

基于相关任务分配的网络计划的算法

郭 强

(西北工业大学理学院应用数学系 西安 710072)

摘 要 研究如何把具有紧前紧后关系的工作集分配给现有的人员(或设备),使完成工作集的总工期最短,并在此条件下,使得用于所有工作上的时间之和最少.文中揭示了任意改变一项工作的用时或最早开工时间引起其它工作的最早开工时间的变化规律,并在此基础上借鉴 Floyd 算法规则,建立了一种获取该问题最优解的迭代算法.这种算法能保证总工期随迭代过程递减,在总工期达到最短时,能保证总工期不变,而总用时随迭代过程递减.使用这种算法,不用绘制 PERT 图,只需输入每个人承担不同工作的用时以及各工作间的紧前紧后关系,即可算出最优分配方案、总工期及各项工作的最早开工时间和松弛时间.

关键词 分配问题;PERT 问题;A-PERT 问题;Floyd 算法;最早开工时间;松弛时间

中图法分类号 TP18

An Algorithm of Network Plan Based on Dependent Task Assignment

GUO Qiang

(Department of Applied Mathematics, School of Science, Northwestern Polytechnical University, Xi'an 710072)

Abstract This paper studies how assigning the jobs with precedence and successor relationship to every person (or facility) so that the engineering period is the shortest, and in this premise the total time of completing all the jobs in the engineering is the least. Through reveal the changed law of the earliest start time of all other jobs when the time of any job or its earliest start time is altered, an iterative algorithm is established to obtain the optimal solution in virtue of Floyd algorithm. The algorithm can ensure that the engineering period is shortened with iteration and the total time is decreased with iteration when the engineering period is the shortest. The algorithm is convenient rather than drawing the PERT figure, only the time of every person doing different jobs and the sequential order of all the jobs are inputted, the optimal assignment solution, the shortest engineering period and the earliest start time and relaxation time of every job can be obtained.

Keywords assignment problem; PERT problem; A-PERT problem; Floyd algorithm; earliest starting time; relaxation time

1 引 言

研究如何把具有紧前紧后关系的任务集分配给现有的人员(或设备),在保证完成任务集的总工期最短的前提下,使总用时最少,是一种充分利用现有

的人力和物力,最大限度地提高工作效率的优化问题.这样的优化问题,在生产调度、机械加工以及工程计划制定与管理等活动中,无疑有着重要的应用价值.但是要解决这样的问题却并不容易,文献[1]证明了使总工期最短的问题是一个 NP-难问题,不存在多项式时间的算法,在问题规模较大的情况下,

很难获得精确最优解. 因此, 人们对这类问题的研究, 普遍着眼于寻找近似最优解或称满意解的算法以及如何提高近似最优解的精度和计算效率. 如, 文献[2] 给出了一种 MCP 近似算法; 文献[3] 给出了一种 ETF 近似算法; 文献[4] 给出了一种 DLS 近似算法; 文献[5] 总结分析了文献[2-3], 给出了一种 BDCP 近似算法; 文献[6-7] 又在上述近似算法的基础上, 给出了新的近似算法; 文献[8-9] 则给出了不同的遗传算法. 研究这类问题的近似算法的文献还有很多, 但是, 却很难找到研究这类问题的精确最优解的文献. 另外, 目前对这类问题的研究都集中在如何获取最短的总工期的问题上, 而没有注意到使总工期最短的分配方案通常会有多种, 而且, 使总工期达到最短的不同分配方案的总用时往往不同, 甚至有一定的差异, 举一个简单例子:

某项工程由 3 项工作构成, 各工作间的紧前、紧后关系如图 1.

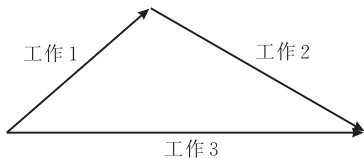


图 1 工作关系示意图

已知 3 个人承担这 3 项工作的用时情况见表 1 所示.

表 1 3 人承担 3 项工作的用时

人	工作用时/周		
	工作 1	工作 2	工作 3
甲	8	3	8
乙	2	7	11
丙	8	9	13

通过穷举, 可以得到所有分配方案下的总用时与总工期, 见表 2.

表 2 总工期和总用时

	工作 1	工作 2	工作 3	总工期/周	总用时/周
方案 1	甲	8	丙	13	15
方案 2	甲	8	乙	11	17
方案 3	乙	2	甲	3	13
方案 4	乙	2	丙	9	11
方案 5	丙	8	甲	3	11
方案 6	丙	8	乙	7	15

从表 2 中可看出, 按第 4、5 套方案进行分配, 总工期最短, 为 11 周, 但是, 第 4 套分配方案的总用时为 19 周, 比第 5 套分配方案的总用时 22 周要少花费 3 周时间, 因此, 选择第 4 套分配方案, 不但能使

总工期达到最短, 而且可以使总用时相对较少.

为此, 本文提出了一种如何寻找相关任务的分配方案, 使总工期达到最短的情况下, 总用时最少. 本文将这种问题称为 A-PERT 问题. 无疑, 这是一个新的、有意义的现实问题.

2 A-PERT 问题的特征及其数学模型

A-PERT 问题的完整描述如下:

某项工程由 n 项工作构成, 各工作之间具有已知的紧前、紧后关系, 现有 m 个人可参与这项工程. 规定每项工作只能由一个人承担. 已知第 i 个人完成第 j 项工作需用时 C_{ij} ($i=1, 2, \dots, m; j=1, 2, \dots, n$). 研究: 要完成这项工程中的所有工作应如何进行分配, 才能够使总工期最短, 并在此条件下, 使用于所有工作上的时间之和最少, 以及在这种要求下的总工期、各项工作的最早开工时间和松弛时间.

A-PERT 问题涉及到 $m < n, m = n, m > n$ 三种情况, 统一的要求是每项工作只能由一个人承担, 不同的要求是, $m < n$ 时, 每个人至少承担一项工作; $m = n$ 时, 每个人只承担一项工作; $m > n$ 时, 每个人至多承担一项工作.

