工件数量
$CSDD$	当前运输工具已经装载的工件总工期
$CSRT$	当前运输工具已经装载的工件总剩余加工时间
CSW	当前运输工具已经装载的工件总权重
NDD	下一个工件的工期
$NMRT$	单元里可以到达的最后一个工件的时间
NRT	下一个到达的工件的时间
$NReT$	下一个到达的工件的剩余加工时间
NW	下一个到达的工件的权重
SDD	当前缓冲区工件总工期
SRT	当前缓冲区工件总剩余加工时间
SW	当前缓冲区工件总权重
$\#$	服从 $[0,1]$ 均匀分布的常量

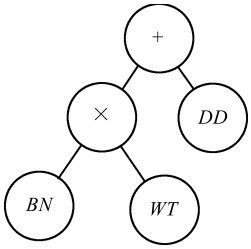


图 4　工序分派的规则举例

Fig. 4　Example of heuristic rules for part assignment

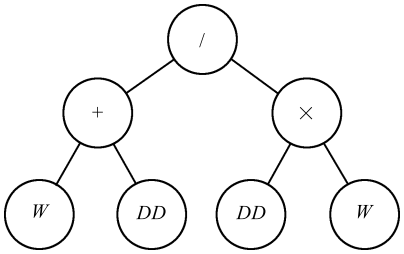


图 5　工序排序的规则举例

Fig. 5　Example of heuristic rules for part sequencing

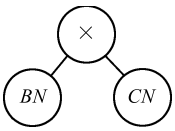


图 6　运输工具组批的规则举例

Fig. 6　Example of heuristic rules for batch formation for vehicles

算法的具体流程为:

步骤 1. 初始化参数.

步骤 2. 初始化青蛙, 计算每个青蛙的适应度.

步骤 3. 划分种群.

步骤 3.1 随机找寻一个解, 计算其他解与它的相似度, 按相似度划分种群;

步骤 3.2 重复步骤 3.1, 直到种群划分结束.

步骤 4. 更新全局最优和局部最优、局部次优和局部最差解.

步骤 4.1 对全局最优和局部最优进行随机扰动;

步骤 4.2 对操作后的解计算适应度, 如果有更优解出现, 则进行更新;

步骤 4.3 重复步骤 4.1 和步骤 4.2, 直至达到预设循环次数.

步骤 5. 对于局部最差解进行替换.

步骤 5.1 利用更新策略对局部最差解进行替换;

步骤 5.2 如果达到迭代次数则停止, 转步骤 6; 如果没有, 则转步骤 3.

步骤 6. Path-relinking.

步骤 6.1 对所有局部最优解向全局最优解进行 Path-relinking;

步骤 6.2 计算每个青蛙的适应度.

步骤 7. 输出全局最优解.

算法流程图如图 7 所示.

2.3.2 编码、解码和规则

1) 编码和解码

由于本文采用的是超启发式算法, 算法搜索到的对象是规则的组合而不是调度解本身, 因此针对问题模型, 本文设计的编码如图 8 所示. 编码分为四段, 其中工序分派段为每道工序指定一个分派规则; 工序排序段为每台机器指定一个排序规则; 路径决策段为每个单元出单元缓冲区指定一个排序规则; 运输组批段为每个单元的运输工具指定一个时间窗规则 (固定的时间窗或利用 GP 生成的规则进行计算得出的时间窗). 编码上的各位均以规则的序号表示指定的规则.

由于启发式规则所包含的因素对于环境的依赖很大, 很难找到一种在任何环境下都具有很好性能的启发式规则[21]. 因此, 许多学者采取组合规则 (Combinatorial rules) 进行调度, 即为不同的实体 (机器、工件等) 使用不同的规则. 研究表明, 组合规则比单一规则具有更好的性能[22−24]. 近年来, 有学者开始使用超启发式算法选择调度规则. Li 等[25] 针对混合流水车间调度问题, 采用超启发式算法为不同的机器选择不同的规则. Yang 等[26] 采用超启发式方法为不同的机器选择相同的规则. 实验结果表明, Li 等的方法与 Yang 等的方法相比具有更好的调度性能. 基于此, 本文扩展了 Li 等和 Yang 等的算法, 在选择规则之前加入生成规则的步骤. 同时, 为不同实体 (包括机器、工件和运输工具) 分配不同的规则.

