文章编号:1001 - 9081(2010)10 - 2745 - 04

基于关键路径法的软件过程控制模型

高　晓[1],徐高峰[2],钟　勇[1]
(1. 中国科学院 成都计算机应用研究所,成都 610041; 2. 中国石油昆仑天然气利用有限公司,广东 深圳 518054)
(shawngao06@gmail.com)

摘　要:为了增强软件过程控制能力,保障软件产品质量,提出一种基于关键路径法的软件过程控制模型。该模型以过程活动间的网络拓扑结构为基础,在确保工期和关键路径上资源需求的条件下,控制其他活动的开始时间,使得过程中所需的总资源耗费较少,且在此耗费下各活动开始时间较早。在最优耗费的约束下,还给出了一个基于资源竞争链的浮动信息更新算法,以便更新各活动的浮动信息。最后,用实例验证了所提模型的有效性和可行性。

关键词:软件过程控制;关键路径;资源均衡;资源竞争链;浮动时间
中图分类号: TP311.5　　**文献标志码**:A

Software process control model based on critical path method

GAO Xiao[1], XU Gao-feng[2], ZHONG Yong[1]
(1. *Chengdu Institute of Computer Application, Chinese Academy of Sciences, Chengdu Sichuan* 610041, *China*;
2. *PetroChina Kunlun Natural Gas Utilization Company Limited, Shenzhen Guangdong* 518054, *China*)

Abstract: A software process control model based on critical path method was presented for enhancing the software process controlling to guarantee the product quality. The model is based on the topological structure of the process activities. Under the precondition of ensuring the time limit for the project and the resource requirement of the critical activities, the start time of the other activities were controlled by a mathematical model with the target of making the total resource cost less and the starting of the activities much earlier under the optimal resource cost. Under the constraint of the optimal resource cost, an algorithm based on resource competition chain was also promoted to update the float information of the uncritical activities. At last, the experimental results show the model is feasible and effective.

Key words: software process control; critical path; resource leveling; resource competition chain; float time

在软件开发实践中,人们逐步认识到软件产品质量在很大程度上依赖于产品开发过程。由于商业环境、开发技术及开发人员等影响因素的多变性,需要对软件过程进行有效地控制并不断加以改进[1-3]。时间和资源是限制软件过程控制的两个对立面:缩短工期要求投入充足的资源,相反在资源受限的情况下只能延长工期。资源约束的关键路径法侧重于在资源受限的情况下使得工期最短[4-5]。然而,为了抢占商机或满足客户要求,软件项目对工期有更高的要求。并且软件活动中的每个人往往可以承担多种角色,而且随着项目的进行,还可以被灵活地添加或削减。基于约束规划的控制模型,虽然解决了资源利用问题,但是由于资源竞争而导致每个活动都是关键活动,任意一活动的延迟或变动都会影响整个工期[6]。因此,如何在可容忍的工期内使用较少的资源并有效地控制过程中的活动就成为软件企业或管理者所关心的问题。为此,本文提出了一个基于关键路径法的软件过程控制模型,该模型旨在解决两个问题:一是在基于关键路径保障工期的前提下,如何控制非关键活动,使得工期内耗费的总资源最少;二是在最优耗费的约束下,提出基于资源竞争链的浮动信息更新算法,使得工期内的各项活动得到有效控制。

1　基于关键路径法的资源优化模型

1.1　关键路径

关键路径是项目计划中最长的路线,它决定了项目的总耗时间。关键路径法正是通过关键路径来预测项目工期的网络分析,它是一个动态系统,会随着项目的进展不断更新,这对处理软件过程本身的多变性是非常有帮助的。

软件过程活动之间存在着约束,最初的约束主要是输入输出间的制约——后继活动的输入是前驱活动的输出,基于此拓扑结构的关键路径保障了工期的最短。然而这是在资源充足的情况下才能实现的。现实情况往往是资源有限,这样只能尽量满足资源需求的同时容忍一定的工期延迟。在初始的关键路径确定后,用本文所提模型求解基于当前关键路径下非关键活动的最早开始时间,使得工期内总资源消耗最优;在所有活动都确定了开始时间后,便可以得到工期内各时间段资源使用情况,此时可以做进一步的时间和资源间的协调。

