6.4 资源约束的项目调度

在资源和前后约束下带有最小化项目持续时间目标的活动(activity)调度问题在文献中被称作资源约束的项目调度(resource-constrained project scheduling)问题[37]。该问题的基本类型可以描述如下：一个项目包含许多相互关联的活动，其中每一个都有确定的持续时间和给定的资源要求。资源的数量是有限的，但会随着时间而更新。资源之间不可替代，活动不能被中断。问题的解是确定考虑前后约束和资源约束的活动开始时间以优化目标。

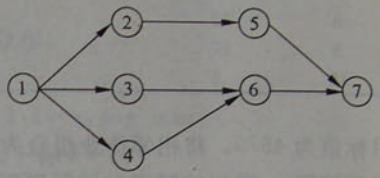


图 6.2 一个项目的网络表示

作为例子，考虑一个简单的包含5个活动的项目(如图6.2所示)。节点表示活动，有向连线表示先后约束。假设总的资源数量为4个单位。每个活动的持续时间和资源消耗见表6.3。

表 6.3 每个活动的持续时间和资源消耗

活动	持续时间	资源消耗	父活动
1	虚拟活动		
2	4	2	1
3	3	3	1
4	2	2	1
5	4	1	2
6	2	2	3,4
7	虚拟活动		

如果项目的调度如图6.3所示，则所需资源随着时间的变化(即资源数据图)用图6.4表示。

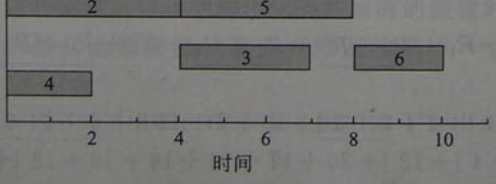


图 6.3 Gantt 图调度

该问题可以用数学语言表示为

$$\min \quad t_n \tag{6.21}$$

$$\text{s.t.} \quad t_j - t_i \geqslant d_i, \quad \forall j \in S_i \tag{6.22}$$

$$\sum_{t_i \in A_{t_i}} r_{ik} \leqslant b_k, \quad k = 1,2,\cdots,m \tag{6.23}$$

$$t_i \geqslant 0, \quad i = 1,2,\cdots,n \tag{6.24}$$

其中 t_i 是活动 i 的开始时间，d_i 是活动 i 的持续时间(进行时间)，S_i 是在活动 i 之前的活动的集合，r_{ik} 是活动 i 需要资源 k 的数量，b_k 是资源 k 的总量，A_{t_i} 是 t_i 时刻进行的活动的集合，m 是不同资源类型的数量。活动 1 和 n 是虚拟活动，其作用是标记项目开始和结束的时间。目标是最小化总的项目持续时间。约束(6.22)确保不违背前后约束。约束(6.23)确保任何时段内所有活动消耗的资源 k 的数量不超过其限制的数量。该研究主要关注如何处理问题中的前后约束。有人提出了一种新的编码方法，该方法本质上可以对于给定的实例表示活动所有的可行排列[105,106,110]。

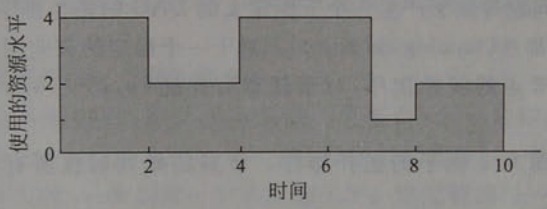


图 6.4 资源数据图

早期的尝试是使用数学规划中的标准求解方法来寻找问题的准确最优解[121,143,506,589,611]。由于资源约束的项目调度问题是 NP 难的，对于大的项目来说，问题的规模可能致使寻求最优解的方法在计算上不现实。在这种情况下，该问题应该采用启发式问题求解方法来求解，这些方法采用相当简单的调度规则来合理地产生次最优调度。在过去的 30 年里，提出并测试了大量启发式算法，我们推荐读者参考 Alvarez-Valdés 和 Tamarit 的综述与计算比较[12]。