设 $x_{ij} = \begin{cases} 1, & \text{第 } i \text{ 人承担第 } j \text{ 项工作} \\ 0, & \text{否则} \end{cases}, i = 1,$

$2, \dots, m; j = 1, 2, \dots, n.$

用 $f[(C_{ij}x_{ij})_{m \times n}]$ 表示按 x_{ij} ($i=1, 2, \dots, m; j=1, 2, \dots, n$) 进行分配时所需总工期. A-PERT 问题的数学模型为

$$\begin{aligned} \min & \sum_{i=1}^m \sum_{j=1}^n C_{ij} x_{ij} \\ \text{s. t. } & \begin{cases} \min f[(C_{ij}x_{ij})_{m \times n}] \\ \alpha \leq \sum_{j=1}^n x_{ij} \leq \beta, \quad i = 1, 2, \dots, m \\ \sum_{i=1}^m x_{ij} = 1, \quad j = 1, 2, \dots, n \\ x_{ij} \in \{0, 1\}, \quad i = 1, 2, \dots, m; j = 1, 2, \dots, n \end{cases} \end{aligned}$$

其中, α, β 为参变量. 当 $m \leq n$ 时, $\alpha = 1, \beta = n - m + 1$; 当 $m > n$ 时, $\alpha = 0, \beta = 1$.

这是一个双层 0-1 整数规划, 虽然可以用穷举法求解, 即, 计算出所有不同分配方案下的总工期, 通过比较, 可以选出总工期最短时总用时最少的分配方案(最优解). 但是这样的方法计算量太大, 在 $m < n$ 时, 要计算 $m! \cdot C_m^n$ 次 PERT 问题; 在 $m = n$ 时, 要计算 $n!$ 次 PERT 问题; 在 $m > n$ 时, 要计算

$n! \cdot C_m^n$ 次 PERT 问题. 所以, 在 A-PERT 问题的规模较大时, 这种方法显然是不可行的.

本文将针对 $m=n$ 情况, 介绍一种借助 Floyd 算法规则求解 A-PERT 问题的精确算法, 以此为基础, 有利于进一步研究 $m < n$ 和 $m > n$ 两种情况下的 A-PERT 问题.

3 A-PERT 问题的算法理论

从上述 A-PERT 问题的描述可以看出: 在 A-PERT 问题所给条件下, 如果只求总用时最少的分配方案, 则涉及的便是经典分配问题; 如果给定了分配方案, 再求总工期及各项工作的最早开工时间和松弛时间, 则涉及的便是经典的计划评审技术问题, 又称 PERT 问题. 因此, 求解 A-PERT 问题, 可以借鉴文献[10]中求解经典分配问题的方法: 按照一定的规律, 反复从一种分配方案变换到另一种总用时更少的分配方案, 直到最终获得总用时最少的分配方案. 即, 按照以下思路解决 A-PERT 问题:

按照一定的规律, 反复从一种分配方案变换到另一种总工期更短或总工期不增的情况下总用时更少的分配方案, 直到最终获得总工期最短的情况下总用时最少的分配方案.

要实现这样的目的, 关键是找出上面提到的规律. 另外, 要使算法具有实用的计算效率, 应尽量降低迭代过程中出现的每一种分配方案下的总工期和总用时的计算量. 研究表明, 再借鉴文献[11]中求解 PERT 问题的算法, 便可以实现这一目的, 达到这样的要求. 为此, 先给出文献[10-11]中的相关理论.

定理 1. 设 $R(i) \in \{1, 2, \dots, n\} (i=1, 2, \dots, n)$ 为上述问题的一个可行分配方案 (表示安排第 i 个人承担第 $R(i)$ 项工作), $\Delta_{ij} = C_{jR(i)} - C_{iR(i)}$, $p_{ij} = j(i, j=1, 2, \dots, n)$ ($C_{iR(i)}$ 为第 i 个人承担第 $R(i)$ 项工作的用时, $C_{jR(i)}$ 为第 j 个人承担第 $R(i)$ 项工作的用时, p_{ij} 是用于记录人员调整的方法), 则通过下面两步运算:

1. 若 $\Delta_{ik} + \Delta_{kj} < \Delta_{ij}$, 则 $\Delta_{ij} \leftarrow \Delta_{ik} + \Delta_{kj}$, $p_{ij} \leftarrow p_{ik}$; 否则 $(\Delta_{ik} + \Delta_{kj} \geq \Delta_{ij}) \Delta_{ij}$ 与 p_{ij} 都不变 ($i, j=1, 2, \dots, n$), 转到步 2.
2. 求 $\min\{\Delta_{ii} \mid 1 \leq i \leq n\} \triangleq \lambda$, 当 $\lambda=0, k < n$ 时, $k \leftarrow k+1$, 转到步 1.

可获得以下信息:

- (1) 当 $\lambda < 0$ 时, 意味着找到了一个可减少总用时的循环调整方案: 记 $\Delta_{uu} = \lambda, u_0 \leftarrow u$,

$$\textcircled{1} v \leftarrow p_{u_0 u}, w \leftarrow R(v), R(v) \leftarrow R(u_0);$$

② 在 $v \neq u$ 时, $R(u_0) \leftarrow w, u_0 \leftarrow v$, 转到 ①, 直到 $v=u$ 为止.

(2) 当 $k=1, 2, \dots, n$ 的过程中, 始终 $\lambda=0$, 意味着当前的分配方案已经是总用时最少的分配方案.

证明. 参见文献[10].

定义 1. 已知一项工程由 n 个具有紧前、紧后关系的工作构成, 第 i 项工作需用时 $t_i (i=1, 2, \dots, n)$. 若第 $i \in \{1, 2, \dots, n\}$ 项工作无紧前工作时, 则令第 0 项工作为其紧前工作; 若第 $j \in \{1, 2, \dots, n\}$ 项工作无紧后工作时, 则令第 $n+1$ 项工作为其紧后工作, $t_0 = t_{n+1} = 0$. 设

$$d_{ij} = \begin{cases} t_i, & \text{第 } j \text{ 项工作是第 } i \text{ 项工作的紧后工作} \\ 0, & i = j \\ -\infty, & \text{否则} \end{cases},$$

$$i, j = 0, 1, \dots, n+1,$$

则称矩阵 $(d_{ij})_{(n+2) \times (n+2)}$ 为初始 PERT 矩阵, 对应的网络称为复线 PERT 网络.