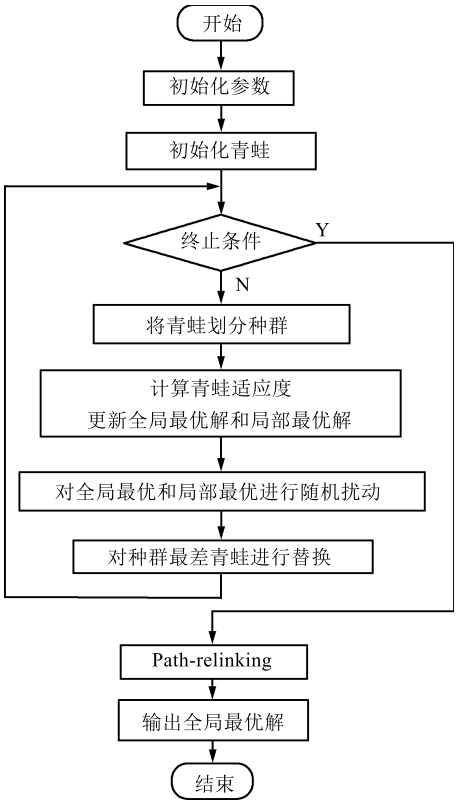


图 7 SHAG 算法流程图

Fig. 7 General algorithm of SHAG

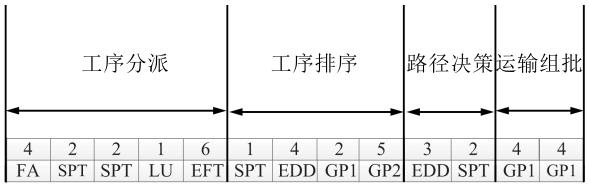


图 8 编码图例

Fig. 8 Example of encoding

图 8 中, 每一部分的第 i 位代表第 i 个工序或者机器或者运输工具进行调度时所用的规则. 例如, 第 7 位表示第 2 个机器按照 EDD 规则来选择下一个加工的工件. 因此, 按照这些规则进行调度, 即可形成一个完整的调度解.

2) 规则

超启发式算法提供了一种高层启发式方法, 通过操纵和调用一系列底层启发式规则来实现对于问题的求解. 本文算法所采用的底层启发式规则, 除了简单的启发式规则还包含了利用 GP 产生的启发式规则. 算法所采用到的启发式规则如下:

工序分派的规则:

a) EFT (Earliest finish time): 选择加工完成的时间最早的机器;

b) FA (First available): 选择最早可进行加工的机器;

c) SPT (Shortest processing time): 选择加工时间最短的机器;

d) MA (Most available): 选择缓冲区中工件个数最少的机器;

e) GP1: 利用 GP 产生的适应度第 1 的工序分派规则;

f) GP2: 利用 GP 产生的适应度第 2 的工序分派规则;

g) GP3: 利用 GP 产生的适应度第 3 的工序分派规则.

工序排序的规则:

a) SRPT (Shortest remaining processing time): 选择剩余加工时间最短的工件;

b) SPT (Shortest processing time): 选择加工时间最短的工件;

c) WEDD (Wighted earliest due date): 选择具有最小加权工期的工件;

d) WSPT (Wighted shortest processing time): 选择具有最小加权加工时间的工件;

e) WT (Waited time): 选择等待时间最长的工件;

f) SPTR (Smallest processing time ratio): 选择具有最小加工时间比率的工件;

g) EDD (Earliest due date): 选择工期最早的工件;

h) GP1: 利用 GP 产生的适应度第 1 的工序排序规则;

i) GP2: 利用 GP 产生的适应度第 2 的工序排序规则;

j) GP3: 利用 GP 产生的适应度第 3 的工序排序规则.

路径决策的规则:

a) EDD (Earliest due date): 选择工期最早的工件;

b) WEDD (Wighted earliest due date): 选择具有最小加权工期的工件;

c) SPT (Shortest processing time): 选择下一道工序加工时间最短的工件 (Wighted shortest processing time, WSPT): 选择具有最小加权加工时间;

d) WT (Waited time): 选择等待时间最长的工件;

e) SRPT (Shortest remaining processing time): 选择剩余加工时间最短的工件;

f) SPTR (Smallest processing time ratio): 选择加工时间比率最小的工件;

g) GP1: 利用 GP 产生的适应度第 1 的运输工具组批规则;

h) GP2: 利用 GP 产生的适应度第 2 的运输工具组批规则;

i) GP3: 利用 GP 产生的适应度第 3 的运输工具组批规则.

运输工具组批 (时间窗) 的规则:

a) 0: 运输工具不需要等待直接进行运输;

b) 20: 运输工具等待时间 20 后进行运输;

c) 40: 运输工具等待时间 40 后进行运输;

d) GP1: 利用 GP 产生的适应度第 1 的时间窗规则;

e) GP2: 利用 GP 产生的适应度第 2 的时间窗规则;

f) GP3: 利用 GP 产生的适应度第 3 的时间窗规则.

2.3.3　初始化和种群划分

算法开始时随机初始化总数量的青蛙, 即随机生成总数量的具有固定长度的一组数字, 每个数字都代表一种规则.