现有的大多数启发式方法可以看作优先调度规则，它们在为解决资源冲突而进行顺序决策时根据临时或与资源相关的启发式规则来分配活动的优先权。最近 Cheng 和 Gen 提出了一种混合遗传算法用于求解资源约束的项目调度问题。该问题本质上包含以下两个基本部分：(1)确定不违反前后约束的活动进行次序；(2)然后确定不违反资源约束的每个活动的开始时间。如何确定活动的次序是问题的关键，原因在于一旦活动次序确定，可以容易地根据次序来构造一个调度。Cheng 和 Gen 方法的基本思想是(1)采用遗传算法来进化适当的活动进行次序；(2)采用最佳配合过程来计算活动的开

始时间。

6.4.1 基于优先权的编码

如何将问题的解编码为染色体是遗传算法其余步骤进行的先决重要问题。在 Cheng 和 Gen 方法中，采用遗传算法来进化适当的活动进行次序，因此仅须对活动次序进行编码。这本质上是排列问题。由于活动之间存在前后约束，任意的排列可能产生不可行的进行次序。如何有效地产生能够处理前后约束的编码是该方法的关键步骤，也是其余步骤的先决条件。他们提出了一种基于优先权的编码方法来处理这个困难，该方法来源于有向无环图模型。

有向无环图(directed acyclic graph) 一个项目的实例可以用有向无环图来表示[9]。一个有向无环图(directed acyclic graph) $G=(V,A)$包含一个代表活动的节点的集合 V 和一个代表活动间前后约束的有向边的集合 A。术语节点(node)和活动(activity)在以下几小节中交替使用。图 6.2 给出了 DAG 表示一个项目的例子。对所有活动进行排序以满足前后约束的问题等效于产生一个下面定义的 DAG 的拓扑排序[134]。

定义 6.3(*拓扑排序*(topological sort)) 对于一个给定的有向图 $G=(V,A)$，一个拓扑排序是一个所有节点的线性次序，对于任意有向边 $(u,v)\in A$，u 在该次序中先于 v 出现。

图 6.5 表示了图 6.2 例子的拓扑排序。节点的排列确保所有有向边的方向从左到右。

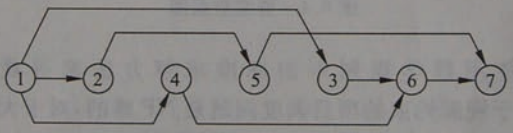


图 6.5 图 6.2DAG 的拓扑排序

容易看出活动的随机排列通常不能对应给定 DAG 的拓扑排序，因此排列编码(实际遗传算法最常用的编码方式)对该问题不适用。Cheng 和 Gen 提出了一种称作基于优先权编码(priority-based encoding)的新编码方式。该方法 (1)可以表示给定实例所有可能的拓扑排序；(2)编码的任何排列总是对应着一个拓扑排序，即一个可行解。

基于优先权的编码(priority-based encoding) 遗传算法的基因包含两种信息：基因座(locus)(基因在染色体结构中的位置)和等位基因(allele)(基因的值)。因此有两种可能的方式来表示一个活动：采用基因座和采用等位基因。这里用位置来表示一个活动的 ID，基因的值用来表示活动的优先权，如图 6.6 所示。基因的值是一个$[1,n]$之间惟一的整数。数值越大则优先权越高。

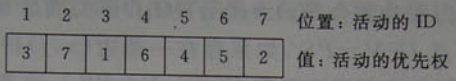


图 6.6 基于优先权的编码

采用了一次通过过程来由染色体产生拓扑排序，即一次性从左到右确定活动。当进行关于一个位置的决策时，若干活动可能就这一位置进行竞争，具有最高优先权的一个赢得该位置。这种编码不是直接表示给定 DAG 的拓扑排序。它仅包含了用于解决冲突的某些信息。大多数情况下可以根据编码来惟一地确定拓扑排序。优先权的任何改变通常导致不同的拓扑排序。因此这种编码本质上能够表示给定 DAG 所有可能的拓扑排序。