显然, 一个初始 PERT 矩阵对应着一个有向网络, 第 i 项工作对应着第 i 个节点 ($i \in \{0, 1, \dots, n+1\}$), 节点 i 到相邻可达的节点 j 的边长为第 i 项工作的用时 t_i . 节点 i 不相邻可达节点 j 时, 令 $d_{ij} = -\infty (i, j \in \{0, 1, \dots, n+1\})$ 是为了便于计算.

定义 2. 若矩阵 $(d_{ij})_{(n+2) \times (n+2)}$ 中的元素 $d_{ij} (i, j=0, 1, \dots, n+1)$ 表示复线 PERT 网络中节点 i 到节点 j 的最长路值, 则称该矩阵为最优 PERT 矩阵.

定理 2. 设 $(d_{ij})_{(n+2) \times (n+2)}$ 为初始 PERT 矩阵, 则通过下列运算:

1. $k \leftarrow 0$.
2. 若 $d_{ik} + d_{kj} > d_{ij}$, 则 $d_{ij} \leftarrow d_{ik} + d_{kj}$; 否则 (即 $d_{ik} + d_{kj} \leq d_{ij}$) d_{ij} 不变 ($i, j=0, 1, \dots, n+1$), 转到步 3.
3. 若 $k < n+1$, 则 $k \leftarrow k+1$, 转到步 2; 否则 (即 $k = n+1$), 输出 $(d_{ij})_{(n+2) \times (n+2)}$. 得到的矩阵 $(d_{ij})_{(n+2) \times (n+2)}$ 是最优 PERT 矩阵, 且有下列结果:
 - (1) 第 j 项工作的最早开工时间为 $d_{0j} (j=1, 2, \dots, n)$;
 - (2) 第 j 项工作的最早完工时间为 $d_{0j} + t_j (j=1, 2, \dots, n)$;
 - (3) 完成该项工程所需总工期为 $d_{0, n+1}$;
 - (4) 第 j 项工作的松弛时间为 $d_{0, n+1} - d_{0j} - d_{j, n+1} \triangleq \delta_j (j=1, 2, \dots, n)$;
 - (5) 第 j 项工作的最迟开工时间为 $d_{0j} + \delta_j = d_{0, n+1} - d_{j, n+1} (j=1, 2, \dots, n)$;
 - (6) 第 j 项工作的最迟完工时间为

$$d_{0j} + t_j + \delta_j = d_{0,n+1} - d_{j,n+1} + t_j (j = 1, 2, \dots, n).$$

证明. 参见文献[11].

定理 1 给出了一种寻找总用时最少的分配方案的方法, 定理 2 给出了一种在每一项工作的用时都确定的情况下, 获取总工期及各项工作最早开工时间和松弛时间的方法. 当某一项工作的用时被改变, 其它相关工作的最早开工时间是否会发生变化, 如何变化, 具有以下规律.

定理 3. 设 $(d_{ij})_{(n+1) \times (n+1)}$ 为最优 PERT 矩阵. 若 $d_{ik} \leq 0 (0 \leq i \leq n+1)$, 则改变第 k 项工作的用时, 不会影响第 i 项工作的最早开工时间.

证明. 由最优 PERT 矩阵知, $d_{ik} \leq 0$ 时, 意味着第 k 项工作不是第 i 项工作的后继工作, 所以, 改变第 k 项工作的用时, 不会影响第 i 项工作的最早开工时间. 证毕.

定理 4. 设第 i 项工作的最早开工时间为 d_{0i} , 用时为 $t_i (i=1, 2, \dots, n+1)$,

$$O_{ij} \leftarrow \begin{cases} 1, & \text{第 } j \text{ 项工作是第 } i \text{ 项工作的紧后工作} \\ 0, & \text{否则} \end{cases},$$

$$i, j = 0, 1, 2, \dots, n+1,$$

又设第 s 项工作是第 k 项工作的任意一个紧后工作 $(0 \leq k \leq n+1)$,

$\max\{d_{0i} + t_i \mid O_{is} = 1, i \neq k, 1 \leq i \leq n+1\} \triangleq L_s$, 则当第 k 项工作的用时改变 Δ_k 时, 下列结论成立:

(1) L_s 不存在时, 第 k 项工作的紧后工作 $j \in \{j \mid O_{kj} = 1, 1 \leq j \leq n+1\}$ 的最早开工时间变为 $d_{0k} + t_k + \Delta_k$, 最早开工时间的改变量为 Δ_k .

(2) L_s 存在时, 第 k 项工作的紧后工作 $j \in \{j \mid O_{kj} = 1, 1 \leq j \leq n+1\}$ 的最早开工时间变为 $\max\{L_s, d_{0k} + t_k + \Delta_k\}$, 最早开工时间的改变量为 $\max\{d_{0k} + t_k + \Delta_k - L_s, 0\}$.

证明. (1) L_s 不存在时, 说明第 k 项工作是其紧后工作 $j \in \{j \mid O_{kj} = 1, 1 \leq j \leq n+1\}$ 的唯一紧前工作, 所以第 k 项工作完成后, 第 k 项工作的紧后工作即可开工, 而第 k 项工作的最早开工时间为 d_{0k} , 用时为 $t_k + \Delta_k$, 所以, 第 k 项工作的紧后工作的最早开工时间为 $d_{0k} + t_k + \Delta_k$.

(2) L_s 存在时, 说明第 k 项工作不是其紧后工作 $j \in \{j \mid O_{kj} = 1, 1 \leq j \leq n+1\}$ 的唯一紧前工作, 所以只有当第 $j \in \{j \mid O_{kj} = 1, 1 \leq j \leq n+1\}$ 项工作的所有紧前工作都完成后, 第 j 项工作才能开工, 而 $\max\{L_s, d_{0k} + t_k + \Delta_k\}$ 是第 j 项工作的完工时间最晚的紧前工作的完工时间, 所以, 第 k 项工作的紧后工作的最早开工时间为 $\max\{L_s, d_{0k} + t_k + \Delta_k\}$.

推论. 在定理 4 的条件下, 当第 k 项工作的最早开工时间改变 Δ_k 时, 下列结论成立:

(1) L_s 不存在时, 第 k 项工作的紧后工作 $j \in \{j \mid O_{kj} = 1, 1 \leq j \leq n+1\}$ 的最早开工时间变为 $d_{0k} + t_k + \Delta_k$, 最早开工时间的改变量为 Δ_k .