如图 1,右侧显示的是各活动的后继活动,粗线标记的 A、C、F、H 是一条关键路径(活动编号后面的数字表示对各类资源的需求量),其余的活动是非关键活动,X 轴是时间轴,Y 轴上显示了各时间段资源使用情况。图中的非关键活动的开始时间是利用本文所提模型求解出的,如此控制活动的开始时间使得在不影响工期的情况下两类资源的总耗费最少。从中可以看到,6、7 时间段内的资源耗费达到了峰值,如果确实无法提供 3 个单位的第一类资源并且可以容忍工期稍稍延迟,便可以将 G 加入到关键路径中使得工期延迟两个时间单位,此时第一类资源的耗费便可以缩减到能够提供的范围之内;对于不便将非关键活动加入到关键路径的情况,可以减少关键活动所耗费的资源而延长其开发的时间。因此,软件过程活动间的约束关系并不是一次确定的,通过本文所提模型

收稿日期:2010 - 04 - 27;**修回日期**:2010 - 07 - 06。　**基金项目**:四川省科技计划项目(2008GZ0003)。
作者简介:高晓(1986 -),男,内蒙古临河人,硕士研究生,主要研究方向:软件过程技术与方法; 徐高峰(1973 -),男,新疆乌鲁木齐人,工程师,主要研究方向:计算机测控、工程自动化; 钟勇(1966 -),男,四川岳池人,研究员,博士生导师,主要研究方向:软件过程技术与方法。

求解，可以将资源使用信息反馈回去，重新调整各活动间的关系，平衡工期和资源耗费。

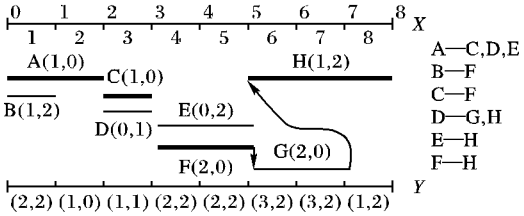


图 1 基于信息反馈的关键路径调整图

1.2 本文的资源优化模型

软件过程是由用于开发和维护产品的一系列有序活动组成的，在确立了活动间的约束关系后，关键路径便可确定。随即各活动的最早、最晚开始时间以及浮动时间等属性也被确定，任何一个活动都可以由它所包含的属性所标识（如图 2）。

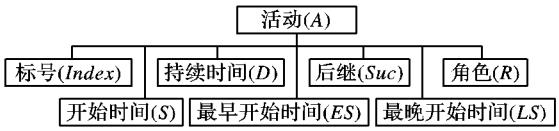


图 2 活动属性

关键活动的开始时间是确定的，而非关键活动的开始时间影响着整个工期内的资源耗费[7]，如何确定非关键活动的开始时间使得工期内资源耗费最少，并且在最优资源耗费下非关键活动的开始时间最早就是本文要解决的问题。与其他工程项目不同，软件过程中所需要的主要资源是人，而其他的工具或资源往往独占地分配给每个人员。因此，人员的投入决定着其他资源的耗费，本文模型中主要考虑人力资源的需求，并且一个活动由一种角色的人员完成，这是符合实际情况的。在描述模型前，首先说明模型中各符号含义。

$AC = \{ac_1, ac_2, \cdots, ac_m\}$ 表示软件过程活动集合；$\boldsymbol{RL} = (\boldsymbol{rl}_1, \boldsymbol{rl}_2, \cdots, \boldsymbol{rl}_m)^{\mathrm{T}}$ 表示活动关系矩阵；$\boldsymbol{rl}_i = (a_{i1}, a_{i2}, \cdots, ac_i m\}$，$a_{ij} = \begin{cases} 1, ac_j \text{ 是 } ac_i \text{ 的后继} \\ 0, \text{其他} \end{cases}$；$\boldsymbol{S} = (s_1, s_2, \cdots, s_m)$ 表示活动的开始时间；$\boldsymbol{DM} = (d_1, d_2, \cdots, d_m)$ 表示活动的持续时间；$\boldsymbol{ESM} = (es_1, es_2, \cdots, es_m)$ 表示最早开始时间；$\boldsymbol{LSM} = (ls_1, ls_2, \cdots, ls_m)$ 表示最晚开始时间；$\boldsymbol{ACC} = (\boldsymbol{acc}_1, \boldsymbol{acc}_2, \cdots, \boldsymbol{acc}_m)^{\mathrm{T}}$ 表示活动资源矩阵，$\boldsymbol{acc}_i = (b_{i1}, b_{i2}, \cdots, b_{im})$，$b_{ij}$ 表示 ac_i 对角色 j 的需求；$\boldsymbol{R} = (r_1, r_2, \cdots, r_m)$，$r_i$ 表示第 i 种角色的总需求；$\boldsymbol{C} = (c_1, c_2, \cdots, c_m)$，$c_i$ 表示第 i 种角色的费用。

