过程挖掘中增强活动依赖的最优对齐算法

邵叱风 方贤文 杨慧慧
(安徽理工大学数学与大数据学院 安徽 淮南 232001)

摘 要 一致性检验是过程挖掘领域中检验日志与模型之间偏差的有效方法,对齐是众多先进方法之一。现阶段最优对齐的成本计算大多只与对齐中移动个数保持相关,缺乏对单一活动成本的考虑。因此,提出一种增强活动依赖的最优对齐加权计算方法。基于动态规划求解日志与模型之间的序列对齐;统计合法移动信息,差异化活动及移动类别的成本进行依赖增强;依据对齐成本加权计算方法计算最优对齐并以 Java 应用的形式对以上方法进行实现。实验利用一个常规化模型及其部分日志对方法的可行性及有效性进行了验证。

关键词 一致性检验 最优对齐 序列对齐 动态规划 依赖增强

中图分类号 TP391.9 **文献标志码** A DOI: 10.3969/j.issn.1000-386x.2023.08.042

OPTIMAL ALIGNMENT CALCULATION METHOD TO ENHANCE ACTIVITY DEPENDENCY IN PROCESS MINING

Shao Chifeng Fang Xianwen Yang Huihui
(*School of Mathematics & Big Data, Anhui University of Science and Technology, Huainan 232001, Anhui, China*)

Abstract Conformance checking is an effective method for examining deviations between logs and models in the field of process mining, and alignment is one of many advanced methods. Most of the cost calculations of optimal alignment at this stage only keep correlation with the number of moves in the alignment and lack consideration of the cost of a single activity. We proposed an optimal alignment weighted calculation method to enhance the activity dependence. We solved the sequence alignment between the log and the model based on dynamic planning. We statistically calculated the cost of legal moves, differentiated activities and moved categories to enhance the dependency. We calculated the optimal alignment based on the alignment cost weighted method, and implemented the above method as a Java application. The experiments used a generalized model and some of its logs to prove the feasibility and validity of the method.

Keywords Conformance checking Optimal alignment Sequence alignment Dynamic planning Dependency enhancement

0 引 言

BPM(Business Process Management)业务流程管理[1-2]伴随着大数据时代的到来得到了进一步发展。从业务流程出发,基于信息及管理技术提供统一的建模、运行和监控环境。经过业务流程的不断执行,信息系统将生成大量的事件日志文件[3],其记录了与流程密切相关可用于进一步业务流程分析的重要数据,因此,流程挖掘对提高业务流程管理水平而言受到人们越来越多的重视[4-7]。在企业管理中,完整的信息管理系统要求流程模型和事件日志之间具有很高的重构性,即希望模型的过程行为和性质可以从完整描述的日志中表现出来[8]。然而,通过比较模型和事件日志可知,信息系统中记录的事件日志与基于业务流程构建的模型之间总是存在一些偏差,因而使得日志中的部分行为无法在模型上重放[9-10]。针对上述问题,对过程模型与记录系统实际运作的事件日志之间进行一

收稿日期:2020-09-22。国家自然科学基金项目(61402011,61572035);安徽省自然科学基金项目(1508085MF111,1608085QF149);安徽理工大学研究生创新基金项目(2019CX2068)。邵叱风,硕士生,主研领域:Petri 网,过程挖掘。方贤文,教授。杨慧慧,硕士生。

致性校验变得至关重要[11]。

近年来,校验给定模型和事件日志之间一致性的多种方法已被提出[4-5,12-19],其中文献[16]所做的对齐方法是最先进的方法之一。文献[17]将日志与模型进行对齐,提出一个复杂的方法用以指出偏差发生的位置及偏差程度;文献[18]分析那些最优对齐包含的移动集合完全相同,只是移动出现顺序不同,继而提出相似最优对齐的概念;一般而言能够检测到最小偏差成本的一组对齐被认为是事件日志与业务流程之间的最优对齐[19]。使用现有方法可以得到事件日志中的迹与 Petri 网的发生序列之间的所有最优对齐。对齐处理与偏差检测作为很多研究工作的重要组成部分,因而在过程挖掘领域中越来越重要,例如,模型修复[20-22]和遗传优化过程挖掘[23]等。

现有的最优对齐方法在计算对齐成本时大多仅考虑对齐中的移动个数(包括相似性最优对齐),缺乏对单一活动成本即活动依赖的考虑。在此提出一种基于依赖增强的最优对齐计算方法,目的提高计算序列对齐的效率、最优对齐对活动的依赖性。基于自定义得分矩阵的配置,使用动态规划的方法计算日志中所有序列与模型可观测迹的对齐;分析对齐集合中移动的信息,通过加权的方式计算模型与日志上合法移动的总成本;最后利用分组排序得到日志与模型的最优对齐序列及对齐成本。

1 基本概念

本节内容列出了文章所需的部分基本概念,用以辅助文章的阅读。Petri 网相关定义参考文献[24]。

定义 1(移动,合法移动) 网系统 $S:=(N,M_{\text{ini}},M_{\text{fin}})$,其中 $N:=(P,T,F,\lambda)$ 上的移动是一个二元组 $(x,y)\in(A\cup\{\gg\})\times(T\cup\{\gg\})\setminus\{(\gg,\gg)\}$。除以下三种合法移动,其余均为非法移动。