(2) L_s 存在时, 第 k 项工作的紧后工作 $j \in \{j \mid O_{kj} = 1, 1 \leq j \leq n+1\}$ 的最早开工时间变为 $\max\{L_s, d_{0k} + t_k + \Delta_k\}$, 最早开工时间的改变量为 $\max\{d_{0k} + t_k + \Delta_k - L_s, 0\}$.

证明. 与定理 4 完全类似, 略.

如果第 i 项工作的最早开工时间改变 Δ_i 时, 我们引入

$$u_j = \begin{cases} 1, & \text{第 } j \text{ 项工作的紧后工作的} \\ & \text{最早开工时间未修改完} \\ 0, & \text{否则} \end{cases},$$

则根据定理 3 和定理 4 及其推论可知, 在第 k 项工作的用时改变 Δ_k 后, 第 k 项工作的所有前期工作的最早开工时间不变, 而所有后期工作 α 的最早开工时间可通过下面运算获得:

1. $t_k \leftarrow t_k + \Delta_k$.
2. 求 $\min\{i \mid O_{ki} = 1, 1 \leq i \leq n+1\} \triangleq s$.
3. 求 $\max\{d_{0i} + t_i \mid O_{is} = 1, i \neq k, 1 \leq i \leq n\} \triangleq L_s, \alpha \leftarrow 1$.
4. 若 L_s 不存在, 则 $T \leftarrow d_{0k} + t_k$; 否则 (即 L_s 存在) $T \leftarrow \max\{d_{0k} + t_k, L_s\}$.
5. 若 $O_{ka} = 1$, 则 $d_{0a} \leftarrow T, u_a \leftarrow 1$, 转到步 6; 否则 (即 $O_{ka} = 0$) 直接转到步 6.
6. 若 $\alpha < n+1$, 则 $\alpha \leftarrow \alpha + 1$ 转到步 5; 否则 (即 $\alpha = n+1$) $u_k \leftarrow 0, \alpha \leftarrow 1$, 转到步 7.
7. 若 $u_a = 1$, 则 $k \leftarrow \alpha$ 转到步 2; 否则 (即 $u_a = 0$) 转到步 8.
8. 若 $\alpha < n$, 则 $\alpha \leftarrow \alpha + 1$ 转到步 7; 否则 (即 $\alpha = n$) 停.

为便于论述, 用 $\Delta_k \oplus d_{0\alpha}$ 表示第 k 项工作用时改变 Δ_k 后, 由上述运算步 1~8 得到的第 α 项工作的最早开工时间, 则按 Floyd 算法规则执行 \oplus 运算, 可获得以下运算规律.

定理 5. 设 C_{ij} 为第 i 人做第 j 项工作的用时 $(i, j=1, 2, \dots, n)$, $R(i) (i=1, 2, \dots, n)$ 为一个可行分配方案, $(d_{ij})_{(n+2) \times (n+2)}$ 是对应该分配方案的最优 PERT 矩阵, $\Delta_{ij} = C_{jR(i)} - C_{iR(i)}$ 表示将第 i 人承担第 $R(i)$ 项工作改为第 j 人承担第 $R(i)$ 项工作后, 造成第 $R(i)$ 项工作的用时改变量 $(i, j=1, 2, \dots, n)$, $T_{ij}(s) = \Delta_{ij} \oplus d_{0s}$ 表示第 $R(i)$ 项工作用时改变 Δ_{ij} 后, 第 s 项工作的最早开工时间 $(i, j=1, 2, \dots, n; s=1, 2, \dots, n+1)$, $\Delta L_{ij} = T_{ij}(n+1) - d_{0,n+1}$ 表示第 $R(i)$ 项工作用时改变 Δ_{ij} 后, 总工期的改变量 $(i, j=1, 2, \dots, n)$, 再引入人员调换记录 $p_{ij} = j (i, j=1,$

2, ..., n), 则通过下面两步运算:

1. 若 $\Delta_{ik} \oplus T_{kj}(n+1) < T_{ij}(n+1)$ 时, 则 $T_{ij}(s) \leftarrow \Delta_{ik} \oplus T_{kj}(s)$, $s=1, 2, \dots, n+1$; $\Delta L_{ij} \leftarrow T_{ij}(n+1) - d_{0,n+1}$, $p_{ij} \leftarrow p_{ik}$; 否则 $T_{ij}(s) (s=1, 2, \dots, n+1)$, ΔL_{ij} , p_{ij} 都不变 ($i, j=1, 2, \dots, n$), 转到步 2.

2. 求 $\min\{\Delta L_{ii} \mid 1 \leq i \leq n\} \triangleq \lambda$, 当 $\lambda=0$, $k < n$ 时, $k \leftarrow k+1$, 转到步 1.

可获得以下信息:

(1) 当 $\lambda < 0$ 时, 意味着找到了一个可缩短总工期的循环调整方案: 记 $\Delta L_{uu} = \lambda$, $u_0 \leftarrow u$,

① $v \leftarrow p_{u_0 u}$, $w \leftarrow R(v)$, $R(v) \leftarrow R(u_0)$,

② 在 $v \neq u$ 时, $R(u_0) \leftarrow w$, $u_0 \leftarrow v$, 转到①, 直到 $v = u$ 为止.

(2) 当 $k=1, 2, \dots, n$ 的过程中, 始终 $\lambda=0$, 意味着当前的分配方案已经是总工期最短的分配方案.