基于上述信息，可以将资源优化模型描述如下：

$$\min F = (\boldsymbol{C} \times \boldsymbol{R}^{\mathrm{T}}) \times \sum_{i=1}^{m} ls_i + \sum_{i=1}^{m} s_i \tag{1}$$

$$\boldsymbol{ESM}^{\mathrm{T}} \leqslant \boldsymbol{S}^{\mathrm{T}} \leqslant \boldsymbol{LSM}^{\mathrm{T}} \tag{2}$$

$$((s_i + d_i) \times \boldsymbol{rl}_i)^{\mathrm{T}} \leqslant (a_{i1} \times s_1, \cdots, a_{im} \times s_m)^{\mathrm{T}};\ i = 1, \cdots, m \tag{3}$$

$$\boldsymbol{R}^{\mathrm{T}} \leqslant \left(\sum_{i=1}^{m} \boldsymbol{acc}_i \right)^{\mathrm{T}} \tag{4}$$

$$\left(\sum_{i=1}^{m} (f(s_i, k) \times \boldsymbol{acc}_i) \right)^{\mathrm{T}} \leqslant \boldsymbol{R}^{\mathrm{T}};\ k = 1, 2, \cdots, T_{\max},$$

$$f(s_i, k) = \begin{cases} 1, & s_i \leqslant k \leqslant s_i + d_i \\ 0, & 1 \leqslant k < s_i \text{ 或 } s_i + d_i < k \leqslant T_{\max} \end{cases} \tag{5}$$

上述模型中，式(1)是目标函数，求解工期内对人员耗费的最小值（第一项 $(\boldsymbol{C} \times \boldsymbol{R}^{\mathrm{T}}) \times \sum_{i=1}^{m} ls_i$），并且在此需求下活动的开始时间最早（第二项 $\sum_{i=1}^{m} s_i$）；第一项中的 $\sum_{i=1}^{m} ls_i$ 是一个安全系数，使得不同的人员需求之间有足够大的距离，保证了相同的人员需求下由于活动开始时间的不同而形成的可行解的子集是不相交的。式(2)约束了活动开始时间介于最早和最晚开始时间之间。式(3)保证了前驱活动的结束时间不得晚于后继活动的开始时间。式(4)是人员需求的上限。式(5)约束了整个工期内，任何一个单位时间段内对各类人员需求的总和不得超过可提供的人员总数，$T_{\max}$ 表示工期的结束时间。

不难看出，模型中只有 S、R 是未知的，而且 R 的取值是由 S 所决定的，因此所有可能的 S 可以视为模型的解空间。随着问题规模的扩大，组合爆炸问题不可避免，常规的搜索算法无法适应问题的求解。随机数编码的遗传算法可以解决此类约束满足问题[8]。本文采用改进模拟退火算法[9-10]求解模型：由于从可行解空间产生扰动的个体只有一个，容易使得新个体满足约束；此外，增加了记忆功能和双阈值的设定，使得在保持最优性的前提下尽快收敛，减少计算量，达到了较好的效果。

2 基于资源竞争链的浮动信息更新算法

使用本文模型求解过后，得到了工期内最优人员需求并在此人员需求下非关键活动的开始时间最早。此时，非关键活动的初始最晚开始时间及浮动时间等信息不再有效，因为各角色人员总数有了限制。Wiest[11]提出的方法可以确定活动的最晚开始时间：在不引起资源冲突（对资源的需求超过可提供的限制）的情况下，依次右移活动到可能的 LS。但由于移动稍早结束的活动时是以后面活动在 LS 处开始为前提，容易在某个活动的稍早开始处引起资源冲突，如图 4 中更新后 2 和 3 在第二个时间单位处可能发生冲突。为了更新活动的浮动信息并避免潜在的资源冲突，提出了基于资源竞争链的浮动信息更新算法。由于只考虑一个活动占用一种角色的资源，因此占用不同角色资源的活动之间并不存在竞争关系，算法只描述占用同种角色资源活动的最晚开始时间更新算法，图 3 为算法执行的流程。

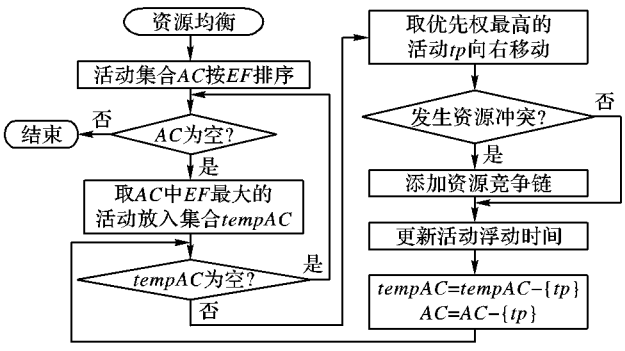


图 3 算法执行流程

2.1 算法描述

基于资源竞争链的浮动信息更新算法可以描述如下。

1) $AC = \{ac_1, ac_2, \cdots, ac_m\}$，$ac_i.EF$，$ac_i.LF$ 分别表示活动的最早和最晚结束时间。

2) 如果 $AC \neq \varnothing$，则 $tempAC = \{tp_1, tp_2, \cdots, tp_\alpha\}$，$tempAC \subseteq AC$，$\forall tp_i \in tempAC: tp_i.EF \geqslant ac_j.EF$，$1 \leqslant j \leqslant |AC|$；否则，算法结束。