在常见的 SFLA 算法中[27−29], 青蛙按适应度进行排序, 以特定的划分原则进行种群划分. 例如种群数为 3, 适应度排在第 1 位的青蛙被分在第 1 个种群, 第 2 位的被分在第 2 个种群, 第 3 位的被分在第 3 个种群, 第 4 位的青蛙被分在第 1 个种群. 以此类推, 将所有的青蛙按照此种原则划分到 3 个种群中.

受 Teekeng 和 Thammano[30] 提出的、根据操作对的相似度形成最相近的解对的思想的启发, 本文提出改进的 SFLA 算法, 利用青蛙的相似度进行种群划分. 本文的相似度的具体算法为对比两个解每一个位置上的数字大小, 如果相同, 则相似度加 1. 如图 9 所示的两个解的相似度为 3.

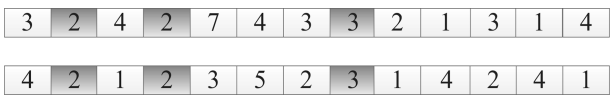


图 9　相似度图例

Fig. 9　Example of the similarity of two solutions

划分种群的具体算法为:

步骤 1. 随机找到一个解, 计算该解与其他解的相似度, 按相似度从高到低选取 n 个解, 划分成一个种群, 其中 $n =$ 青蛙总数/种群个数.

步骤 2. 在剩余解中随机找寻一个解, 利用步骤 1 的方法进行种群划分, 直至种群划分结束.

2.3.4 初始化和种群划分

在 SFLA 算法中, 每个青蛙种群内都进行信息交换, 即算法在不同搜索方向上进行搜索. 由于本文提出的改进后的 SFLA 算法利用相似度对种群进行了划分, 为了增强在不同搜索方向上的搜索能力, 算法采用了定向对种群内最差青蛙进行改变. 许多研究在对种群内进行更新时, 利用两个较优质的解对最差解进行更新[27, 30−31]. 因此, 本文的具体算法为对比每个种群内局部最优和局部次优解, 如果相同位置上有相同的规则, 则对该位置进行记录. 由于该位置上的规则很可能是有利于增加适应度的, 即有利于调度的, 所以对种群内最差解的该位置上的规则进行替换. 该操作如图 10 所示.

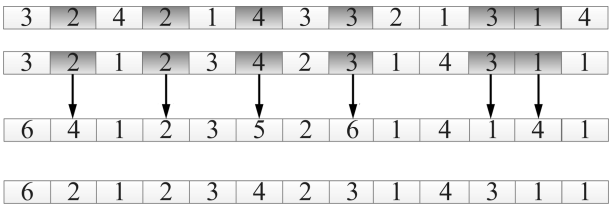


图 10 替换图例

Fig. 10 Example of offspring generation

对替换后的解进行评价, 如果适应度反而小于替换前的解, 那么就随机生成一个解代替局部最差解.

2.3.5 随机扰动和 Path-relinking

为了避免算法陷入局部最优解, 改进的 SFLA 算法在基本混合蛙跳算法上增加了两种策略, 即随机扰动和 Path-relinking.

随机扰动是对全局最优解和局部最优解随机位置上的规则进行随机替换. 这个过程进行若干次, 每次都对新产生的解计算适应度, 如果有更优的解出现, 则用其替换当前解.

Path-relinking 由 Glover 等[32] 提出, 是一种可以增强搜索强度和增加搜索多样性的方法, 它通过探查连接两个高质量解的路径去生成新的解, 也可以看成是将两个高质量解的优良属性进行混合.

本文中的 Path-relinking 过程, 把全局最优解作为导向解, 各个局部最优解作为初始解. 通过初始解向导向解的改变, 来寻求更优解. 这里使用一个候选解集存放此过程中产生的解. 此过程具体为:

步骤 1. 选出一个局部最优解, 对比自己与全局最优青蛙的每个位置上的规则, 如果有不同则记录该位置.

步骤 2. 在候选解记录的位置中, 随机选取若干个位置替换成导向解该位置上的规则, 计算解的适应度.

步骤 3. 重复步骤 2, 直到局部最优解与全局最优解完全相同, 将该过程中产生的适应度最高的解放入候选解集中.

步骤 4. 再选出一个局部最优解重复步骤, 直到所有局部最优解都被选择, 转步骤 5.

步骤 5. 对比候选解集中的解与全局最优解, 如果有更优解则更新.

3 实验与分析

3.1 实验设计

本文仿真实验运行在 2.2 GHz CPU, 2 GB RAM 的 PC 机上.