下面考察如何从编码产生一个拓扑排序。考虑图 6.2 表示的例子。向量 $\boldsymbol{A}[\cdot]$用来存储产生的拓扑排序。最初 $\boldsymbol{A}[1]=1$。然后三个活动 2,3,4 来竞争 $\boldsymbol{A}[2]$。它们的优先权分别是 7,1,6。由于活动 2 具有最高的优先权，它赢得该位置。在固定 $\boldsymbol{A}[2]=2$ 以后，下一个位置 $\boldsymbol{A}[3]$的竞争者是活动 3,4,5。活动 4 赢得该位置并固定 $\boldsymbol{A}[3]=4$。重复下面两步：

第 1 步：对于当前位置构造一个候选集合。

第 2 步：选择具有最高优先权的活动，直到活动一个图 6.5 所示的拓扑排序。

拓扑排序(topological sort) 产生一个拓扑排序的过程按照一次通过方式进行：从左到右产生一个拓扑排序，一次固定一个节点的次序。该过程的每一次迭代中所有的节点都处于下列三种状态之一：

1. 已排序节点(sorted node)：已经在构造的拓扑排序中的节点
2. 合格节点(eligible node)：那些所有父节点都是已排序节点的节点
3. 自由节点(free node)：所有其他节点

当然关键部分就是如何寻找合格节点的集合。下面的定义和定理解释了如何产生这样的集合以及过程如何进行。

对于有向图 $G=(V,A)$的边(u,v)，我们称该边从 u 传出(incident from u)以及传入 v(incident to v)。此外称 u 与 v 相临近，v 也与 u 相临近。节点 u 的内度(indegree)，用 $d^{\text{IN}}(u)$表示，就是传入节点 u 的节点数量。图 G 的一个割集是一些边的集合，该集合具有 $(X, V-X)$的形式，其中 $u\in X, v\in V-X$。通常用 $\overline{X}$ 来表示 $V-X$。一个部分拓扑排序(partial topological sort)就是一个仅包含固定次序的前 $t(t<|V|)$个节点的排序。设 $PS_t\subset V$ 表示关于给定部分拓扑排序的节点集合，其中下标 t 代表集合的规模，即$|PS_t|=t$。设 $C(PS_t, V-PS_t)=\{(i,j)\mid i\in PS_t, j\in V-PS_t\}$表示关于给定部分拓扑排序的有向图的割集。于是有下面的引理。

引理 6.2(合格节点) 对于一个给定的节点为 PS_t 的部分拓扑排序，节点 $j\in V-PS_t$

是合格的当且仅当存在传入 j 的边集合 $S(j)=\{(i,j)\mid(i,j)\in A\}$ 满足 $S(j)\subseteq C(PS_t,V-PS_t)$。

证明　对于给定的节点 $j\in V-PS_t$，如果存在一条边 (x,j) 传入 j，节点 x 是节点 j 的父节点。如果所有的这样的边都属于割集 $C(PS_t,V-PS_t)$，这意味着所有节点 j 的父节点都属于 PS_t，即它们是已排序节点，因此节点 j 是合格的。

假设对于合格节点 j，并非所有传入 j 的边都属于割集。即 $|C(PS_t,V-PS_t)\cup S(j)|>|C(PS_t,V-PS_t)|$。于是至少一个父节点属于集合 $V-PS_t$，即至少其一个父节点不是已排序节点。这与合格节点的定义矛盾。

理论上我们可以通过本引理来检查一个节点是否合格，但很难程序化地检查是否一个集合是另一个集合的子集。下面的定理提供了确定一个合格节点的判据。

定理 6.2(合格节点判据)　对于一个合格节点 $j\in V-PS_t$，设 $S_t(j)$ 是割集 $C(PS_t,V-PS_t)$ 的一个包含所有传入 j 的边的适当子集。于是我们有 $|S_t(j)|=d^{\text{IN}}(j)$。

证明　对于一个合格节点，根据引理 6.2 有 $S_t(j)\cap S(j)=S(j)$，而且 $S_t(j)\cup S(j)=S(j)$。由于 $|S(j)|=d^{\text{IN}}(j)$，于是证明了定理。