3) 按以下优先权[11]依次对 $tempAC$ 中的元素执行右移操作：

① $tempAC$ 中，可能有最晚的 LS 的活动拥有最高优先权；

② 如果活动间 LS 相同，则优先移动所需资源最少的活动；

③ 如果除了可能的 *LS* 相同外,所需资源也相同,则首先移动标号最小的活动;

4) $AC = AC - tempAC$, 回到②。

活动右移操作 moveBackward(Activetp)描述如下。

①如果 $tp.ES \neq tp.LS$, 则进行下面操作;否则,此活动是关键活动,算法结束。

②右移*tp*直到某一时间点其结束端与其他活动发生资源冲突,移动过程中要考虑可能与 *tp* 并行执行的其他活动的所有可能出现的时间,包括正常执行出现的时间(*ES* 到 *EF*) 及因推迟开始时间或超时而可能出现的位置。

③ 如果 *tp* 的结束端可以右移到原始的 *LF*,则跳到 ⑤;否则,进行下面操作:$Conflic = \{cf_1, cf_2, \cdots, cf_\beta\}$,表示有可能与 *tp* 并行执行而发生资源冲突的非关键活动集合。$Conflic = C_1 \cup C_2$, $C_1 = \{cf_i \mid cf_i.ES + cf_i.D \leqslant Time\}$,表示可以在发生资源冲突前结束的活动,其中 *Time* 表示发生资源冲突的时间;$C_2 = \{cf_i \mid cf_i.ES + cf_i.D > Time\}$,表示无法在发生资源冲突前结束的活动。用 $cf_i.T$ 表示 cf_i 的资源可以被其他活动占有到的最晚时间,对于 C_1 中的活动,由于可以在 *Time* 之前结束,它们的资源可以被其他活动占用到其 *LF*,即 $cf_i.T = cf_i.LS + cf_i.D$, $cf_i \in C_1$; C_2 中的活动无法在 *Time* 之前结束,因此它们的资源可以被占用到其 *LS*,即 $cf_i.T = cf_i.LS$, $cf_i \in C_2$。

④如果 $\exists Satisfied = \{sf_1, \cdots, sf_\gamma\}$, $Satisfied \subseteq Conflic$, $\sum_{i=1}^{\gamma} sf_i.acc \geqslant tp.acc$, 且 $\forall sf_i \in Satisfied, \forall cf_j \in Conflic - Satisfied$, $sf_i.T \geqslant cf_j.T$($sf_i.acc$ 表示活动 sf_i 的资源占有量),则在 *tp* 到 *Satisfied* 的活动间建立资源竞争链(资源竞争链可以被表示为一个四元组 (*Cpt*,*Disp*,*Time*,*Num*),其中 *Cpt* 是资源竞争者,*Disp* 是资源被竞争者,*Time* 是发生冲突的时间,*Num* 是竞争的资源数量),表示如果 *tp* 由于延迟或推迟开始而在此时间段占用资源(*Satisfied* 中的活动要在此时间之前结束或在 *tp* 结束之后开始。从而 *tp* 的 *LS* 可以被更新为 $\min(\min\{sf_i.T \mid 1 \leqslant i \leqslant \gamma\} - tp.D, tp.LS)$。需要注意:首先,资源竞争链一旦建立,*Cmp* 被假设从 *Time* 到 *Cmp*. *LF* 的时间段内占有 *Disp* 的 *Num* 数量的资源,而 *Disp* 留给后面被移动的活动能够竞争的资源只有 $Disp.acc - Num$($Disp.acc$ 表示其占有的资源数);其次,关于最晚开始时间的确定,假设 A、B、C 三个活动中,A 和 B、B 和 C 间分别在时间 t_1, t_2 处建立资源竞争链,如果 B 可以在 t_1 前完成,A 就可以竞争到 B 的资源,进而可以继续竞争 C 的资源而获得浮动时间。因此,一些活动的浮动时间是通过传递竞争获得的。