实验采用了 12 种规模不同的测试用例, 工件数量在 $[4, 80]$ 之间随机产生, 机器数量在 $[17, 25]$ 之间随机产生, 单元数量在 $[4, 6]$ 之间随机产生. 每个工件的工序数服从 5 ∼ 19 之间离散均匀分布, 单元之间的转移时间服从 6 ∼ 50 之间离散均匀分布, 运输工具的容量为 20, 每个工件的权重从 1 ∼ 12 之间产生. 每个工件的工期用式 (22) 计算.

$$d_j = dl \sum_{k=1}^{K} p_{jk} \tag{22}$$

其中, dl 是交货期因子, 表示交货期的紧张程度, 在下文的实验中默认 1. $\sum_{k=1}^{K} p_{jk}$ 表示工件 j 在工序 k 所有可选机器上的最小加工时间的总和.

不同问题模型下的机器在单元内的分布如表 6 所示.

由于本文的问题来源于生产实际, 是比较新的问题, 难以找到经典算例或针对相同问题的已有方法. 上述的 12 种测试用例是作者根据本文问题模型而设计. 由于缺少 benchmark, 为了验证算法性能, 本文中增加了与 CPLEX 的对比. 不但对于同时含有生产和运输的原问题进行了实验, 还针对其中的机器调度维度和运输调度维度分别与 CPLEX 进行了比较实验. 实验结果表明, 本文算法具有较好的性能和较高的计算效率.

3.2 参数分析

本文提出的 SHAG 算法的核心部分是改进的 SFLA 算法, 算法中有很多参数可能对性能造成影响, 所以本文先把所有可能因素作为参数做初步方差分析 (Analysis of variance, ANOVA). 再对其中对算法性能有影响的参数进行准确的方差分析, 获得最优的算法参数.

首先, 考虑青蛙总数 (Population)、种群数量 (Group)、局部搜索次数 (Localtime)、局部搜索变

表 6　机器在单元内的分布表

Table 6　Information about machines located in cells

工件 j/机器 m/单元 c	单元 1	单元 2	单元 3	单元 4	单元 5	单元 6
$j4/m17/c4$	3	4	5	5	–	–
$j8/m17/c4$	3	4	5	5	–	–
$j11/m21/c5$	4	5	4	3	5	–
$j20/m21/c5$	4	5	4	3	5	–
$j28/m21/c5$	4	5	4	3	5	–
$j36/m21/c5$	4	5	4	3	5	–
$j40/m25/c6$	2	5	4	4	4	5
$j48/m25/c6$	2	6	4	4	4	5
$j56/m25/c6$	2	6	4	4	4	5
$j64/m25/c6$	2	6	4	4	4	5
$j72/m25/c6$	2	6	4	4	4	5
$j80/m25/c6$	2	6	4	4	4	5

异率 (Localratio)、Path-relinking 变异率 (Pathratio)、迭代次数 (Maxloop) 这 6 个算法参数, 进行一个初步的具有 2 个水平值的 6 个参数的方差分析实验. 从实验中看出青蛙总数和局部搜索次数这两个参数对算法性能几乎没有影响 ($P > 0.05$). 所以在以后的实验中, 分别取定值 100 和 5.

随后, 考虑种群数量、局部搜索变异率、Path-relinking 变异率、迭代次数这 4 个算法参数, 设计了 4 个参数 3 个水平值的全析因子实验.

根据分析, 为了最小化目标函数 TWT (Total weight tardness), 各个参数的取值分别为: 迭代次数 Maxloop = 200, 种群数量 Group = 20, 局部变异率 Localratio = 0.25, Path-relinking 变异率 Pathratio = 0.01.

3.3　性能对比

为了验证算法的有效性, 本文实现了两个对比算法, 与本文算法进行对比实验, 参与性能对比的三个算法如下:

1) GBA (GP-based approach): 应用第 2.3.2 节所描述的规则, 包括常见的规则和 GP 生成的规则, 为本文的四个子问题 (即工序分派、机器排序、运输工具组批和时间窗) 随机分配规则, 运行若干次, 挑选出最优解. 这里为了方便对比, 迭代次数设置与 SHAG 相同, 即 200 次.

2) SBA (SFLA-based approach): 由本文提出改进的 SFLA 算法在常见规则中进行搜索, 即候选规则集中不包括由 GP 生成的规则.

3) SHAG: 本文所提出的算法.

表 7 中 GAP 由式 (23) 计算得出:

$$GAP = \frac{ref_i - SHAG_i}{SHAG_i} \times 100\,\% \qquad (23)$$

其中, ref_i 为 GBA 或 SBA 所得目标函数值, $SHAG_i$ 为 SHAG 所得目标函数值.

GBA 与 SHAG 相比, GAP 值的范围为 4.73 %～23.05 %, 平均值为 15.89 %. 值得注意的是, GBA 采用了与 SHAG 相同的候选规则集, 这说明了 SHAG 中的改进的混合蛙跳算法可以有效地搜索出优异的规则组合. 并且 SHAG 比较高效, 在最大规模下的运行时间也仅为 39.362 s.