现在可以仅通过验证割集中传入一个节点的边的数量是否等于其内度来检查该点是否为合格节点。这个判据易于编程。让我们考虑图 6.7 给出的例子。部分拓扑排序是 $PS_3=\{1,2,3\}$，割集包括有向边 $C(PS_3,V-PS_3)=\{(1,4),(2,5),(2,6),(3,6),(3,7)\}$。由于节点 6 的内度 $d^{\text{IN}}(6)=2$ 而且传入该节点的两个边属于割集，因此节点 6 是合格节点。由于节点 7 的内度为 2，而仅有 1 条属于割集的边传入该节点，因此节点 7 是自由节点。该节点的一个父节点是节点 4，它是一个合格节点，不是已排序节点。

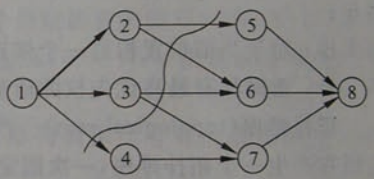
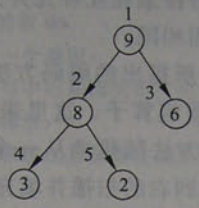


图 6.7　部分拓扑排序，割集和合格节点

拓扑排序过程的基本思想是，在过程进行的每一步都要 (1)根据定理 6.2 确定合格节点集合；(2)将该集合中具有最高优先级的一个节点移去；(3)将该节点固定在部分拓扑排序中。该过程中最大的部分是维持合格节点集合的内容，这项内容可以通过优先队列(priority queue)来有效地实现[134]。优先队列是用来有效存储和操作项目集合的抽象数据类型，集合中的每个项目都有一个与之相关联的值，称作关键字(key)或优先权(priority)。该数据结构支持将项目插入队列和从队列中将具有最高优先权的项目移除，即我们用来维持合格节点的操作。优先队列的一个特殊实现通过堆(heap)来完成。堆是一个具有特殊属性的二进制树，该树中每个节点的优先权都不小于其子节点的优先权[134]。堆可以存储在任何向量中，对于一个下标为 i 的节点，其父节点的下标为 $[i/2]$，其左子节点的下标为 $2i$，右子节点的下标为 $2i+1$。假设合格节点集合包含 3,4,5,6,7 节点。每个节点所对应的优先权为 2,6,9,3,8。对应的用二进制树来表示的堆及其实现

向量见图 6.8。树中每个节点圈中的树是优先权值。节点旁边的数值是其在向量中对应的下标。



(a)

1	2	3	4	5	下标
9	8	6	3	2	优先权
5	7	4	6	3	活动

(b)

图 6.8 堆看作(a)二进制树;(b)向量

设 t 是过程的迭代下标,V 是所有节点的集合,Q_i 是所有活动 i 的直接紧前活动集合,PS[·]是存储拓扑排序的向量,CUT[i]是割集中传入节点 i 的边的数量,S_t 是第 t 步合格节点的集合。从染色体中产生拓扑排序的过程如下:

拓扑排序(topological sort)过程

第 1 步:初始化

$t \leftarrow 1$(迭代下标)

PS[t]$\leftarrow 1$(初始化拓扑排序)

$S_t \leftarrow Q_1$(初始化优先队列)

CUT[i]$\leftarrow -1, \forall i \in Q_1$(初始化割集中边的数量)

CUT[i]$\leftarrow -1, \forall i \in V - Q_1$

第 2 步:执行终止测试。如果 PS[t]$=n$,进入第 6 步,否则 $t \leftarrow t+1$,继续。

第 3 步:固定第 t 个节点,从优先队列 S_t 中移除具有最高优先权的节点 i^* 并将其放入向量 PS[t]。

第 4 步:执行割集更新

$$\mathrm{CUT}[i] \leftarrow \mathrm{CUT}[i] + 1, \qquad \forall i \in Q_{i^*}$$

第 5 步:更新合格节点集合。对于所有的 $i \in Q_{i^*}$,如果 CUT[i]$=d^{\mathrm{IN}}(i)$,将 i 放入优先队列 S_t,返回第 2 步。