⑤如果没有建立从 *tp* 到其他资源的竞争链,则将当前时间设置为 *tp* 的 *LF* 并更新其 *LS*, 算法结束。

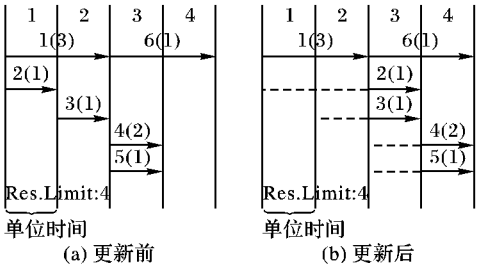
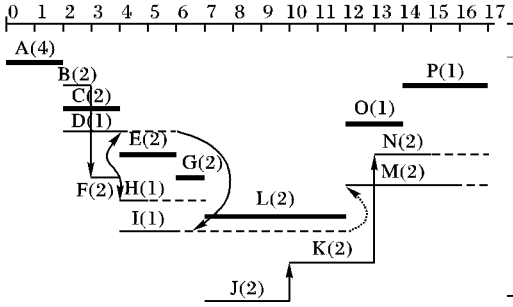


图 4　Wiest 最晚开始时间确定法实例

2.2　算法实例分析

如图 5,实线描述了活动的开始时间是 *ES* 的进行情况,虚线表示时间上可以向后移动的范围,粗线标记的是关键活动,字母及后面的数字分别表示活动标号和资源需求,资源限制是 5。图 5 的右侧列出了非关键活动的后继活动 (*Suc*)、最早开始时间(*ES*)、初始最晚开始时间(*LS*) 及更新后的最晚开始时间(*LS*(*Upt*)) 等信息。注意这里的 *ES* 是经本文模型求解后得到的 *ES*。



活动	*Suc*	*ES*	*LS*	*LS*(*Upt*)
B	E,F	2	3	3
D	O	2	10	10
F	I,J	3	9	5
H	L	4	6	6
I		4	15	11
J	M	7	10	9
K	N	10	12	12
M		12	13	13
N			15	15

图 5　基于资源竞争链的浮动信息更新算法实例图

图 5 中首先移动 M 和 N 且其结束端可以移到其初始 *LF*。依次右移 K,J,I,都会在其 *LF* 之前发生资源冲突,分别与 N,K,M 建立资源竞争链;如果这些活动推迟开始,那么被链接的活动就要推迟开始时间。N 可以移动到其初始 *LF*。D 在移动到 7 时与 I 发生资源竞争,I 刚好满足 D 的资源需求,D 到 I 建立资源竞争链;D 要延长到时间 7 或因推迟开始时间而在时间 7 需要占用资源时,I 或者在 7 前结束或者在 D 结束后开始才能得到所需资源。由于 I 的资源全部被 D 剥夺,从 4 到 13(D 的 *LF*) 不能被其他活动竞争,而 D 是可以被竞争的。F 右移到 4 可能发生冲突,D 的资源可以被竞争而 I 就不可以了。虽然 I 在 4 ~ 7 可能会出现,但考虑与 D 的资源链,在这段时间内 I 仍被视为已占有资源,但它不再被竞争。由于 I 可以在 7 前结束,D 的结束时间可以最晚到 I 的 *LF*; 但 I 在 13 与 M 竞争资源, D 的结束时间要到 13(I 的 *LF*), 需要竞争到 M 的资源,即 M 推迟到 13 开始。虽然这里 D 最晚只能在 12 结束,这里只是借此来说明竞争链与竞争链间的传递关系。

2.3　复杂度分析

从算法描述到实例分析可知,活动按 *EF* 排序、在可活动范围判断资源是否冲突、搜索与要移动的活动发生资源冲突的活动集、查找可竞争的活动集是实现调度控制的主要操作。如果有 m 项活动,工期是 t,则按 *EF* 排序的复杂度是$m \log m$; 对于每个要移动的活动,判断是否发生资源冲突的时间范围 $t' < t$, 搜索可能与之发生资源冲突的活动数 $m_{cf} < m$, 查找可竞争的活动数 $m_{cp} < m$。因此,最坏情况下算法花费的时间为 $m \log m + m(t + m + m)$, 所以算法的时间复杂度为 $O(m^2 + mt)$。

3　应用及分析

以一个基于 B/S 架构的加气站管理系统开发过程进行实验,活动相关属性是项目实际执行过程中统计而来的:该过程共有 23 项活动,关键活动 9 项,非关键活动 14 项,具体属性见表 1。笔者采用 Java 语言实现了一个控制系统,能够求解资源优化模型并实现浮动信息更新算法。系统以过程中各活动的属性值作为输入,得到最优资源需求及更新的浮动信息,并输出甘特图。经过资源优化模型的求解,开发过程最多