证明. (1) 因为, 如果存在一个分配方案: 安排第 i_1 人做第 $R(1)$ 项工作, 第 i_2 人做第 $R(2)$ 项工作, ..., 第 i_n 人做第 $R(n)$ 项工作时的总工期比安排第 i 人做第 $R(i) (i=1, 2, \dots, n)$ 项工作时的总工期短, 则 i_1, i_2, \dots, i_n 是 $1, 2, \dots, n$ 的一个全排列. 为了便于论述, 不妨设 (其它情况证明类似).

$$i_1 = 2, i_2 = 3, \dots, i_{\alpha-1} = \alpha, i_\alpha = 1,$$

$$i_{\alpha+1} = \alpha + 1, \dots, i_n = n,$$

则用第 i_1 人替换第 1 人, 第 i_2 人替换第 2 人, ..., 第 i_α 人替换第 α 人, 总工期便会缩短. 因此, 以下必有一组按次序执行的运算式成立:

$$\left\{ \begin{array}{l} \text{因为 } \Delta_{\alpha-1,\alpha} \oplus T_{\alpha 1}(n+1) < T_{\alpha-1,1}(n+1), \\ \text{所以 } T_{\alpha-1,1}(n+1) \leftarrow \Delta_{\alpha-1,\alpha} \oplus \\ \quad T_{\alpha 1}(n+1), p_{\alpha-1,1} \leftarrow p_{\alpha-1,\alpha}, \\ \text{因为 } \Delta_{\alpha-2,\alpha-1} \oplus T_{\alpha-1,1}(n+1) < T_{\alpha-2,1}(n+1), \\ \text{所以 } T_{\alpha-2,1}(n+1) \leftarrow \Delta_{\alpha-2,\alpha-1} \oplus \\ \quad T_{\alpha-1,1}(n+1), p_{\alpha-2,1} \leftarrow p_{\alpha-2,\alpha-1}, \\ \dots \\ \text{因为 } \Delta_{12} \oplus T_{21}(n+1) < T_{11}(n+1), \\ \text{所以 } T_{11}(n+1) \leftarrow \Delta_{12} \oplus T_{21}(n+1), p_{11} \leftarrow p_{12}, \\ \text{因为 } \Delta_{\alpha 1} \oplus T_{12}(n+1) < T_{\alpha 2}(n+1), \\ \text{所以 } T_{\alpha 2}(n+1) \leftarrow \Delta_{\alpha 1} \oplus T_{12}(n+1), p_{\alpha 2} \leftarrow p_{\alpha 1}, \\ \text{因为 } \Delta_{\alpha-1,\alpha} \oplus T_{\alpha 2}(n+1) < T_{\alpha-1,2}(n+1), \\ \text{所以 } T_{\alpha-1,2}(n+1) \leftarrow \Delta_{\alpha-1,\alpha} \oplus \\ \quad T_{\alpha 2}(n+1), p_{\alpha-1,2} \leftarrow p_{\alpha-1,\alpha}, \\ \dots \\ \text{因为 } \Delta_{23} \oplus T_{32}(n+1) < T_{22}(n+1), \\ \text{所以 } T_{22}(n+1) \leftarrow \Delta_{23} \oplus T_{32}(n+1), p_{22} \leftarrow p_{23}, \\ \dots \end{array} \right.$$

$$\left\{ \begin{array}{l} \text{因为 } \Delta_{\alpha-2,\alpha-1} \oplus T_{\alpha-1,\alpha}(n+1) < T_{\alpha-2,\alpha}(n+1), \\ \text{所以 } T_{\alpha-2,\alpha}(n+1) \leftarrow \Delta_{\alpha-2,\alpha-1} \oplus \\ \quad T_{\alpha-1,\alpha}(n+1), p_{\alpha-2,\alpha} \leftarrow p_{\alpha-2,\alpha-1}, \\ \text{因为 } \Delta_{\alpha-3,\alpha-2} \oplus T_{\alpha-2,\alpha}(n+1) < T_{\alpha-3,\alpha}(n+1), \\ \text{所以 } T_{\alpha-3,\alpha}(n+1) \leftarrow \Delta_{\alpha-3,\alpha-2} \oplus T_{\alpha-2,\alpha}(n+1), \\ \quad p_{\alpha-3,\alpha} \leftarrow p_{\alpha-3,\alpha-2}, \\ \dots \\ \text{因为 } \Delta_{\alpha 1} \oplus T_{1\alpha}(n+1) < T_{\alpha\alpha}(n+1), \\ \text{所以 } T_{\alpha\alpha}(n+1) \leftarrow \Delta_{\alpha 1} \oplus T_{1\alpha}(n+1), p_{\alpha\alpha} \leftarrow p_{\alpha 1}. \end{array} \right.$$

因为, 这 α 组按次序执行的运算, 都属于定理中的步 1、2 两步运算的搜索范围, 只是执行的是能够成立的一组, 又因为运算前 $T_{ii}(n+1) = \Delta_{ii} \oplus d_{0,n+1} = d_{0,n+1} (i=1, 2, \dots, n)$, 所以执行步 1、2 的结果是 $\Delta L_{ii} = T_{ii}(n+1) - d_{0,n+1} (i=1, 2, \dots, n)$ 中必有一个小于 0, 所以, $\min\{\Delta L_{ii} \mid 1 \leq i \leq n\} = \lambda < 0$, 用 $p_{ij} (i, j=1, 2, \dots, n)$ 记录到的调整过程, 按定理中的①、②进行调整, 便得到一个总工期更短的分配方案.

(2) 对(1)的证明已揭示, 只要存在比当前分配方案下的总工期更短的分配方案, 定理中的步 1、2 两步运算一定能将其找出来. 同时也表明, 如果当前的分配方案已经使总工期达到最短了, 则在 $k=1, 2, \dots, n$ 的过程中, 按定理中的步 1、2 两步运算, 始终不会出现 $\Delta_{ik} \oplus T_{ki} < T_{ii}(n+1) (i=1, 2, \dots, n)$, 所以, $\Delta L_{ii} = 0$ 始终不变, 故 $\min\{\Delta L_{ii} \mid 1 \leq i \leq n\} = 0$, 即 $\lambda=0$ 始终成立. 证毕.

定理 6. 设分配方案 $R(i) (i=1, 2, \dots, n)$ 使总工期达到了最短, $\Delta_{ij} = C_{jR(i)} - C_{iR(i)}$, $\delta_{ij} \leftarrow \Delta_{ij} (i, j=1, 2, \dots, n)$, 则通过下面两步运算:

1. 若 $\Delta_{ik} \oplus T_{kj}(n+1) = T_{ij}(n+1)$, 且 $\delta_{ik} + \delta_{kj} < \delta_{ij}$, 则 $T_{ij}(s) \leftarrow \Delta_{ik} \oplus T_{kj}(s) (s=1, 2, \dots, n+1)$, $\delta_{ij} \leftarrow \delta_{ik} + \delta_{kj}$, $p_{ij} \leftarrow p_{ik}$;
否则 $T_{ij}(s) (s=1, 2, \dots, n+1)$, δ_{ij} , p_{ij} 都不变 ($i, j=1, 2, \dots, n$), 转到步 2.