规则选择框架都选用改进的混合蛙跳算法 SHAG 和 SBA 相比, 候选规则集包含 GP 产生的规则的 SHAG 在平均性能上要比候选规则集中只含有预设规则的 SBA 高出 37 %, 这充分说明了引入 GP 产生规则可以在很大程度上改善候选规则集, 进而提升算法的性能.

3.4　最优性分析

跨单元调度需要考虑生产和运输的协同, 与经典调度问题相比问题模型更为复杂, 难以求解下界, 因此对算法的最优性进行理论分析存在很大困难. 为了验证算法的最优性, 本文增加了与 CPLEX 的对比.

由于问题比较复杂, CPLEX 对于大部分测试问题在限定时间内 (3 h) 得不到解. 因此, 作者额外设计了只含有 2 个或 3 个单元的小规模测试问题. 从表 8 中可以看出, 对于测试问题 $j6/m6/c2$, CPLEX 的解比本文算法好 12.58 %; 当测试问题增加 1 个工件 ($j7/m6/c2$), CPLEX 的运行时间达到时间上限 (3 h), 本文算法耗时 0.339 s, 且比 CPLEX 找到的可行解优 36.99 %. 对于更大的问题规模, CPLEX 无法在运行时间上限之内找到可行解.

由于 CPLEX 对于绝大多数测试问题, 难以在可接受时间内求得可行解, 为了进一步验证算法性能, 本文将原问题分解为机器调度维度和运输调度维

表 7 GBA、SBA 和 SHAG 对比

Table 7 Comparison between GBA, SBA and SHAG

问题模型	GBA	SBA	SHAG	SHAG 对 GBA 的 GAP (%)	SHAG 对 SBA 的 GAP (%)	SHAG 的运行时间 (s)
$j4/m17/c4$	9 156.4	10 771.0	8 742.4	4.73	23.20	1.348
$j8/m17/c4$	14 394.8	18 314.4	13 655.4	9.36	34.11	2.642
$j11/m21/c5$	38 648.6	43 434.8	34 004.4	13.65	27.73	4.172
$j20/m21/c5$	95 821.8	114 757.4	82 043.8	16.79	39.87	7.471
$j28/m21/c5$	164 684.6	190 316.8	143 412.8	14.83	32.70	9.381
$j36/m21/c5$	377 139.0	427 002.8	315 435.2	19.56	35.36	13.510
$j40/m25/c6$	158 113.0	190 436.6	128 492.4	23.05	48.20	16.421
$j48/m25/c6$	251 846.0	300 007.4	208 558.4	20.75	48.20	22.003
$j56/m25/c6$	407 283.2	499 743.0	339 499.0	19.96	47.20	25.572
$j64/m25/c6$	33 831.6	402 733.6	295 323.2	14.73	36.37	31.922
$j72/m25/c6$	467 445.6	557 942.8	402 147.0	16.23	38.74	37.140
$j80/m25/c6$	464 694.6	550 884.6	397 006.0	17.04	38.75	39.362
平均值	232 383.2	275 528.7	197 360.0	15.89	37.17	17.579

表 8 GBA、SBA 和 SHAG 对比

Table 8 Comparison between GBA, SBA and SHAG

问题模型	SHAG 算法	SHAG 的运行时间 (s)	CPLEX	CPLEX 的运行时间 (s)	SHAG 对 CPLEX 的 GAP (%)	SHAG 对 CPLEX 运行时间的 GAP
$j6/m6/c2$	218.6	0.201	191.1	443.3	−12.58	2 109.95
$j7/m6/c2$	257.4	0.339	352.6	10 800.0	36.99	31 857.41
$j7/m8/c3$	539.66	0.373	—	—	—	—

度, 分别利用 CPLEX 求解, 并与本文算法进行比较.

在针对机器调度的对比中, 两种算法设定了相同的运输调度, 仅对机器调度进行决策. 从表 9 中可以看出, 在问题规模为 $j4/m17/c4$ 到 $j40/m25/c6$ 时, SHAG 算法性能要差于 CPLEX, GAP 为 $-4.61\%\sim-39.26\%$. 但在问题规模为 $j48/m25/c6$ 到 $j80/m25/c6$ 时, SHAG 算法得到的解要更好, GAP 为 $2.51\%\sim28.06\%$. SHAG 算法对 CPLEX 的平均 GAP 为 22.93%.