1) $Move(x,y)$ 是一个日志迹上的移动,当且仅当 $x\in\mathcal{A}$ 且 $y=\gg$。

2) $Move(x,y)$ 是一个系统上的移动,当且仅当 $x=\gg$ 且 $y\in T$。

3) $Move(x,y)$ 是一个同步移动,当且仅当 $x\in\mathcal{A}$,$y\in T$ 且 $\lambda(y)=x$。

定义 2(对齐) 迹 $\upsilon\in\mathcal{A}^*$ 和网系统 $S:=(N,M_{\text{ini}},M_{\text{fin}})$,其中 $N:=(P,T,F,\lambda)$ 上的执行 σ 之间的对齐是拥有 S 上合法移动的有限序列 $\gamma\in M_s^*$,其中 $\pi_1(\gamma)\mid\mathcal{A}=\upsilon$ 且 $\pi_2(\gamma)\mid T=\sigma$。

$$\gamma_1=\begin{array}{|c|c|c|c|c|c|c|c|}\hline a&c&b&d&\gg&f&g&\gg\\\hline a&c&b&d&\tau&f&g&\tau\\\hline t_1&t_4&t_2&t_5&t_7&t_8&t_9&t_{11}\\\hline\end{array}$$

$$\gamma_2=\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|}\hline a&c&b&d&\gg&\gg&\gg&\gg&\gg&f&g&\gg\\\hline a&c&b&d&e&c&\tau&d&\tau&f&g&\tau\\\hline t_1&t_4&t_2&t_5&t_6&t_4&t_3&t_5&t_7&t_8&t_9&t_{11}\\\hline\end{array}$$

图 1 同一序列的两个不同对齐

例如,γ_1 和 γ_2 展示了指定两个域的两个序列的移动分别为 $<(a,t_1),(c,t_4),(b,t_2),(d,t_5),(\gg,t_7),(f,t_8),(g,t_9),(\gg,t_{11})>$ 和 $<(a,t_1),(c,t_4),(b,t_2),(d,t_5),(\gg,t_5),(\gg,t_4),(\gg,t_3),(\gg,t_5),(\gg,t_7),(f,t_8),(g,t_9),(\gg,t_{11})>$,并且 $\pi_1(\gamma_1)\mid\mathcal{A}=<a,c,b,d,f,g>$,$\pi_2(\gamma_1)\mid T=<t_1,t_4,t_2,t_5,t_7,t_8,t_9,t_{11}>$,$\pi_1(\gamma_2)\mid\mathcal{A}=<a,c,b,d,f,g>$,$\pi_2(\gamma_2)\mid T=<t_1,t_4,t_2,t_5,t_6,t_4,t_3,t_5,t_7,t_8,t_9,t_{11}>$,同一个迹之间的两个对齐显示了观察到的迹和建模的流程实例之间的不同偏差,比较对齐的常用方法是根据对齐包含的移动对其进行量化。

定义 3(对齐成本) 网系统 S 上的合法移动成本可表示为一个函数 $c:M_S\to N_0$,将成本分配到 S 上的合法移动。故迹 $\upsilon\in\mathcal{A}^*$ 和网系统 $S:=(N,M_{\text{ini}},M_{\text{fin}})$,其中 $N:=(P,T,F,\lambda)$ 上的执行 σ 之间的对齐 γ 对应成本依据合法移动成本函数 c,以及 γ 上所有的移动成本总和定义为:

$$c(\gamma)=\sum_{i=1}^{|\gamma|}c(\gamma_i)\tag{1}$$

定义 4(最优对齐) 根据在 S 上的合法移动的成本函数 c 及加权函数 w,一个日志迹 $\upsilon\in A^*$ 和系统 $S\in$ 之间的最优对齐 $\gamma\in{}^{\upsilon}_{S}$,$\forall\gamma'\in{}^{\upsilon}_{S}$:$c(\gamma)\leqslant c(\gamma')$,其中: 为 S 的多重集; ${}^{\upsilon}_{S}$ 表示日志迹 υ 与系统 S 的对齐集合。

即日志迹 υ 与模型中可观测迹的对齐成本最小的对齐为最优对齐,可能有多个最优对齐。比如两个对齐均插入或跳过一个活动,为体现日志与网系统中变迁的重要程度,降低最优对齐个数。下面给出定义 5 和定义 6 如下:

定义 5(标签加权) 依据活动不同重要性,考虑单一活动成本,对对齐序列中的标签 γ_i 进行加权处理,对于权函数 w:

$$w(\pi_1(\gamma_i)\mid\mathcal{A})=w(\pi_2(\gamma_i)\mid T)=w(\gamma_i),w(\gamma_i)\in\mathbf{N}$$

在输入过程中,输入形式如下:

$$Weight_{\text{activity}}:(A_1,w(A_1),A_2,w(A_2),\cdots,A_i,w(A_i)),A=(\mathcal{A}\cup\{\tau\})^*$$

定义 6(移动加权) 依据偏差出现位置,考虑其

对对齐成本的影响,在计算总的对齐成本时对于系统上的合法移动,迹上的合法移动按类别 $Weight_{\text{skip}}$、$Weight_{\text{insert}}$进行加权处理。输入形式如下:

$$\begin{aligned}&Weight_{\text{skip}}/Weight_{\text{insert}}:\\&(Cost_{\text{skip}},W(Cost_{\text{skip}}),Cost_{\text{insert}},W(Cost_{\text{insert}}))\\&Cost_{\text{total}}=Cost_{\text{skip}}\times W(Cost_{\text{skip}})+Cost_{\text{insert}}\times W(Cost_{\text{insert}})\end{aligned}\tag{2}$$

则:

$$c(\gamma_i)=\begin{cases}w(\gamma_i)\times w(Cost_{\text{skip}})\\ \gamma_i=Move(x,y),x=\gg \text{ and } y\in T\\ w(\gamma_i)\times w(Cost_{\text{insert}})\\ \gamma_i=Move(x,y),x\in\mathcal{A} \text{ and } y=\gg\\ w(\gamma_i)\times 0\quad \gamma_i=Move(x,y)\\ x\in\mathcal{A},y\in T \text{ 和 } \lambda(y)=x\end{cases}\tag{3}$$

定义 7(最优对齐成本)　事件日志 $L\in\mathcal{B}(\mathcal{A}^*)$($\mathcal{B}(\mathcal{A}^*)$表示当前系统完备日志)与网系统 $S\in$　的最优对齐成本根据在 S 上的合法移动的成本函数 c 及加权函数 w,定义为:

$$Cost[L,S,c,w]=\sum_{v\in L}(L(v)\times Cost[v,s,c,w])\tag{4}$$

通过对活动的加权区分了对其中不同活动的成本; 对偏差类别的加权区分了偏差发生位置的重要程度; 相较于单一量化偏差个数的对齐成本求解进行了改进。

2　算法及其实现

此部分主要介绍基于动态规划求解两个序列的对齐,通过循环调用求得 $v\in\mathcal{A}^*$ 和网系统 $S:=(N,M_{\text{ini}},M_{\text{fin}})$,其中 $N:=(P,T,F,\lambda)$ 上的执行 σ 的对齐; 然后利用加权函数 w 对每个对齐的成本 $c(\gamma)$ 进行计算,并按照日志的大小分成 $L.Size()$ 组进行排序,求得日志迹与模型的最优对齐 $\gamma\in$　$_S^v$。