2. 求 $\min\{\delta_{ii} \mid 1 \leq i \leq n\} \triangleq \lambda$, 当 $\lambda=0$, $k < n$ 时, $k \leftarrow k+1$ 转到步 1.

可获得以下信息:

(1) 当 $\lambda < 0$ 时, 意味着找到了一个可减少总用时的循环调整方案: 记 $\Delta L_{uu} = \lambda$, $u_0 \leftarrow u$,

① $v \leftarrow p_{u_0 u}$, $w \leftarrow R(v)$, $R(v) \leftarrow R(u_0)$,

② 在 $v \neq u$ 时, $R(u_0) \leftarrow w$, $u_0 \leftarrow v$, 转到①, 直到 $v = u$ 为止.

(2) 当 $k=1, 2, \dots, n$ 的过程中, 始终 $\lambda=0$, 意味着在保持总工期不变的前提下, 当前的分配方案已经使总用时达到最少.

证明. 类似于定理 5 的证明, 略.

4 A-PERT 问题的程序算法

上述 A-PERT 问题的算法理论揭示, 求解 A-PERT 问题, 可以按图 2 所示的模块化的算法流程图进行.

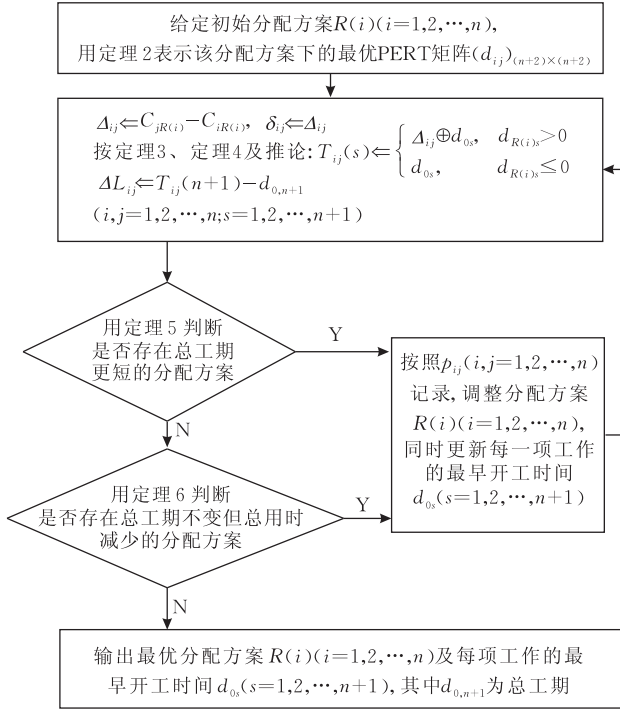


图 2 算法流程

求解 A-PERT 问题的精确最优解的具体算法步骤如下:

1. 输入已知数据:
 $C_{ij} (i, j = 1, 2, \dots, n)$,
 $O_{ij} \leftarrow \begin{cases} 1, & \text{第 } j \text{ 项工作是第 } i \text{ 项工作的紧后工作} \\ 0, & \text{否则} \end{cases}$,
 $i, j = 0, 1, 2, \dots, n+1$.
2. 用最小元素法定初始分配方案 $R(i) (i=1, 2, \dots, n)$.
3. 用定理 2 求最优 PERT 矩阵 $(d_{ij})_{(n+2) \times (n+2)}$.
4. $\Delta_{ij} \leftarrow C_{jR(i)} - C_{iR(i)}$, $\delta_{ij} \leftarrow \Delta_{ij} (i, j = 1, 2, \dots, n)$.
5. 按照定理 3 和定理 4, 计算将第 $R(i)$ 项工作改由第 j 人承担后, 第 s 项工作的最早开工时间:

$$T_{ij}(s) \leftarrow \begin{cases} \Delta_{ij} \oplus d_{0s}, & d_{R(i)s} > 0 \\ d_{0s}, & d_{R(i)s} \leq 0 \end{cases},$$

$$i, j = 1, 2, \dots, n; s = 1, 2, \dots, n+1.$$

对应的总工期的改变量:

$$\Delta L_{ij} \leftarrow T_{ij}(n+1) - d_{0,n+1}, i, j = 1, 2, \dots, n.$$

记录调整过程:

$$p_{ij} \leftarrow j (i, j = 1, 2, \dots, n).$$

6. 计算逐步增加改变工作承担者的工作后, 搜寻使总

工期变短或总工期不变时使总用时变少的调整方案:

$$i \leftarrow 1, j \leftarrow 1, k \leftarrow 1.$$

7. 若 $\Delta_{ik} \oplus T_{kj}(n+1) < T_{ij}(n+1)$, 则 $T_{ij}(s) \leftarrow \Delta_{ik} \oplus T_{kj}(s) (s=1, 2, \dots, n+1)$; $\Delta L_{ij} \leftarrow T_{ij}(n+1) - d_{0,n+1}$, $\delta_{ij} \leftarrow \delta_{ik} + \delta_{kj}$, $p_{ij} \leftarrow p_{ik}$, 转到步 9; 否则转到步 8.

8. 若 $\Delta_{ik} \oplus T_{kj}(n+1) = T_{ij}(n+1)$ 且 $\delta_{ik} + \delta_{kj} < \delta_{ij}$, 则 $T_{ij}(s) \leftarrow \Delta_{ik} \oplus T_{kj}(s)$, $s=1, 2, \dots, n+1$, $\delta_{ij} \leftarrow \delta_{ik} + \delta_{kj}$, $p_{ij} \leftarrow p_{ik}$, 转到步 9; 否则直接转到步 9.

9. 若 $j < n$, 则 $j \leftarrow j+1$ 转到步 7; 否则 (即 $j=n$) 转到步 10.

10. 若 $i < n$, 则 $i \leftarrow i+1, j \leftarrow 1$ 转到步 7; 否则 (即 $i=n$) 转到步 11.

11. 判断是否存在缩短总工期的部分工作承担者的循环调整:

$$\text{求 } \min\{\Delta L_{ii} \mid 1 \leq i \leq n\} \triangleq \lambda,$$

- 若 $\lambda < 0$, 则 $u \leftarrow \min\{i \mid \Delta L_{ii} = \lambda, 1 \leq i \leq n\}$, $u_0 \leftarrow u$, 转到步 13; 否则 (即 $\lambda=0$) 转到步 12.