需要 4 名开发人员，并确定了基于最优资源需求限制下各活动的 *ES*，表 1 中的最优最早开始时间显示了此信息。工期和最优资源需求确定后，就可以得到更新的浮动信息，见表 1 中最优最晚开始时间及资源竞争链。

表 1 活动属性信息表

序号	后继活动	持续时间	初始最早开始时间	初始最晚开始时间	最优最早开始时间	最优最晚开始时间	所需资源	资源竞争链
1	2,3,4	4	0	0	0	0	4	—
2	5,6	1	4	5	4	4	2	(2,6,5,2)
3	5,8	2	4	4	4	4	2	—
4	15	2	4	25	7	11	1	(4,9,11,1)
5	7	3	6	6	6	6	2	—
6	9,10	2	5	11	5	5	2	—
7	10,11,12	2	9	9	9	9	2	—
8	12	1	6	10	7	10	1	(8,4,8,1)
9	—	2	7	29	8	11	1	(9,11,11,1)
10	13,17	3	11	13	11	13	1	(10,14,14,1)
11	14,17	3	11	13	11	13	1	(11,10,14,1)
12	15,17	5	11	11	11	11	2	—
13	—	4	14	27	25	27	2	—
14	—	2	14	29	14	16	2	(14,15,16,1)
15	16	2	16	27	16	20	1	(15,20,18,1)
16	—	2	18	29	18	22	1	(16,19,23,1)
17	18,19,20	2	16	16	16	16	2	—
18	21,22	5	18	18	18	18	2	—
19	—	2	18	29	20	22	1	(19,22,23,1)
20	—	2	18	29	18	22	1	(20,16,20,1)
21	23	3	23	23	23	23	2	—
22	23	2	23	24	23	24	2	(22,13,25,2)
23	—	5	26	26	26	26	2	—

利用 RCS[11]、RCPM[4]、本文所提模型分别求解表 1 中的实例，表 2 是结果对比。针对此实例，RCS、RCPM 都可以在资源约束为 4、工期为 31 下完成调度。

3.1 本文模型与 RCS 对比分析

RCS 是传统的资源约束调度，不同的优先权策略决定了最终结果。表 2 中的 RCS 调度结果中，各活动的 *ES* 可以确定。但由于没能提供活动间的资源竞争信息，除了关键活动，其他活动的 *LS* 和 *TF* 都没有被确定。本文模型在满足时间和资源的约束下安排了活动的 *ES*，通过资源竞争链约束了活动间的资源依赖，从而确定了活动的 *LS* 和 *TF*，为控制过程提供了有意义的信息。

3.2 本文模型与 RCPM 对比分析

RCPM 通过资源链约束活动间的资源流向，由于其静态性，使得并行执行的任务只能竞争固定的资源，即便有些活动并没有竞争后续开始活动的资源，该资源也不能供其他活动使用。而本文模型中，资源竞争链约束活动间的横向竞争的同时也考虑了并行活动可能形成竞争；实际执行过程中，可以根据动态执行情况，决定被竞争资源的使用者。表 1 中只有活动 2、6 的 RCPM 的 *TF* 优于本文模型的结果；而活动 4、8、9、15、20 的 *TF*，本文模型要优于 RCPM；其他活动的 *TF*，两模型结果一样。此外，RCPM 添加了 13 个链，而本文模型只有 12 个链，减小了网络拓扑的复杂度。

此外，将资源约束关系加入甘特图，可以对管理者提供更为有效的控制信息。图 6 是输出的甘特图。上部是以活动的最优最早开始时间确定的常规甘特图，而下面部分将资源竞争关系加入其中。每一个活动条代表此活动的范围，活动最早结束位置可以向后延伸，直到发生资源竞争的位置。对于被竞争的活动，在发生竞争的时间点为其加入一个子活动。但这个子活动是没有实际意义的，它只表示其父活动可能被其他活动竞争，或者在此时间前结束或者在竞争者结束后开始，方便建立竞争者和被竞争者间的竞争关系。将普通甘特图与包含资源链的甘特图对比使用，管理者就可以清楚地知道每一个活动是否可以推迟开始，在什么时候有可能发生资源冲突以及如何解决冲突。实验结果表明，此模型有效地解决了软件过程中的时间和资源间的平衡问题，保障工期的同时优化资源的利用，并为管理者提供了有效的控制信息，提高了软件过程的控制管理能力。