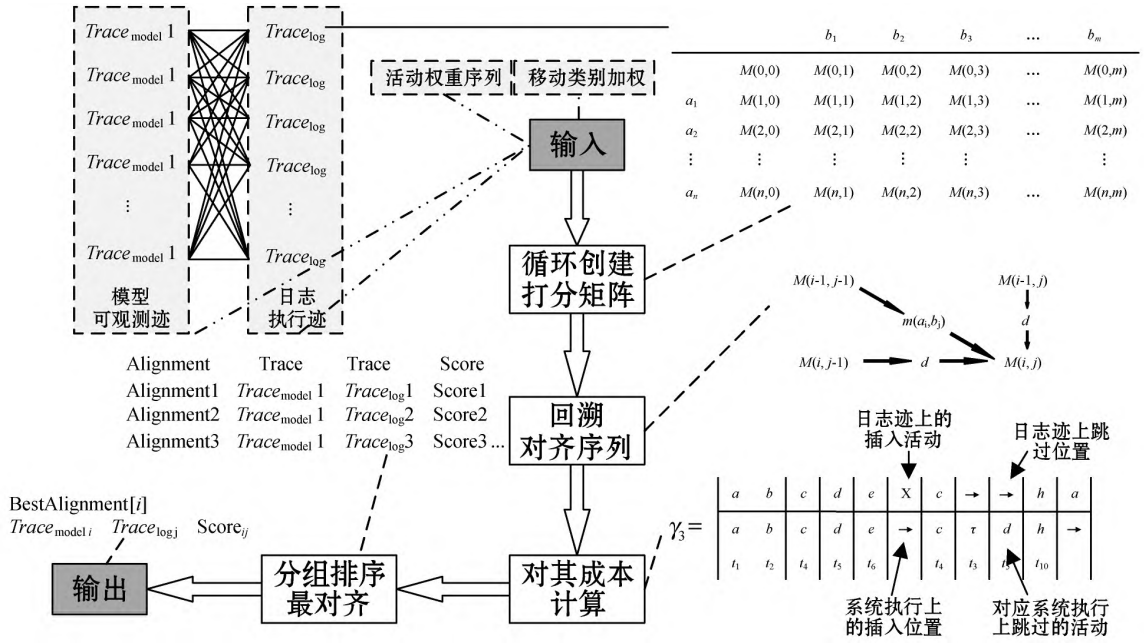


图 2　增强活动依赖的最优对齐算法实现流程

2.1　序列对齐

基于动态规划计算日志迹与模型可观测序列的对齐,主要是构建日志迹与模型可观测序列的打分矩阵,并回溯出得分最高的对齐结果。其中参数包括打分矩阵中的匹配得分、不匹配得分及间隙罚值,即同步移动、错误对齐及合法移动的分值。方法有以下几个主要步骤:

1) 矩阵初始化。对于日志迹 $v=a_1,a_2,\cdots,a_n$ 和模型可观测序列 $\sigma=b_1,b_2,\cdots,b_m$,然后创建大小为 $(n+1)\times(m+1)$ 的打分矩阵,其中: n 为行数且是当前日志迹 v 的长度; m 是列数且是当前模型可观测序列 σ 的长度。然后用间隙罚值填充值矩阵的第一行和第一列。间隙惩罚是将对齐中未匹配字符与空白字符(间隙)进行比较时获得的值。

序列 A 和序列 B 的打分矩阵如图 3 所示。

		b_1	b_2	b_3	…	b_m
	$M(0,0)$	$M(0,1)$	$M(0,2)$	$M(0,3)$	…	$M(0,m)$
a_1	$M(1,0)$	$M(1,1)$	$M(1,2)$	$M(1,3)$	…	$M(1,m)$
a_2	$M(2,0)$	$M(2,1)$	$M(2,2)$	$M(2,3)$	…	$M(2,m)$
⋮	⋮	⋮	⋮	⋮		⋮
a_n	$M(n,0)$	$M(n,1)$	$M(n,2)$	$M(n,3)$	…	$M(n,m)$

图 3　序列 A 和序列 B 的打分矩阵

2) 矩阵填充。假设打分矩阵称为矩阵 M,则矩阵

M 的元素的公式为:

$$M(i,j)=\max\begin{cases}M(i-1,j-1)+m(a_i,b_j)\\M(i-1,j)-d\\M(i,j-1)-d\end{cases}\tag{5}$$

式中:$M(i-1,j-1)$ 为矩阵元素 $M(i,j)$ 对角线左上角;$M(i,j-1)$ 为 $M(i,j)$ 的左侧;$M(i-1,j)$ 为 $M(i,j)$ 的上方;$m(a_i,b_j)$ 为序列 a 中的残基 i 和序列 b 中的残基 j 的残差矩阵;d 为间隙惩罚的符号。假设间合法移动模型 $s(\rightarrow,a)=s(a,\rightarrow)=-d$ 对于 $a\in Q,d>0$,那么长度为 L 的间隙区域的值等于 $-dL$。

假设合法移动分数是 d,那么 $s(0,j)=-j\times d$,$s(i,0)=-i\times d$,$s(0,0)=0$。

$$s(a_i,b_j)=\begin{cases}matchScore & a_i=b_j\\mismatchScore & a_i\neq b_j\end{cases}\tag{6}$$

部分打分矩阵代码如 Code_1 和 Code_2 所示。

Code_1: Calculate the first row and first column of matrix according to gapvalue

```
1  private void InitScore(int Gap,int X,int Y) {
2      SCORE[0][0]=0;
3      for(int i=1;i<=X;i++) {
4          SCORE[i][0]=SCORE[i-1][0]+Gap;
5      }
6      for(int j=1;j<=Y;j++) {
7          SCORE[0][j]=SCORE[0][j-1]+Gap;
8      }
9  }
```