12. 判断是否存在减少总用时的部分工作承担者的循环调整:

$$\text{求 } \min\{\delta_{ii} \mid 1 \leq i \leq n\} \triangleq \lambda,$$

- 若 $\lambda < 0$, 则 $u \leftarrow \min\{i \mid \delta_{ii} = \lambda, 1 \leq i \leq n\}$, $u_0 \leftarrow u$, 转到步 13; 否则 (即 $\lambda \geq 0$) 转到步 15.

13. 调整分配方案:

$$v \leftarrow p_{u_0 u}, w \leftarrow R(v), R(v) \leftarrow R(u_0), d_{0s} \leftarrow T_{uv}(s) (s=1, 2, \dots, n+1).$$

14. 若 $v \neq u$, 则 $R(u_0) \leftarrow w, u_0 \leftarrow v$, 转到步 13; 否则 (即 $v=u$), 转到步 4.

15. 若 $k < n$, 则 $k \leftarrow k+1, i \leftarrow 1, j \leftarrow 1$ 转到步 7; 否则 (即 $k=n$), 输出到最优分配方案 $R(i) (i=1, 2, \dots, n)$ 及各工作的最早开工时间 $d_{0R(i)} (i=1, 2, \dots, n)$.

注: (1) 如果需要各项工作的松弛时间 $\delta_{R(i)} (i=1, 2, \dots, n)$, 可以根据最优分配方案 $R(i) (i=1, 2, \dots, n)$ 下的各项工作用时 $C_{iR(i)} (i=1, 2, \dots, n)$, 用定理 2 获得. 也可从 $d_{0,n+1}$ 开始, 利用 $C_{iR(i)}$ 与 $d_{0R(i)} (i=1, 2, \dots, n)$ 倒着递推出各项工作的松弛时间 $\delta_{R(i)} (i=1, 2, \dots, n)$.

(2) A-PERT 问题算法必然在有限次迭代运算中获得精确最优解, 其理论依据为定理 5 和定理 6.

(3) A-PERT 问题的算法, 是以缩短总工期或总工期不能再缩短时减少总用时为前提, 由一个可行分配方案转换到另一个可行分配方案的迭代算法 (对于 n 个人, n 项工作的分配问题, 称一人做一项工作, 每项工作只由一个人做的分配方案为可行分配方案). 由于该问题的所有不同的可行分配方案共有 $n!$ 种, 所以, 理论上在最坏情况下, 可能要迭代运算 $n!$ 次才能获得最优分配方案. 因此, 按迭代次数

统计, A-PERT 问题算法的复杂度为 $O(n!)$, 属于指数算法. 但是, 算法的初始可行分配方案, 是用最小元素法定出的, 而且迭代运算是按总工期递减或总工期不能缩短时按总用时递减进行的, 所以, 运算中跳过了大量的不能使总工期缩短或总工期达到最短时不能减少总用时的分配方案下相关计算. 因此, 如用单纯形法求解线性规划问题那样, A-PERT 问题算法虽然属于指数算法, 却具有适用的计算效率和稳定的数值结果. 从我们计算过的一些算例也显示, 用本文给出的算法求解 A-PERT 问题, 实际迭代次数都远远少于 $n!$ 次. 如下面一个例子:

例. 已知某项工程含有 8 项工作, 各工作之间

的紧前紧后关系如表 3 所示.

表 3 工作之间的关系

工 作	紧前工作	紧后工作
①	无	③
②	无	④⑤
③	①	⑥
④	②	⑥
⑤	②	⑦
⑥	③④	⑧
⑦	⑤	⑧
⑧	⑥⑦	无

现有 8 个工作组, 每个工作组完成各项工作的用时见表 4.

表 4 每个工作组完成各项工作的用时

工作组	工作用时/单位时间							
	①	②	③	④	⑤	⑥	⑦	⑧
1	12	16	14	21	18	7	11	9
2	6	9	11	19	12	12	10	7
3	14	21	16	12	14	5	23	13
4	15	15	27	21	17	19	16	20
5	9	13	19	20	15	16	11	14
6	23	18	24	10	8	7	15	12
7	15	20	8	14	17	6	9	18
8	12	16	11	9	10	15	14	8

要求每项工作只能由一个工作组承担, 每个工作组只能承担一项工作. 研究: 如何进行工作分配, 使完成该项工程的总工期最短的前提下, 所有工作组的用时之和最少, 并求出这种要求下各项工作的最早开工时间、最迟开工时间、松弛时间.

解. 由算法步 2 和算法步 3 分别得到初始分配方案和当前各项工作的最早开工时间, 见表 5.

表 5 初始分配方案和最早开工时间

	工作组	最早开工时间/单位时间
工作 1	2	0
工作 2	5	0
工作 3	7	6
工作 4	4	13
工作 5	6	13
工作 6	3	34
工作 7	1	21
工作 8	8	39

此时, 总工期为 47, 总用时为 80.

通过算法步 4~15 由一个可行分配方案转换到另一个总工期更短或总工期最短时总用时更少的可行分配方案, 共进行 11 次转换 (这远远小于 $8!$ 次), 便得到最优分配方案及最优工程计划下的各项工作的最早开工时间, 见表 6. 在此基础上, 还可得到最优分配方案下各项工作的松弛时间, 见表 6.

表 6 最终求得的最早开工时间和松弛时间

	工作组	最早开工时间/单位时间	松弛时间/单位时间
工作 1	4	0	0
工作 2	2	0	0
工作 3	7	15	0
工作 4	8	9	5
工作 5	6	9	0
工作 6	3	23	0
工作 7	5	17	0
工作 8	1	28	0

此时, 总工期为 37, 总用时为 74.