表 9 机器调度对比

Table 9 Comparison between CPLEX and SFLA with respect to part sequencing

问题模型	CPLEX	SHAG 算法	SHAG 对 CPLEX 的 GAP (%)
$j4/m17/c4$	21 564.0	30 029.2	−39.26
$j8/m17/c4$	32 356.0	43 937.4	−35.79
$j11/m21/c5$	55 675.0	70 673.2	−26.94
$j20/m21/c5$	142 648.0	162 373.0	−13.83
$j28/m21/c5$	208 672.0	226 267.8	−8.43
$j36/m21/c5$	333 709.0	350 057.4	−4.90
$j40/m25/c6$	438 285.0	458 490.4	−4.61
$j48/m25/c6$	556 850.0	542 891.6	2.51
$j56/m25/c6$	851 285.0	762 744.6	10.40
$j64/m25/c6$	1 445 203.0	1 044 731.0	27.71
$j72/m25/c6$	1 819 874.0	1 177 098.0	35.32
$j80/m25/c6$	2 098 906.0	1 300 024.0	38.06
平均值	514 109.9	514 109.9	22.93

在针对运输调度的对比中, 两种算法设定了相同的机器调度, 仅对运输调度进行决策. 从表 10 中可以看出, CPLEX 在规定时间内只得到了一个测试用例的解, 且差于 SHAG 得到的解, GAP 为 1.08%. 而 SHAG 可以得到所有测试用例的解.

实验结果表明, 对于机器调度子问题, 本文算法在较小规模的测试问题上不如 CPLEX, 但随着问题规模增大, 本文算法优势愈加明显, 且计算效率较高; 对于运输调度子问题, 本文算法在小规模测试问题上略优于 CPLEX, 随着问题规模增大, CPLEX 无法在时间上限之内求得可行解.

3.5 收敛性分析

在收敛性方面, 本文与基本混合蛙跳算法进行了对比, 本文算法对基本混合蛙跳算法在种群划分和种群内更新策略上进行了改进, 并增加了随机扰动和 Path-relinking 两种策略. 从图 11 可以看出, 基本混合蛙跳算法在 103 次趋于稳定, 此时 TWT 值大约为 1 475 000. SHAG 算法在 137 次趋于稳定,

表 10　运输调度对比

Table 10　Comparison between CPLEX and SFLA with respect to vehicle routing

问题模型	CPLEX	SHAG 算法	SHAG 对 CPLEX 的 GAP (%)
$j4/m17/c4$	30 365.0	30 029.2	1.08
$j8/m17/c4$	–	43 937.4	–
$j11/m21/c5$	–	70 673.2	–
$j20/m21/c5$	–	162 373.0	–
$j28/m21/c5$	–	226 267.8	–
$j36/m21/c5$	–	350 057.4	–
$j40/m25/c6$	–	458 490.4	–
$j48/m25/c6$	–	542 891.6	–
$j56/m25/c6$	–	762 744.6	–
$j64/m25/c6$	–	1 044 731.0	–
$j72/m25/c6$	–	1 177 098.0	–
$j80/m25/c6$	–	1 300 024.0	–
平均值	–	514 109.9	–

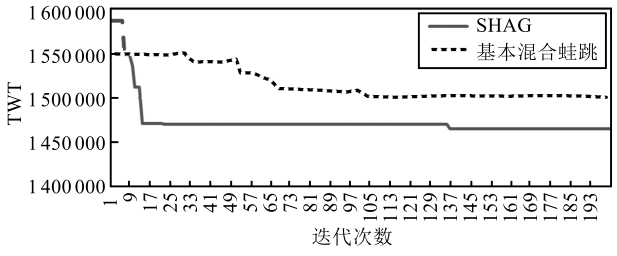


图 11　SHAG 与基本混合蛙跳的对比

Fig. 11　Comparison between SHAG and basic SFLA

TWT 值为 1 425 000 左右. SHAG 算法比基本混合蛙跳算法能收敛到更优解.

对比实验的参数按照第 3.2 节进行设置. 实验采用包含 80 个工件、25 台机器、6 个单元的最大规模的测试用例.

结果表明, 本文算法收敛速度要快于基本混合蛙跳算法, 并能得到更优解.

4　结论

本文在考虑了复杂的跨单元调度问题的同时, 结合制造企业的生产实际, 将运输能力受限的运输决策问题加入了跨单元调度问题模型中, 提出了运输能力受限的跨单元调度问题模型. 针对此问题模型, 本文提出了一种基于混合蛙跳与遗传规划的超启

发式算法, 把新型元启发式算法 SFLA 作为高层框架, 用来搜素底层启发式规则, 并用 GP 产生优质规则扩充候选规则集. 实验表明, SHAG 中的改进的混合蛙跳算法可以有效地搜索出优异的规则组合; 且引入 GP 产生规则可以在很大程度上改善候选规则集, 进而提升算法的性能.