Code_2: Calculate the score matrix according to the interface input

```
1  private void CalcScore(int Match,int Miss,int Gap,int X,
int Y,
2      String stringX,String stringY) {
3      int diag,up,left;
4      for(int j=1;j<=Y;j++) {
5          for(int i=1;i<=X;i++) {
6              diag=Diag(Match,Miss,i,j,stringX,stringY);
7              up=Up(Gap,i,j);
8              left=Left(Gap,i,j);
9              if(diag>=up && diag>=left) {
10                 SCORE[i][j]=diag;
11                 DIRECTION[i][j]='D';
12             } else if(up>=left) {
13                 SCORE[i][j]=up;
14                 DIRECTION[i][j]='U';
15             } else {
16                 SCORE[i][j]=left;
17                 DIRECTION[i][j]='L';
18             }
19         }
20     }
21 }
```

Code_1 依据合法移动分值计算打分矩阵首行首列,Gap、X、Y 分别为间隙罚值、序列 X 的长度、序列 Y 的长度。Code_1(2) 对打分矩阵 [0,0] 位置元素赋值为 0,Code_1(3-5) 对首列除 [0,0] 位置外元素依据间隙罚值进行赋值处理,Code_1(6-8) 对首行除 [0,0] 位置外元素依据间隙罚值进行赋值处理。

Code_2 依据错误对齐、同步移动、合法移动计算首行首列外的打分矩阵及方向矩阵,$Match$(合法移动得分)、$Miss$(错误对齐得分)、Gap(间隙罚值得分)、X(序列 X 的长度)、Y(序列 Y 的长度)、$StringX$(序列 X)、$StringY$(序列 Y),$Diag()$、$Up()$、$Left()$ 为分值计算函数。Code_2(3-19) 使用两层嵌套对打分矩阵首行首列外的位置元素进行赋值。

3) 回溯步骤。在完全填充大小为 $(n+1)\times(m+1)$ 的打分矩阵之后,对齐得分(所有替换值之和加上所有间隔惩罚之和)是两个序列的打分矩阵最右下角元素的值,$M(n+1,m+1)=m(n,m)$。回溯步骤如图 4 所示。

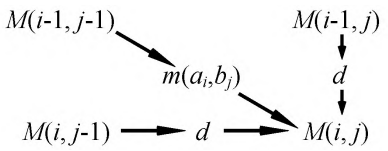


图 4 回溯步骤

从起点 $M(n,m)$ 开始回溯到终点 $m(0,0)$。如果 $M(i,j)=M(i-1,j-1)+m(a_i,b_j)$,那么回溯轨迹为 $(i,j)\rightarrow(i-1,j-1)$。

4) 确定对齐结果。若回溯到左上角单元格,将 a_i 添加到匹配字串 A(即日志迹),将 b_j 添加到匹配字串 B(即模型可观测序列)。

若回溯到上边单元格,将 a_i 添加到匹配字串 A,将 $\rightarrow$ 添加到匹配字串 B。

若回溯到左边单元格,将 $\rightarrow$ 添加到匹配字串 A,将 b_j 添加到匹配字串 B。

部分回溯代码及界面右侧矩阵输出代码如 Code_3 和 Code_4 所示。

Code_3: Backtracking matrix output stringX

```
1  private String PrintOptimalX(String StringX,String Opti-
malX,int i,int j) {
2      if(i==0 && j==0) {
3          return OptimalX;
4      }
5      if(DIRECTION[i][j]=='D') {
6          OptimalX=PrintOptimalX(StringX,OptimalX,
i-1,j-1);
```

```
7            OptimalX = OptimalX + StringX.charAt(i-1);
8        } else if (DIRECTION[i][j] == 'L' || j==0) {
9            OptimalX = PrintOptimalX(StringX, OptimalX, i-1,j);
10           OptimalX = OptimalX + StringX.charAt(i-1);
11       } else {
12           OptimalX = PrintOptimalX(StringX, OptimalX, i, j-1);
13           OptimalX = OptimalX + '→';
14       }
15       return OptimalX;
16   }
```

Code_4: Backtracking matrix output stringY

```
1  private String PrintOptimalY(String StringY,String OptimalY,int i,int j) {
2        if (i==0 && j==0) {
3            return OptimalY;
4        }
5        if (DIRECTION[i][j] == 'D') {
6            OptimalY = PrintOptimalY(StringY,OptimalY,i-1,j-1);
7            OptimalY = OptimalY + StringY.charAt(j-1);
8        } else if (DIRECTION[i][j] == 'U' || i==0) {
9            OptimalY = PrintOptimalY(StringY, OptimalY, i, j-1);
10           OptimalY = OptimalY + StringY.charAt(j-1);
11       } else {
12           OptimalY = PrintOptimalY(StringY,OptimalY,i-1,j);
13           OptimalY = OptimalY + '→';
14       }
15       return OptimalY;
16   }
```

Code_3 用于回溯出对齐序列 X,$StringX$(对齐序列 X)、$OptimalX$(对齐后序列 X),i、j 为元素在矩阵中的位置,依据确认对齐结果中的描述进行 $OptimalX$ 的拼接,Code_4 同理回溯对齐后序列 Y。

计算单条日志迹与单条模型可观测序列对齐的方法通过 Java 编程实现,通过设定 MatchScore(同步移动)、MismatchScore(错误对齐)、GapScore(合法移动)三个预值,并输入需要对齐的两个序列,点击 ALIGNMENT 按钮即可计算得出两对序列之间的最优对齐,并输出得分、运行耗时,且在界面右侧打印序对齐计算时生成的打分矩阵。SAVE 按钮功能可将计算结果进行. txt 或. rtf 格式的保存。

插件进行日志迹 $<a,c,b,d,f,g>$ 与系统可观测行为 $<a,c,b,d,\tau,f,g,\tau>$ 的对齐处理,结果如图 5 所示,打分矩阵与回溯过程如图 6 所示。