第 6 步:返回完整的拓扑排序 PS[·]。

6.4.2 遗传算子

遗传算法由遗传算子来实现并受到选择压力的导向。通常杂交算子用作主要算子,遗传系统的性能主要依赖于该算子;变异算子用于在不同的染色体中产生自发随机变化,因此成为次要算子。

在 Cheng 和 Gen 的实现中,采用了一种交互式方法来设计遗传算子:一个算子用于

执行广度搜索以探索局部最优以外的区域；另一个算子用于执行深度搜索以寻找改进的解。两种类型的搜索方法(深度和广度)构成了遗传搜索相互补充并完备的组成部分。采用这种方法时杂交和变异算子在遗传搜索中的作用相同。

基因位置的杂交(position-based crossover)　所提出的编码方法的本质可以看作一种排列编码。人们已经针对排列表示研究了许多重组算子。这里采用了 Syswerda 提出的基于位置的杂交算子[604]，如图 6.9 所示。这种方法随机地从一个父代中提取一些基因，子代中余下的基因通过对另一个父代进行从左到右的扫描并填充空位得到。

交换变异(swap mutation)　这里采用了交换变异算子，该方法简单地随机选择两个位置并交换其内容，如图 6.10 所示。

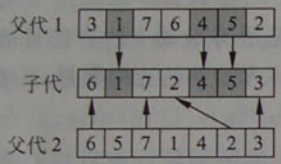


图 6.9　基于位置的杂交算子

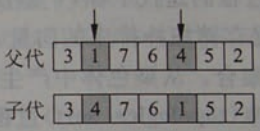


图 6.10　交换变异算子

基于局部搜索的变异(local search-based mutation)　局部搜索通过搜索当前解的邻域来寻找问题的改进解。局部搜索的实现需要初始当前解、对于初始当前解邻域的定义以及选择下一个当前解的方法。通过少量改变来寻找一个改进解的思想可以用于变异算子。观察交换变异，通过成对交换产生的染色体可以看作原始染色体的邻居。一个染色体的邻域(neighborhood)于是可以定义为通过成对交换产生的染色体集合。对于一对基因，一个称作中心(pivot)，该基因对于给定的邻域来说是固定的，另一个随机选择，如图 6.11 所示。在一个给定的邻域中，如果一个染色体在适应值上比其他任意染色体都要好，称其为局部最优(local optimum)。邻域的规模影响局部最优的质量。小规模邻域和大规模领域的折衷是清晰的：邻居数越大，找到优秀邻居的概率就越大，但寻找的时间也越长。这种变异算子的摘要描述如下：

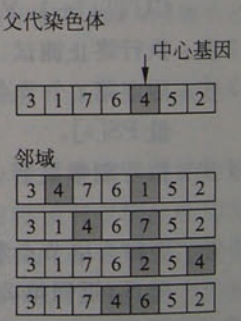


图 6.11　当前染色体及其邻域

基因局部搜索的变异过程

inputs: P(父代)
output: C(子代)
begin

```
设 P 为当前染色体;
选择一个中心基因;
while i<特定的数 do
    随机选择一个基因;
    通过将其与中心基因交换产生一个邻居;
    评价这个邻居;
    如果这个邻居比当前染色体更优秀,将其设为当前染色体;
    i←i+1
end
保存 C 中的当前基因
end
```

6.4.3 评价与选择

在每一代中,采用某种适应值度量方式对染色体进行评价。在评价阶段采用了下面4个主要步骤:

第1步:将染色体转换为拓扑排序。

第2步:从拓扑排序产生调度。

第3步:计算每个调度的目标值。

第4步:将目标值转换为适应值。

第1步的过程在6.4.2节讨论。由于拓扑排序给出了活动的可行次序,我们根据活动在拓扑排序中出现的次序来构造调度,一旦资源允许就一次调度一个活动。设 i 为过程迭代的下标,V 为所有节点的集合,P_i 为活动 i 的所有直接紧前活动集合,PS[·]为存储拓扑排序的向量,σ_j 和 ϕ_j 分别为与活动 j 相关的开始和结束时间,$\boldsymbol{b}_k[l]$为记录时间 l 可行的资源 k 数量的向量,d_j 为活动 j 的持续时间,r_{jk}为活动 j 对资源 k 的消耗。根据给定的拓扑排序确定每个活动开始和结束时间的过程如下:

活动的开始和结束时间过程

第1步:初始化

$i \leftarrow 1$(迭代下标)

$j \leftarrow \mathrm{PS}[i]$(初始化活动)

$\sigma_j \leftarrow 0, \phi_j \leftarrow 0$(初始化活动的开始和结束时间)

$\boldsymbol{b}_k[l] \leftarrow b_k, l=1,2,\cdots,\sum_{j=1}^{n} d_j, k=1,2,\cdots,m$(初始化资源)

第2步:执行终止测试。如果 $i=n$,进入第5步,否则 $i \leftarrow i+1$,继续。

第3步:确定开始和结束时间。

$j \leftarrow \mathrm{PS}[i]$

$\sigma_j^{\min} \leftarrow \max\{\phi_l \mid l \in P_j\}$

$\sigma_j \leftarrow \min\{t \mid t \geqslant \sigma_j^{\min},\ \boldsymbol{b}_k[l] \leqslant r_{jk},\ l=t,t+1,\cdots,t+d_j,\quad k=1,2,\cdots,m\}$

$\phi_j \leftarrow \sigma_j + d_j$

第4步：更新可用资源

$\boldsymbol{b}_k[l] \leftarrow \boldsymbol{b}_k[l] - r_{jk},\quad l=t,t+1,\cdots,t+d_j,\quad k=1,2,\cdots,m$

返回第2步。

第5步：停止。返回 σ_j 和 ϕ_j。

由于目标是项目的持续时间，因此最后一个活动的结束时间就是目标值。我们处理的是最小化问题，因此必须将原始目标值转换为适应值以确保优秀个体具有大的适应值。

设 v_k 是当前种群的第 k 个染色体，$g(v_k)$是适应值函数，$f(v_k)$是目标值(即项目持续时间)，$f_{\max}$和 $f_{\min}$分别是当前种群的最大目标值和最小目标值。转换方法如下：

$$g(v_k) = \frac{f_{\max} - f(v_k) + \gamma}{f_{\max} - f_{\min} + \gamma} \tag{6.25}$$

其中 γ 是正实数，通常限制在开区间(0,1)中。使用 γ 的目的有2：(1)防止式(6.25)产生被零除；(2)使得可以将选择行为从适应值比例选择调整到纯随机选择。如果染色体间适应值的差距相对较大，则采用适应值比例选择；如果区别相对较小，则选择趋向于在相互竞争的染色体中进行纯随机选择。

在 Cheng 和 Gen 的试验中采用了轮盘赌(roulette wheel)方法(一种适应值比例选择)。最优性选择(elitist selection)与这种方法相结合在下一代中维持最优的染色体同时避免采样误差。在最优性选择中，如果当前代的最优个体没有在下一代中得到复制，随机从下一代中移除一个个体并将最优个体加入种群。

6.4.4 试验结果

初步的计算试验构造为两个部分：(1)调整遗传算法的参数以研究它们是如何影响性能的，(2)验证遗传搜索中基于局部搜索变异的性能。

参数设置 为了研究种群规模是如何影响算法性能的，将最大遗传代数固定为100，杂交率和变异率均固定在0.1。杂交率和变异率低时种群规模就成为遗传算法性能的主要影响因素了。种群规模从10到100变化。图6.12显示了每个参数设置下100次随机运行的最好、最差和平均目标值。从结果中可以看出，种群规模大于50以后，继续增加对于遗传算法的性能没有显著影响。