表 2 RCS、RCPM、本文模型结果对比表

序号	RCS				RCPM				本文模型			
	ES	*LS*	*TF*	*LK*	*ES*	*LS*	*TF*	*LK*	*ES*	*LS*	*TF*	*LK*
1	0	0	0	—	0	0	0		0	0	0	
2	4	—	—	—	4	5	1	(2,4)	4	4	0	(2,6,5,2)
3	4	4	0	—	4	4	0		4	4	0	
4	5	—	—	—	5	7	2	(4,6)	7	11	4	(4,9,11,1)
5	6	6	0	—	6	6	0		6	6	0	
6	7	—	—	—	7	9	2	(6,9)	5	5	0	
7	9	9	0	—	9	9	0		9	9	0	
8	6	—	—	—	6	8	2	(8,6)	7	10	3	(8,4,8,1)
9	9	—	—	—	9	11	2	(9,10)	8	11	3	(9,11,11,1)
10	11	—	—	—	11	13	2	(10,14)	11	13	2	(10,14,14,1)
11	11	—	—	—	11	13	2	(11,14)	11	13	2	(11,10,14,1)
12	11	11	0	—	11	11	0		11	11	0	
13	18	—	—	—	18	20	2	(13,22)	25	27	2	
14	14	—	—	—	14	16	2	(14,15)	14	16	2	(14,15,16,1)
15	16	—	—	—	16	18	2	(15,13)	16	20	4	(15,20,18,1)
16	25	—	—	—	25	29	4		18	22	4	(16,19,23,1)
17	16	16	0	—	16	16	0		16	16	0	
18	18	18	0	—	18	18	0		18	18	0	
19	25	—	—	—	25	27	2	(19,20)	20	22	2	(19,22,23,1)
20	27	—	—	—	27	29	2		18	22	4	(20,16,20,1)
21	23	23	0	—	23	23	0		23	23	0	
22	23	—	—	—	23	24	1	(22,16) (22,19)	23	24	1	(22,13,25,2)
23	26	26	0	—	26	26	0		26	26	0	

4 结语

基于关键路径法的软件过程控制模型，在平衡时间和资源的基础上，利用所提出资源优化模型确定非关键活动的开始时间，使得工期内资源耗费最优。在已优化耗费的约束下，通过添加资源竞争链标识活动间可能存在的资源竞争关系，使得过程活动能够有效地被控制。通过实例实验证明了本文所提模型的有效性。虽然本文模型有效地解决了时间和资源间的平衡，但在加强控制的稳定性以更小的代价应对环境变化方面，仍需更多深入的研究。

（下转第 2753 页）

行到第 45 次时,函数 41、42、43 出口和入口信息完整,函数 44 没有出口信息,运行记录如表 1 所示,因此可以将故障定位到函数 44。

Nodeid	Type	WinID	InID	PreID	Times
11	input	1	0	01	100
11	output	1	0	01	100
41	input	4	5	11	45
41	output	4	5	11	45
42	input	4	5	41	45
42	output	4	5	41	45
43	input	4	5	42	45
43	output	4	5	42	45
44	input	4	5	43	45
					0

图 7　部分运行记录

通过监测软件运行 100 次可以发现,窗口 s_4 中的函数 44 总共执行了 45 次,在执行第 45 次时发生故障,因此可以计算其可靠性为 44/45 = 97.8%,进而为计算窗口 s_4 的可靠性以及文件保存功能的可靠性和记事本模拟程序的可靠性计算提供依据。相对于其他 GUI 自动化测试模型来说,很难检测到软件多次运行之后出现的故障,并且这些故障难以再现,因此,无法对软件可靠性的评估提供数据支持,并且对出现的故障无法定位到代码中的具体函数。

4　结语

基于运行监测的 GUI 自动化测试模型将 GUI 分为窗口框架层、界面元素层、功能结构层和运行记录层。窗口框架层采用窗口导航有向图描述,反映为了实现 GUI 功能在用户输入下所有的控件的顺序组合,在功能结构层提出的覆盖准则的要求下,结合界面元素层中的用户输入,通过对窗口导航有向图中的环进行处理,采用 DFS 方法自动生成测试用例序列。测试开始时,通过测试用例序列驱动被测软件自动化运行。运行记录层通过插桩记录每一条路径的执行情况和实现窗口功能的所有函数的执行状态,进行动态监测和故障定位,为提高故障定位的速率和精度以及覆盖率提供依据。由于监测软件代码中每一个函数的执行情况,因此在多次执行后,可以检测出由于时空变化导致的空间错误引起的故障,进而得到每一个函数在多次执行后的失效次数,为计算可靠性提供数据支持。

参考文献:

[1] BROOKS P A, MEMOEN A M. Automated GUI testing guided by usage profiles [C]// Proceedings of 22th IEEE/ACM International Conference on Automated Software Engineering. Washington, DC: IEEE, 2007: 333 - 342.