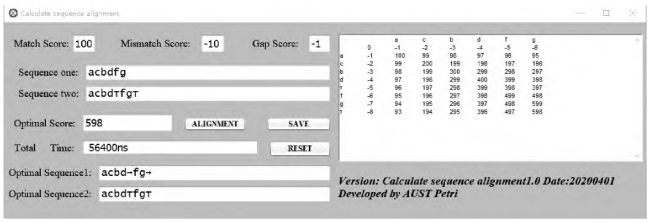


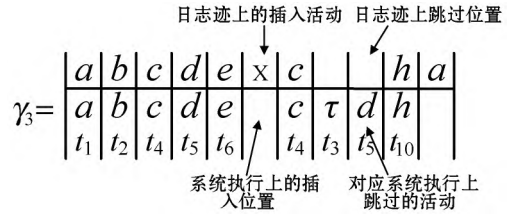
图 5　对齐序列计算界面

		a	c	b	d	f	g
	0	-1	-2	-3	-4	-5	-6
a	-1	100	99	98	97	96	95
c	-2	99	200	199	198	197	196
b	-3	98	199	300	299	298	297
d	-4	97	198	299	400	399	398
τ	-5	96	197	298	399	398	397
f	-6	95	196	297	398	499	498
g	-7	94	195	296	397	498	599
τ	-8	93	194	295	396	497	598

图 6　打分矩阵及回溯过程

2.2　对齐成本

使用常规成本计算方法在计算日志与模型的最优对齐时,大多仅考虑对齐中的移动个数,一条日志迹可能与多个模型可观测迹形成最优对齐。在实际系统中,每个活动的重要程度必然不是完全相等的,且日志上移动与模型上移动的成本在模型修复时显然也是不等的。在此使用加权函数 w 加权计算对齐成本 $c(\gamma)$,考虑对齐计算时不同活动的权重、插入及跳过的权重,增加对齐计算时需考虑的因素,增加对齐成本计算结果对活动的依赖性,差异化具有相同移动个数且包含不同活动的对齐序列的成本。加权计算如图 7 所示。



$$c(\gamma) = \sum_{i=1}^{|\gamma|} c(\gamma_i)$$

$$c(\gamma_i) = \begin{cases} w(\gamma_i) * w(Cost_{\text{skip}}) & \gamma_i = Move(x,y),\ x = \rightarrow \text{和 } y \in T \\ w(\gamma_i) * w(Cost_{\text{Insert}}) & \gamma_i = Move(x,y),\ x \in \mathcal{A} \text{和 } y = \rightarrow \\ w(\gamma_i) * 0 & \gamma_i = Move(x,y),\ x \in \mathcal{A}, y \in T \text{ 和 } \lambda(y) = x \end{cases}$$

图 7　加权计算示意图

加权计算对齐成本具体方法如 Code_5 所示,展示 calculateCost 函数进行对齐序列计算及其加权成本计算,tWeightstr(标签加权字符串)、logSEQstr(对齐后的日志序列)、modelSEQstr(对齐后的模型可观测迹)、MatchValue(同步移动分值)、MisMatchValue(错误对齐分值)、GapValue(合法移动分值)、weightofskip(跳过操作的权值)、weightofinsert(插入操作的权值),getAlignment() 是对序列匹配的重复调用求解日志与模型

的对齐序列，getCharacterPosition( ) 获取移动的位置，getPositionCharacter( ) 获取移动中包含的活动。

Code_5: Weighted calculation of alignment costs

```
1  public List<ModulePairSeq> calculateCost(String tWeightstr,
String logSEQStr,String modelSEQStr,String MatchValue,
2                String MismatchValue,String GapValue,String
Weightofskip,String Weightofinsert) {
3      HashMap<String,String> tWeight=operationOfWeight
(tWeightstr);
4      String[] logseq=logSEQStr.split("\r\n");
5      String[] modelseq=modelSEQStr.split("\r\n");
6      List<ModulePairSeq> result=getAlignment(MatchVal-
ue,MismatchValue,GapValue,logseq,modelseq);
7      for(int j=0;j<result.size();j++) {
8          Double skipscore=0.00,insertscore=0.00;
9          if(result.get(j).getLogResult().contains("→")) {
10             List<Integer> Logindex=getCharacterPo-
sition(result.get(j).getLogResult(),"→");
11             List<String> Modelstr=getPositionChar-
acter(Logindex,result.get(j).getModelResult());
12             skipscore=addWeightScore(tWeight,Modelstr);
13             result.get(j).setSkipCost(skipscore);
14             result.get(j).setLogIndex(Logindex);
15             result.get(j).setModelStr(Modelstr);
16         }
17         if(result.get(j).getModelResult().contains("
→")) {
18             List<Integer> Modelindex=getCharacter-
Position(result.get(j).getModelResult(),"→");
19             List<String> Logstr=getPositionCharac-
ter(Modelindex,result.get(j).getLogResult());
20             insertscore=addWeightScore(tWeight,Logstr);
21             result.get(j).setInsertCost(insertscore);
22             result.get(j).setModelIndex(Modelindex);
23             result.get(j).setLogStr(Logstr);
24         }
25         BigDecimal b1=new BigDecimal(Double.toString
(skipscore)),b2=new BigDecimal(Double.toString(insertscore));
26         BigDecimal b3=getWeightOfKind(Weightofskip),
b4=getWeightOfKind(Weightofinsert);
27         result.get(j).setTotalCost(Double.parseDouble
(b1.multiply(b3).add(b2.multiply(b4)).toString()));
28     }
29     return result;
30 }
```

1) 在 2.1 节的基础上实现序列对齐方法的循环调用，Code_5(4 -6) 计算日志 L 中每条迹与模型 S 的所有可观测序列的对齐。

2) 依据定义 5 的输入形式，Code_5(3) 初始化活动权值 Map。

3) Code_5(7 -28) 为加权计算对齐成本部分，其中 Code_5(9 -16) 统计日志迹上的跳过位置以及对应系统执行上跳过的活动并赋值给对象 result; Code_5(17 -24) 统计系统执行上的插入位置以及日志迹上的插入活动并赋值给对象 result。