针对本文的研究内容, 还可以在如下几个方面展开进一步的研究. 一是本文考虑的 CMS 内所有机器均为单处理机, 每次只能处理一个工件. 下一步的研究中可以考虑较为复杂的条件, 例如允许同时处理多个工件的批处理机和在运输过程中在各个单元中装载工件, 以更加贴近生产实际; 二是算法本身可在细节上加以完善, 例如在 Path-relinking 时设置不同接受条件, 或是在不同时间为工件或机器设置不同的启发式规则等, 以进一步提高寻优能力. 此外, 我们对国内大型装备制造企业进行调研时, 发现在复杂产品的跨单元生产中, 除了本文的运输模式之外, 还存在另一种模式, 即运输工具由多个单元共享, 在单元制造系统的统一调配下可以运送不同单元的工件. 我们将在未来的工作中对此种运输模式进行研究.

References

1 Rheault M, Drolet J R, Abdulnour G. Dynamic cellular manufacturing system (DCMS). *Computers and Industrial Engineering*, 1996, **31**(1−2): 143−146

2 Garza O, Smunt T L. Countering the negative impact of intercell flow in cellular manufacturing. *Journal of Operations Management*, 1991, **10**(1): 92−118

3 Khaksar-Haghani F, Kia R, Mahdavi I, Kazemi M. A genetic algorithm for solving a multi-floor layout design model of a cellular manufactur-ing system with alternative process routings and flexible configuration. *The International Journal of Advanced Manufacturing Technology*, 2013, **66**(5−8): 845−865

4 Kia R, Baboli A, Javadian N, Tavakkoli-Moghaddam R, Kazemi M, Khorrami J. Solving a group layout design model of a dynamic cellular manufacturing system with alternative process routings, lot splitting and flexible reconfiguration by simulated annealing. *Computers and Operations Research*, 2012, **39**(11): 2642−2658

5 Gupta J N D, Schaller J E. Minimizing flow time in a flowline manufacturing cell with family setup times. *Journal of the Operational Research Society*, 2006, **57**(2): 163−176

6 Tsai C H, Li R K. A due-date oriented scheduling heuristic for job shop cellular manufacturing system. *International Journal of Industrial Engineering — Theory Applications and Practice*, 2000, **7**(1): 76−88

7 Solimanpur M, Elmi A. A tabu search approach for cell scheduling problem with makespan criterion. *International Journal of Production Economics*, 2013, **141**(2): 639−645

8 Golmohammadi A, Ghodsi R. Applying an integer Electromagnetism-like algorithm to solve the cellular manufacturing scheduling problem with an integrated approach. In: Proceedings of the 2009 International Conference on Computers and Industrial Engineering. Troyes: IEEE, 2009. 34−39

9 Mosbah A B, Dao T M. Optimimization of group scheduling using simulation with the meta-heuristic extended great deluge (EGD) approach. In: Proceedings of the 2010 IEEE International Conference on Industrial Engineering and Engineering Management. Macao: IEEE, 2010. 275−280

10 Gholipour K Y, TavakkoliM R, Khorrami A. Solving a multicriteria group scheduling problem for a cellular manufacturing system by scatter search. *Journal of the Chinese Institute of Industrial Engineers*, 2011, **28**(3): 192−205

11 Li W L, Murata T. Particle swarm optimization method for rescheduling of job processing against machine breakdowns for nondisruptive cell manufacturing system. In: Proceedings of the 6th International Conference on New Trends in Information Science and Service Science and Data Mining (ISSDM). Taipei, China: IEEE, 2012. 523−528

12 Meng X W, Ju Y H, Wang X H, Wang Y, Li D N. An ACO-based approach for intercell scheduling with various types of machines. In: Proceedings of the 25th Chinese Control and Decision Conference (CCDC). Guiyang, China: IEEE, 2013. 1812−1817

13 Pajoutan M, Golmohammadi A, Seifbarghy M. CMS scheduling problem considering material handling and routing flexibility. *The International Journal of Advanced Manufacturing Technology*, 2014, **72**(5−8): 881−893

14 Li Dong-Ni, Xiao Guang-Xue, Wang Yan, Tang Jia-Fu. An intercell scheduling approach considering flexible processing routes. *Acta Automatica Sinica*, 2012, **38**(6): 969−975 (李冬妮, 肖广雪, 王妍, 唐加福. 一种柔性路径下的跨单元调度方法. 自动化学报, 2012, **38**(6): 969-975)

15 Li D N, Wang Y, Xiao G X, Tang J F. Dynamic parts scheduling in multiple job shop cells considering intercell moves and flexible routes. *Computer and Operations Research*, 2013, **40**(5):1207−1223

16 Eusuff M M, Lansey K E. Optimization of water distribution network design using the shuffled frog leaping algorithm. *Journal of Water Resources Planning and Management*, 2003, **129**(3): 210−225