[2] 李翔,高建华.一种基于事件关系的图形用户界面程序测试方法[J].小型微型计算机系统,2005,26 (4):671 - 675.

[3] 吴恒山,王金红.基于界面状态有效性的 GUI 自动测试模型[J].华中科技大学学报:自然科学版,2004,32 (12):34 - 37.

[4] CSAIL A C. Performance profiling with endoscope, an acquisitional software monitoring framework [J]. Proceeding of the VLDB Endowment, 2008, 1(1): 42 - 53.

[5] ALLAN C, AVGUSTINOV P, CHRISTENSEN A S, *et al*. Adding trace matching with free variables to AspectJ [C]// Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications. New York: ACM, 2005: 345 - 364.

[6] 叶茂,高海昌,冯博琴,等.基于窗口导航有向图的 GUI 测试覆盖准则[J].西南交通大学学报,2006,41(4):476 - 480.

[7] OSTRAND T, ANODIDE A, FOSTER H, *et al*. A visual test development environment for GUI systems [C]// Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: ACM, 1998: 82 - 92.

[8] MEMON A M, SOFFA M L, POLLACK M E. Coverage criteria for GUI testing [C]// Proceedings of the 8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York: ACM, 2001: 256 - 267.

[9] 阮辉,严俊,张健.基于路径分析的死循环检测[J].计算机学报,2009,32(9):1750 - 1758.

[10] 张广梅,李晓维,韩丛英.路径测试中基本路径集的自动生成[J].计算机工程,2007,33(22):195 - 197.

[11] 晏华,袁海东,尹立孟.代码自动插装技术的研究与实现[J].电子科技大学学报,2002,31(1):62 - 66.

[12] 王斌,张伟,段见飞,等.基于软件运行记录的测试方法[C]//第三届全国软件测试会议与移动计算、栅格智能化高级论坛论文.山西:电脑开发与应用,2009:79 - 82.

(上接第 2748 页)

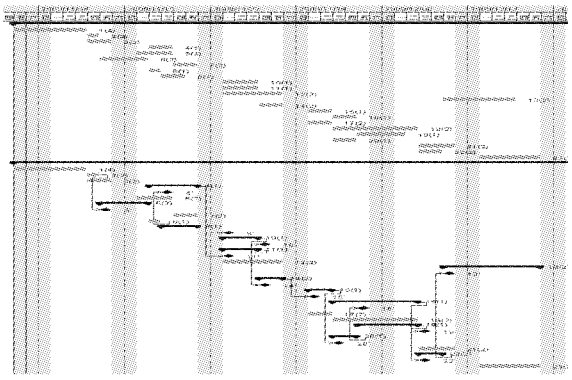


图 6　甘特图

参考文献:

[1] Software Engineering Institute (SEI). CMMI for Development, Version1.2 [S/OL]. Carnegie Mellon: Software Engineering Institute, 2006 [2009 - 12 - 15]. http://www.sei.cmu.edu/reports/06tr008.pdf.

[2] Project Management Institute (PMI). A guide to the project management body of knowledge [S]. 3rd ed. Pennsylvania: Project Management Institute, 2004.

[3] 武占春,王青,李明树.一种基于 PDCA 的软件过程控制与改进模型[J].软件学报,2006,17(8):1669 - 1680.

[4] KIM K, de la GARZA J M. Evaluation of the resource-constrained critical path method algorithms [J]. Journal of Construction Engineering and Management, 2005, 131(5): 522 - 532.

[5] LU MING, LI HENG. Resource-activity critical-path method for construction planning [J]. Journal of Construction Engineering and Management, 2003, 129(4): 412 - 420.

[6] 万静,何月娇,易军凯.面向任务的软件过程控制模型[J].计算机过程与应用,2009,45(22):56 - 58.

[7] EASA S M. Resource leveling in construction by optimization [J]. Journal of Construction Engineering and Management, 1989, 115 (2): 302 - 316.

[8] CHAN W-T, CHUA D K H, KANNAN G. Construction resource scheduling with genetic algorithms [J]. Journal of Construction Engineering and Management, 1996, 122(2): 125 - 132.

[9] 朱颢东,钟勇.一种改进的模拟退火算法[J].计算机技术与发展,2009,19(6):32 - 35.

[10] RUSSELL S, NORVING P. 人工智能——一种现代方法[M].2 版.北京:人民邮电出版社,2004.

[11] WIEST J D. Some properties of schedules for large projects with limited resources [J]. Operations Research, 1964, 12(3): 395 - 418.