4) 依据定义 5 的标签权值，定义 6 的移动类别权值，Code_5(25 -27) 计算对齐成本 $c(\gamma)$。

5) 将对齐成本计算结果按模型可观测序列的条数等分为多组(Code_6)，并排序得到每条日志迹的最优对齐。

Code_6 用于对齐结果的分组，list 为日志与模型的对齐结果，groupSize 为分组大小，Code_6(2 -4) 用于计算分组数，Code_6(5 -9) 用于分组结果 newList 的赋值。分类加权计算对齐成本的方法由 Java 编程实现如图 8 所示。

Code_6: splite result

```
1  private static List<List<ModulePairSeq>> splitList(List
<ModulePairSeq> list,int
2  groupSize) {
3      int length=list.size();
4      int num=(length+groupSize-1)/groupSize;
5      List<List<ModulePairSeq>> newList=new Array-
List<>(num);
6      for(int i=0;i<num;i++) {
7          int fromIndex=i* groupSize;
8          int toIndex=(i+1)* groupSize<length?(i+
1)* groupSize:length;
9          newList.add(list.subList(fromIndex,toIndex));
10     }
11     return newList;
12 }
```

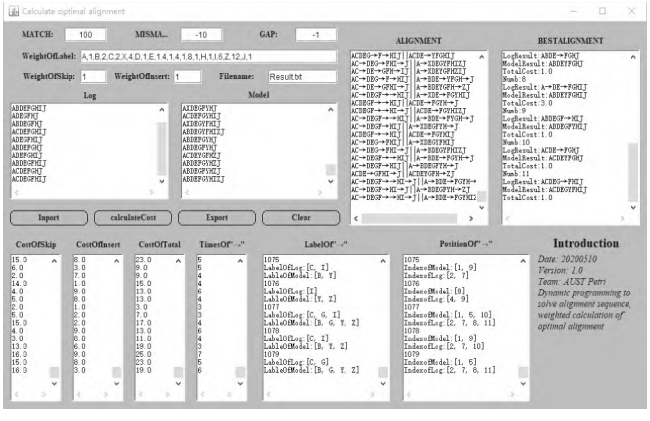


图 8 加权计算对齐成本求解最优对齐

3　实验分析

本节主要对动态规划序列对齐方法、对齐成本加权计算方法进行验证，分析序列对齐方法的耗时，不同权重对最优对齐结果的影响。实验选取如图 9 所示模型，该模型包含选择、并发和循环结构，以及静默变迁 X、Y、Z。选取日志序列如表 1 所示。所有的实验均在配有 I5-7300HQ 2.5 GHz 四核处理器和 16 GB 运存的机器上进行的，使用 Java SE 1.7 开发环境。

表 1　部分运行日志

序列号	序列	执行次数
1	ADEFGHJ	155
2	ACDEGFHJ	175
3	ABDEFGHIJ	188
4	ADEGFHJ	176
5	ABDEGFHJ	174
6	ACDEFGHIJ	162
7	ADEGFHIJ	172
8	ABDEFGHJ	150
9	ADEFGHIJ	162
10	ABDEGFHIJ	156
11	ACDEFGHJ	163
12	ACDEGFHIJ	167

Petri 网模型常见的结构关系包括并发、选择和循环等[24]，为增加执行序列的多样性及产生带有偏差的日志，选取如图 9 所示模型，包含静默变迁 X、Y 和 Z，分布在选择、并发和循环结构中。对文献[25]中增广 Petri 网模型模拟运行插件进行改进，并对图 9 模型进行日志生成，为防止状态爆炸，模型中循环执行的次数限定为 1，部分生成日志如表 1 所示。

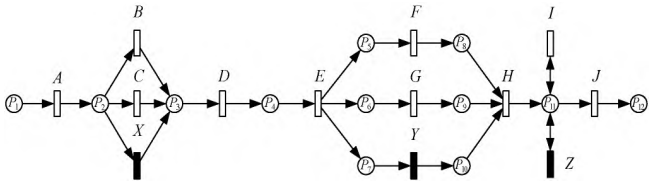


图 9　含有选择、并发和循环结构的模型

3.1　对齐耗时

为对动态规划求解耗时情况进行分析，在此取长度 5∶5∶40 的序列进行对齐，对齐耗时结果如图 10 所示。对齐计算的耗时与序列长度成正比，主要耗时为打分矩阵的构建（算法复杂度为 $O(mn)$）与序列对齐结果的回溯（时间复杂度为 $O(m+n)$）。

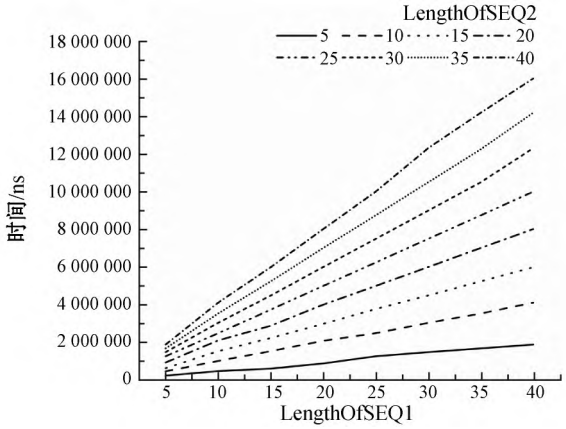
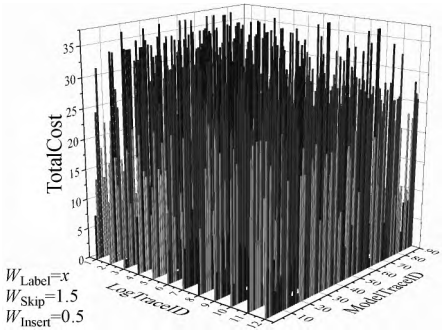