5 结束语

在关键作业上增加人力、物力、技术的投入, 可以减少关键作业的用时, 并且, 在一定程度上能达到缩短总工期, 提高生产效率的目的^[12], 但是, 值得注意的是, 使用这种方法必然要加重成本的投入, 更何况并非所有关键作业都能减少用时. 另外, 不论减少多少关键作业的用时, 总工期也不会短于 PERT 网络的次最长路. 因此, 当 PERT 网络的最长路与次最长路相差无几时, 通过减少关键作业的用时来缩短总工期, 效果是微不足道的. 例如, 某个工程由 9 个作业项目构成, 这 9 个作业项目间的紧前紧后关

系如图 3 所示, 每条边上的数字, 表示对应作业的用时。

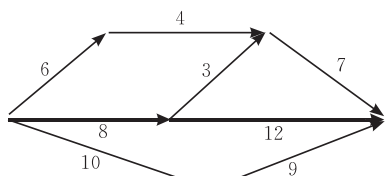


图 3

不难看出该工程的总工期为 20, 不论将关键作业(粗箭线对应的作业)的用时如何缩短, 新的总工期都不会短于 19。

本文研究的 A-PERT 问题, 是一种不额外投入人力、物力和财力的情况下, 充分利用现有资源, 最大限度地提高生产效率优化问题。所给算法, 不但能获得使总工期最短的分配方案, 而且能保证总工期最短的情况下, 使总用时最少。这种算法有稳定的计算性能和适用的计算效率, 可在企业特别是大型企业的生产任务分配与进度计划制定中发挥重要的作用。另外, 由于这种算法获得的是精确最优解, 所以, 可以供各种近似算法进行近似最优解的精确程度比较, 这比目前许多文献采用的近似最优解与近似最优解的优劣比较更客观, 更能说明问题。

参 考 文 献

- [1] Garey M R, Johnson Ds. Computers and Intractability—A Guide to the Theory of NP-Completeness. New York: Freeman W. H. and Co., 1979
- [2] Wu M Y, Gajski D D. Hypertool. A programming aid for message-passing systems. IEEE Transactions on Parallel and Distributed Systems, 1990, 1(3): 330-343
- [3] Hwang J J, Chow Y C, Anger F D, Lee C Y. Scheduling precedence graphs in systems with interprocessor communication times. SIAM Journal on Computing, 1989, 18(2): 244-257
- [4] Sih G C, Lee E A. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. IEEE Transactions on Parallel and Distributed Sys-

tems, 1993, 4(2): 75-87

- [5] Shi Wei, Zheng Wei-Min. The balanced dynamic critical path scheduling algorithm of dependent task graphs. Chinese Journal of Computers, 2001, 24(9): 991-997(in Chinese)
(石威, 郑伟民. 相关任务图的均衡动态关键路径调度算法. 计算机学报, 2001, 24(9): 991-997)
- [6] Shang Ming-Sheng. Efficient algorithm for scheduling dependent task graphs on multi-processor system. Computer Engineering, 2005, 31(14): 18-20, 29(in Chinese)
(尚明生. 相关任务图的一种有效并行调度算法. 计算机工程, 2005, 31(14): 18-20, 29)
- [7] Jiang Ting-Yao, Li Qing-Hua. A scheduling algorithm for DAG task graphs. Journal of Chinese Computer Systems, 2003, 24(10): 1796-1799(in Chinese)
(蒋延耀, 李庆华. DAG 任务图的一种调度算法. 小型微型计算机系统, 2003, 24(10): 1796-1799)
- [8] Kwok Yu-Kwong, Ishfaq Ahumad. Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm. Journal of Parallel and Distributed Computing, 1997, 47(1): 58-77
- [9] Zhong Qiu-Xi, Xie Tao, Chen Huo-Wang. Task allocation & scheduling by computational model of coevolution. Chinese Journal of Computers, 2001, 24(3): 308-314(in Chinese)
(钟求喜, 谢涛, 陈火旺. 任务分配与调度的共进化方法. 计算机学报, 2001, 24(3): 308-314)
- [10] Guo Qiang. New iterative algorithm for an assignment problem. Systems Engineering and Electronics, 2004, 26(12): 1915-1916, 1949(in Chinese)
(郭强. 分配问题的一种新的迭代算法. 系统工程与电子技术, 2004, 26(12): 1915-1916, 1949)
- [11] Guo Qiang. A new algorithm of PERT. Mathematics in Practice and Theory, 2003, 33(2): 48-51(in Chinese)
(郭强. PERT 问题的新算法. 数学的实践与认识, 2003, 33(2): 48-51)
- [12] Yin Jian-Wei, Han Wei-Li, Chen Gang, DONG Jin-Xiang. Fair algorithm of goal-driven time limit modification for a project. Journal of Computer-Aided Design & Computer Graphics, 2002, 14(3): 270-274(in Chinese)
(尹建伟, 韩伟力, 陈刚, 董金祥. 项目工期调整的公平负担算法. 计算机辅助设计与图形学学报, 2002, 14(3): 270-274)

GUO Qiang, born in 1961, associate professor. His research interests include the theory and algorithm of optimization, operational research and mathematics programming in network.



Background

The problem of network plan based on the assignment of dependent tasks is the integration and generalization of assignment problem and PERT problem (Program Evaluation and Review Technique), and belong to operational research field. The problem generally is described as follow: The engineering consists of n tasks with precedence and successor relationship. Each task can only be taken by one person, and every person must take at least one task when $m(<n)$ persons are assigned to complete those n tasks. Let C_{ij} is the time that i th person takes to complete j th task ($i=1,2,\dots,m; j=1,2,\dots,n$). The objective of the research is how assigning those jobs to every person so that the engineering period is the shortest, and in this premise the total time of completing all those jobs is the least. Finally the algorithm must give the earliest start time and relaxation time for each task.

Such problem has important application in production planning and human resources management for modern enterprises, especially large enterprises. Almost studies at home and abroad are on finding the assignment plan of shortest engineering period in the circumstances $m=n$. But they do not

involve that further reduce the total time of all the tasks when the engineering period is shortest. It is well known such assignment problem is the NP-hard problem. Therefore, the existing research results are through the use of the approximate algorithm. According to the case of $m=n$, this paper gives a precise algorithm, which not only the shortest engineering period can be solved, but also ensure the total time of all the tasks least when the engineering period is shortest. Although this method is not polynomial time algorithm, the iterative operations of this algorithm have always been in the direction of shortening the engineering period. When the shortest engineering period is achieved, the algorithm will ensure that the iterative operations are always in the direction of decreasing the total time, and remaining the shortest engineering period not be changed. Therefore, the algorithm has applicable and stable computing performance.

The precise algorithm have not been found for the problem that $m<n$. But the precise algorithm can be obtained when n tasks are mutual independent, it will be given in another paper of the author.