下面测试的目的是比较杂交和变异算子以研究哪个是遗传搜索中更为重要的算子。遗传算法在下面两种情况中进行了测试：

1. 固定变异率为0，将杂交率从0.1变化到0.9。

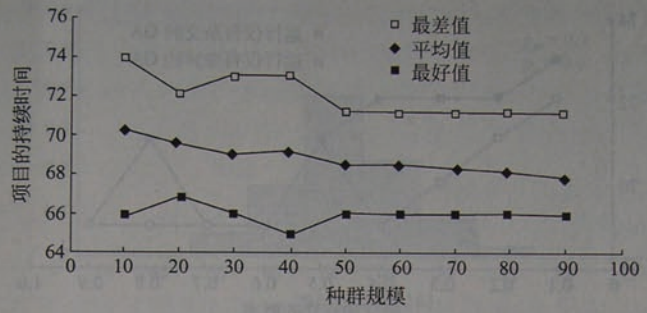


图 6.12 在不同 *pop_size* 值下目标值最好最差和平均值的比较

2. 固定杂交率为 0,将变异率从 0.1 变化到 0.9。

两种情况下均固定 $max\_gen=100$，$pop\_size=20$。图 6.13 给出了每个参数设置下 200 次随机运行的最好目标函数值,平均值见图 6.14,最差值见图 6.15。

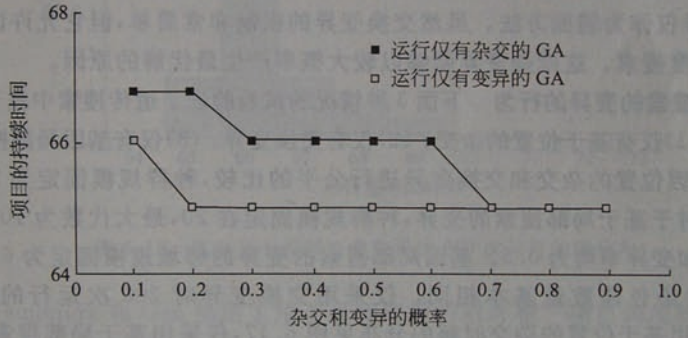


图 6.13 不同杂交率和变异率下 200 次随机运行最好值的比较

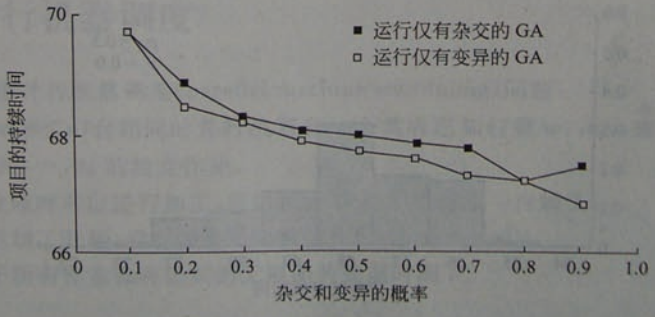


图 6.14 不同杂交率和变异率下 200 次随机运行平均值的比较

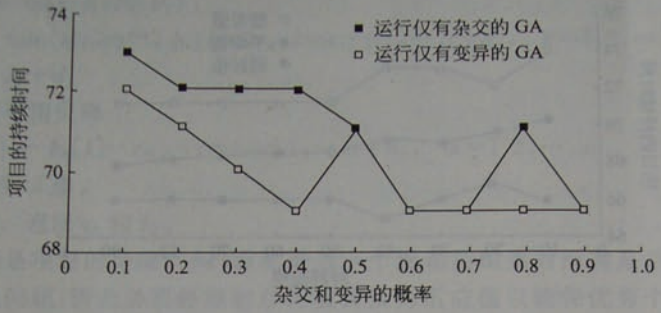


图 6.15 不同杂交率和变异率下 200 次随机运行最差值的比较

从结果中可以看出对于平均值来说，杂交和变异之间没有明显的区别，但如果遗传算法的只有变异算子时获得最优解的机会要远大于只有杂交算子时的机会。这个结果表明变异在遗传中起决定性的作用，这与传统的观念相矛盾。在传统的遗传算法中，杂交作为主要算子，变异仅作为辅助方法。虽然交换变异的机制非常简单，但它允许该给定的染色体周围进行广度搜索。这就是变异能够以较大概率产生最优解的原因。