17 Xu Y, Wang L, Wang S Y, Liu M. An effective shuffled frog-leaping algorithm for solving the hybrid flow-shop scheduling problem with identical parallel machines. *Engineering Optimization*, 2013, **45**(12): 1409−1430

18 Luke S, Panait L, Balan G, Paus S, Skolicki Z, Kicinger R, Popovici E, Sullivan K, Harrison J, Bassett J, Hubley R, Desai A, Chircop A, Compton J, Haddon W, Donnelly S, Jamil B, Zelibor J, Kangas E, Abidi F, Mooers H, O′Beirne J, Talukder K A, McDermott J. ECJ: a java-based evolutionary computation research system. [Online], available: http://cs.gmu.edu/ eclab/projects/ecj/, June 5, 2014

19 Koza J R. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 1994, **4**(2): 87−112

20 Robilliard D, Marion-Poty V, Fonlupt C. Genetic programming on graphics processing units. *Genetic Programming and Evolvable Machines*, 2009, **10**(4): 447−471

21 Park S C, Raman N, Shaw M J. Adaptive scheduling in dynamic flexible manufacturing systems: a dynamic rule selection approach. *IEEE Transactions on Robotics and Automation*, 1997, **13**(4): 486−502

22 Barman S. Simple priority rule combinations: an approach to improve both flow time and tardiness. *International Journal of Production Research*, 1997, **35**(10): 2857−2870

23 Laforge R L, Barman S. The performance of simple priority rule combinations in a flow dominant shop. *Production and Inventory-Management Journal*, 1989, **30**(3): 1−4

24 Sarper H, Henry M C. Combinatorial evaluation of six dispatchingrules in a dynamic two-machine flow shop. *Omega*, 1996, **24**(1): 73−81

25 Li D N, Meng X W, Liang Q Q, Zhao J Q. A heuristic-search genetic algorithm for multi-stage hybrid flow shop scheduling with single processing machines and batch processing machines. *Journal of Intelligent Manufacturing*, DOI: 10.1007/s10845-014-0874-y

26 Yang T, Kuo Y, Cho C. A genetic algorithms simulation approachfor the multi-attribute combinatorial dispatching decision problem. *European Journal of Operational Research*, 2007, **176**(3): 1859−1873

27 Ebrahimi J, Hosseinian S H, Gharehpetian G B. Unit commitment problem solution using shuffled frog leaping algorithm. *IEEE Transactions on Power Systems*, 2011, **26**(2): 573−581

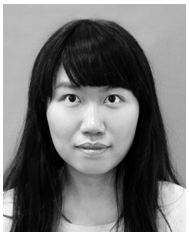
28 Rahimi-Vahed A, Mirzaei A H. A hybrid multi-objective shuffled frog-leaping algorithm for a mixed-model assembly line sequencing problem. *Computers and Industrial Engineering*, 2007, **53**(4): 642−666

29 Eusuff M, Lansey K, Pasha F. Shuffled frog-leaping algorithm: a memetic metaheuristic for discrete optimization. *Engineering Optimization*, 2006, **38**(2): 129−154

30 Teekeng W, Thammano A. A combination of shuffled frog leaping and fuzzy logic for flexible job-shop scheduling problems. *Procedia Computer Science*, 2011, **6**: 69−75

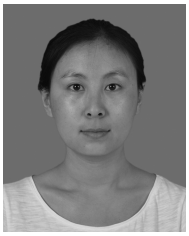
31 Luo X H, Ye Y, Li X. Solving TSP with shuffled frog-leaping algorithm. In: Proceedings of the 8th International Conference on Intelligent Systems Design and Applications. Washington, D. C., USA: IEEE, 2008. 228−232

32 Glover F, Laguna M, Martí R. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 2000, **39**(3): 653−684



贾凌云 北京理工大学计算机学院硕士研究生. 主要研究方向为演化计算和生产调度. E-mail: lingyun_jia@163.com
(**JIA Ling-Yun** Master student at the School of Computer Science, Beijing Institute of Technology. Her research interest covers evolutionary computation and production scheduling.)

李冬妮 北京理工大学计算机学院副教授. 主要研究方向为智能优化, 企业计算, 物流管理等. 本文通信作者.
E-mail: ldn@bit.edu.cn
(**LI Dong-Ni** Associate professor at the School of Computer Science, Beijing Institute of Technology. Her research interest covers intelligent optimization, enterprise computation, and logistics management. Corresponding author of this paper.)



田云娜 北京理工大学计算机学院博士研究生. 主要研究方向为演化计算与智能优化方法.
E-mail: ydtianyunna@163.com
(**TIAN Yun-Na** Ph. D. candidate at the School of Computer Science, Beijing Institute of Technology. Her research interest covers evolutionary computation and intelligent optimization approaches.)