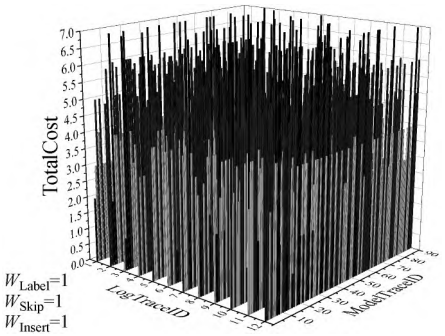
图 10　不同长度序列对齐计算耗时

3.2　可行性及有效性

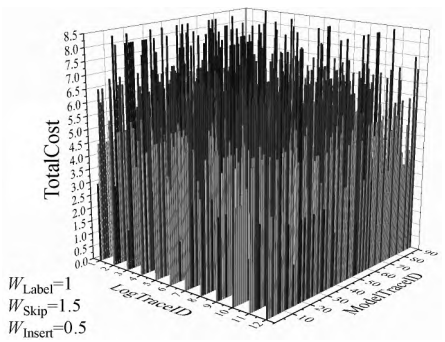
对于加权计算对齐成本方法的可行性，利用表 1 所示的日志与图 9 所示网系统进行对齐，并计算对齐成本（活动成本分别为默认 1 和（A,1,B,2,C,2,X,4,D,1,E,1,F,1,G,1,Y,1,H,1,I,6,Z,12,J,1）两种），插入及跳过成本权重设置为默认 1 和（$W_{Skip}=1.5$，$W_{Insert}=0.5$），对齐成本计算结果如图 11 所示，方法是可行的。



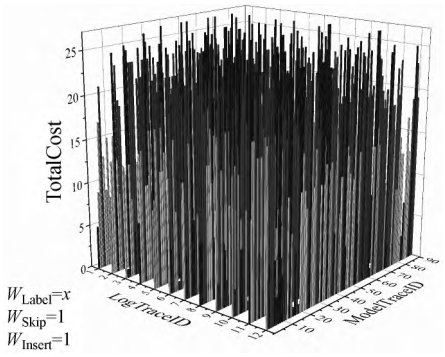
（a）



（b）



（c）



(d)

图 11 不同标签权重、插入跳过成本权重的日志与模型对齐成本

图 12 展示了不同标签取值对 $Cost_{Skip}$、$Cost_{Insert}$的影响，$Weight_{Skip}$、$Weight_{Insert}$的设定增加了对齐成本线图的可区分度。图 13 中的线为对齐不同可观测迹的对齐总成本，非重叠处即出现了不同权值设置标签的插入或跳过。

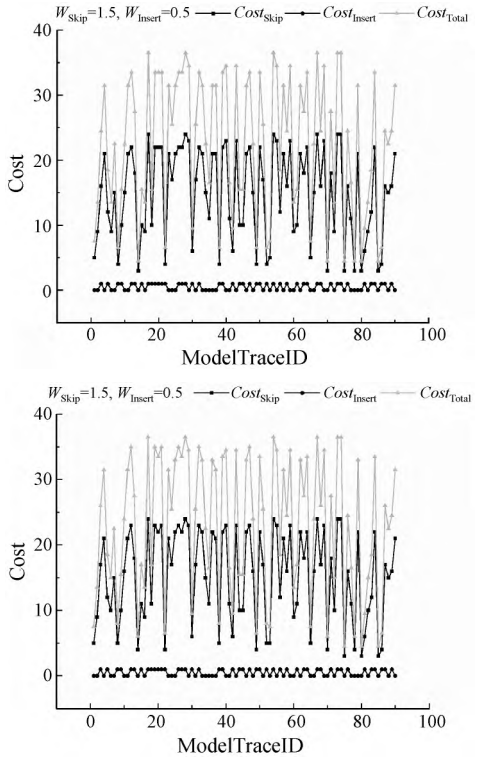


图 12 设置不同标签权重 ADEFGHJ 与模型可观测迹的对齐成本

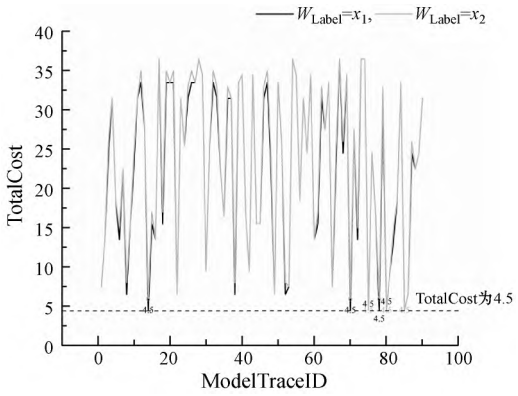


图 13 不同标签权重对 ADEFGHJ 最优对齐的影响

表 2 与表 3 展示了日志序列 < ADEFGHJ > 在不同标签权值设定时(将选择结构中的 B,C 活动标签加入不同权值)，最优对齐及对齐成本会有所不同，方便在进行日志与模型对齐时对某些活动或结构进行侧重，方法是有效的。

x_1 = A,1,B,2,C,3,X,4,D,1,E,1,F,1,G,1,Y,1,H,1,I,6,Z,12,J,1

表 2 标签权值序列为 x_1 时最优对齐序列

log	model	ID	$Cost_{skip}$	$Cost_{insert}$	$Cost_{total}$
A→DE→FGHJ	ABDEYFGHJ	75	3	0	4.5
A→DEF→GHJ	ABDEFYGHJ	80	3	0	4.5
A→DEFG→HJ	ABDEFGYHJ	85	3	0	4.5

x_2 = A,1,B,2,C,2,X,4,D,1,E,1,F,1,G,1,Y,1,H,1,I,6,Z,12,J,1

表 3 标签权值序列为 x_2 时最优对齐序列

log	model	ID	$Cost_{skip}$	$Cost_{insert}$	$Cost_{total}$
A→DE→FGHJ	ACDEYFGHJ	14	3	0	4.5
A→DE→FGH→J	ACDEFYGHJ	70	3	0	4.5
A→DEFG→→H→J	ABDEYFGHJ	75	3	0	4.5
A→DE→FGH→→J	ACDEFGYHJ	78	3	0	4.5
A→DEFG→→H→J	ABDEFYGHJ	80	3	0	4.5
A→DEF→GH→J	ABDEFGYHJ	85	3	0	4.5