基于局部搜索的变异的行为 下面 3 种情况的试验验证了遗传搜索中基于局部搜索的变异的行为：(1)仅有基于位置的杂交；(2)仅有交换变异；(3)仅有基因局部搜索的变异。

为了对基因位置的杂交和交换变异进行公平的比较，种群规模固定为 50，最大代数固定为 200；对于基于局部搜索的变异，种群规模固定在 20，最大代数为 100。在每种情况下，杂交率和变异率均为 0.3。基因局部搜索的变异的邻域规模固定为 6，这样使得每种算子处理的染色体数量基本相同。仅采用交换变异时 200 次运行的解的分布见图 6.16，仅采用基于位置的杂交时解的分布见图 6.17，仅采用基于局部搜索的变异时解的分布见图 6.18。容易看出基于局部搜索的变异对遗传算法的性能有明显的影响。

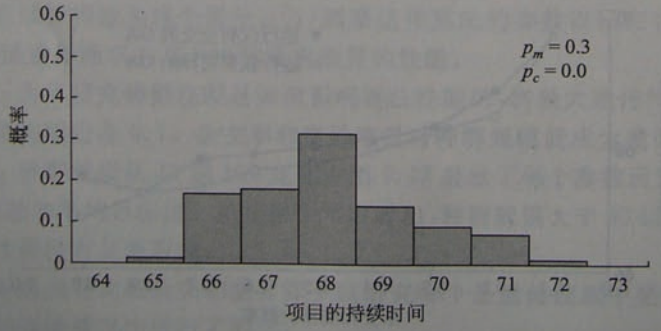


图 6.16 仅有交换变异的 200 次运行的解分布

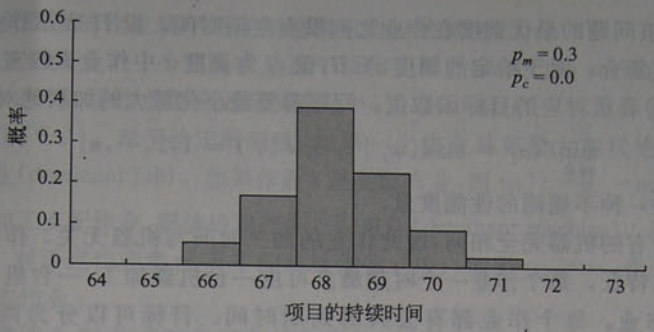


图 6.17 仅有基于位置杂交的 200 次运行的解分布

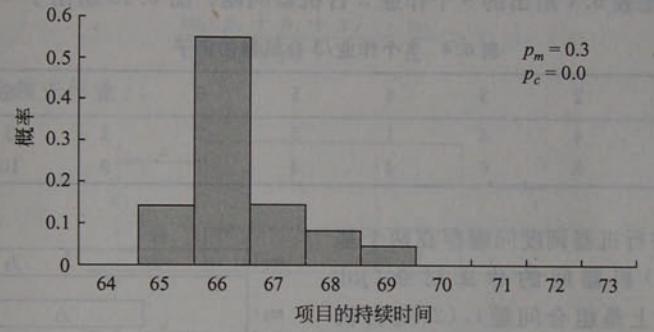


图 6.18 仅有基于局部搜索变异的 200 次运行的解分布

最近 Tsujimura 和 Gen 提出了用于求解资源约束环境下项目调度的进化算法[736]，Özdamar 提出了对于一般类别项目调度问题的遗传算法[727]。

6.5 并行机器调度

下面考虑并行机器调度(parallel machine scheduling)问题：

- 有 $m(m<n)$ 台相同的并行机器和 n 个具有已知权重 $w_1, w_2, \cdots, w_n$ 和加工时间 $p_1, p_2, \cdots, p_n$ 的独立作业。
- 作业随时可以进行加工，可以被 m 台机器中任意一台加工。
- 一旦加工开始，没有作业可以被占先(be preempted)。
- 对于所有作业都有公共的无限制的到期时间 d。

$$d \geqslant \sum_{j=1}^{n} p_j$$