4 结 语

本文认为一致性检验在信息管理系统中发挥着越来越重要的作用，其中对齐是最先进、最全面的方法之一，且最优对齐被广泛使用。现有的最优对齐计算缺乏对活动依赖的考虑，继而在此提出一种新的方法，用于计算 Petri 网模型和日志迹之间的最优对齐。首先基于动态规划求解对齐(时间复杂度 $O(mn+m+n)$)，然后对于对齐结果依据自定义的活动权重及插入、跳过成本的权重计算对齐成本(时间复杂度 $O(mn)$)，最后对计算结果分组排序得出模型与日志迹之间的每一对最优对齐。实验结果表明动态规划求解对齐、加权计算对齐成本的方法是可行的，且最优对齐的求解方法在一定程度上体现了不同活动的重要性，增加了最优对齐对活动的依赖性，且降低了日志迹与模型之间相同最优对齐成本的个数，对原有的单一量化对齐成本求解进行了改进。

依据实验结果分析，未来工作可从以下几个方面展开：(1) 利用 BPIC 的真实数据和模型对方法进行测试和验证；(2) 改进加权计算对齐成本的方法，将其与

对齐的求解相结合; (3) 在求解所有对齐、排序最优对齐时加入多线程的使用提高计算效率; (4) 加权计算后得到的对齐成本是不同的，进一步分析其对一致性检验的影响。

参 考 文 献

[1] Aalst W, Hofstede T, Weske M. Business process management: A survey [C] //International Conference on Business Process Management, 2003: 1 – 12.

[2] Aalst W. Business process management: A comprehensive survey [J]. International Scholarly Research Notices, 2013, 2013: 507984.

[3] Buijs J. Mapping data sources to XES in a generic way [D]. Eindhoven University of Technology, 2010: 63 – 76.

[4] Rozinat A, Aalst W. Conformance checking of processes based on monitoring real behavior [J]. Information Systems, 2008, 33(1): 64 – 95.

[5] Weidlich M, Polyvyanyy A, Desai N, et al. Process compliance analysis based on behavioural profiles [J]. Information Systems, 2011, 36(7): 1009 – 1025.

[6] Weber P, Bordbar B, Tino P. A framework for the analysis of process mining algorithms [J]. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2013, 43(2): 303 – 317.

[7] Song W, Jacobsen H, Ye C, et al. Process discovery from dependence-complete event logs [J]. IEEE Transactions on Services Computing, 2015, 9(5): 714 – 727.

[8] Bose R, Aalst W. Process diagnostics using trace alignment: Opportunities, issues, and challenges [J]. Information Systems, 2012, 37(2): 117 – 141.

[9] Aalst W, Adriansyah A, Dongen B. Replaying history on process models for conformance checking and performance analysis [J]. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 2012, 2(2): 182 – 192.

[10] Fahland D, Aalst W. Repairing process models to reflect reality [C] //International Conference on Business Process Management, 2012: 229 – 245.

[11] Aalst W, Stahl C. Modeling business processes: A petri net-oriented approach [M]. MIT Press, 2011: 13 – 75.

[12] Adriansyah A, Dongen B, Aalst W. Towards robust conformance checking [C] //International Conference on Business Process Management, 2010: 122 – 133.

[13] Rozinat A. Process mining: Conformance and extension [D]. Eindhoven University of Technology, 2010.

[14] Adriansyah A, Dongen B, Aalst W. Conformance checking using cost-based fitness analysis [C] //2011 IEEE 15th International Enterprise Distributed Object Computing Conference, 2011: 55 – 64.

[15] Verbeek H. Decomposed replay using hiding and reduction as abstraction [M] //Transactions on Petri Nets and Other Models of Concurrency XII. Springer, 2017: 166 – 186.

[16] Adriansyah A. Aligning observed and modeled behavior [D]. Eindhoven University of Technology, 2014.

[17] Leoni M, Maggi F, Aalst W. Aligning event logs and declarative process models for conformance checking [C] //International Conference on Business Process Management, 2012: 82 – 97.

[18] 田银花，杜玉越，韩咚，等. 基于 Petri 网基本结构的相似最优校准计算方法 [J]. 计算机集成制造系统, 2016, 22(2): 433 – 447.

[19] Adriansyah A, Dongen B, Aalst W. Cost-based conformance checking using the A* algorithm [R]. BPM Center Report, 2011.

[20] Polyvyanyy A, Aalst W, Hofstede A, et al. Impact-driven process model repair [J]. ACM Transactions on Software Engineering and Methodology, 2016, 25(4): 1 – 60.

[21] Fahland D, Aalst W. Model repair—aligning process models to reality [J]. Information Systems, 2015, 47: 220 – 243.

[22] Macedo N, Jorge T, Cunha A. A feature-based classification of model repair approaches [J]. IEEE Transactions on Software Engineering, 2016, 43(7): 615 – 640.

[23] Eck M, Buijs J, Dongen B. Genetic process mining: Alignment-based process model mutation [C] //International Conference on Business Process Management, 2014: 291 – 303.

[24] 邵叱风，方贤文，盛梦君. 基于增强日志的过程挖掘算法 [J]. 安徽理工大学学报(自然科学版), 2020, 40(4): 25 – 32.

[25] 邵叱风. 算术计算 Petri 网模型及实现 [J]. 赤峰学院学报(自然科学版), 2020, 36(8): 21 – 24.

(上接第 206 页)

[20] Wang H W, Zhang F Z, Zhang M, et al. Knowledge-aware graph neural networks with label smoothness regularization for recommender systems [C] //25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019: 968 – 977.

[21] Kipf T N, Welling M. Semi-Supervised classification with graph convolutional networks [C] //5th International Conference on Learning Representations, 2017.

[22] Wang X, Jin H Y, Zhang A, et al. Disentangled graph collaborative filtering [C] //43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, 2020: 1001 – 1010.

[23] Sun J N, Zhang Y X, Guo W, et al. Neighbor interaction aware graph convolution networks for recommendation [C] //43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, 2020: 1289 – 1298.