数据流中随机型分形维数计算方法研究

倪志伟 公维峰 周之强 唐李洋

（合肥工业大学管理学院 合肥 230009）（过程优化与智能决策教育部重点实验室 合肥 230009）

摘 要 分形维数能够有效地描述数据集，反映复杂数据集中隐含的规律性，基于分形理论的数据挖掘算法通常都涉及到分形维数的计算。但是现有的分形维数计算方法的时间复杂度和空间复杂度都比较高，大大降低了算法的效率，使算法很难适应高速、海量的数据流环境。因此，总结分析了现有的几种分形维数计算方法，并提出一种随机型方法，利用固定的内存空间快速估计数据流的关联维数。最后通过与现有算法进行对比实验，证明了这一随机型算法的有效性。

关键词 分形，分形维数，数据流

中图法分类号 TP301.6 **文献标识码** A

Research of Stochastic Fractal Dimension Calculation Algorithm in Data Stream

NI Zhi-wei GONG Wei-feng ZHOU Zhi-qiang TANG Li-yang

(School of Management, Hefei University of Technology, Hefei 230009, China)

(Key Laboratory of Process Optimization and Intelligent Decision-making, Ministry of Education, Heifei 230009, China)

Abstract Fractal dimension can describe the data set effectively and can reflect the hidden regularity of the complex data set. Data mining algorithms based on fractal theory are usually related to the calculation of fractal dimension. But most of the existing fractal dimension calculation algorithms are with high time complexity and space complexity, which greatly reduces the efficiency and is not applicable for data stream with high-speed and massive data. In this paper, several existing fractal dimension calculation algorithms were analyzed and a stochastic fractal dimension calculation algorithm were proposed to fast estimate the correlation dimension in fixed space. The comparative experiment and analysis demonstrate the effectiveness of this stochastic fractal dimension calculation algorithm.

Keywords Fractal, Fractal dimension, Data stream

1 引言

分形理论是现代非线性科学研究中十分活跃的一个数学分支，它利用整体与局部间具有的自相似性，揭示复杂现象中所蕴含的规律。分形维数是描述具有分形特征物体的重要指标，可以定量地分析分形集的复杂程度。近几年，许多学者的研究表明[2-5]分形维数在数据挖掘领域有着非常特殊的作用，将分形技术应用于数据挖掘领域能够更好地克服传统数据挖掘技术的不足，有效地解决在结构复杂、高维数据集上的数据挖掘问题[1]。目前，对于快速到达、潜在无限的数据流环境的挖掘已经成为数据挖掘的一个重要的研究方向，而分形维数计算的低效使得在数据流挖掘中应用分形理论面临着困难。在数据流的挖掘中，用精度换取效率是一种常用的方法。同时，准确的分形维数很难得到，人们往往更关心分形维数的相对大小和分形维数的变化情况，计算分形维数时常用的盒计数法(box-counting)计算出来的分形维数本身就是对分形维数的一种估计。因此，在数据流环境中，对分形维数进行高效和准确的估计具有重要的意义。

2 相关研究

分形是自然界中普遍存在的现象。分形体具有自相似性，即局部与整体相似。大量实际数据集具有分形特征，即数据集的部分分布与整体分布有相似的结构或属性。在实际数据集中，这种自相似性一般表现为统计意义上的自相似性。分形理论的创始人 Mandelbrot 认为分形集这类奇异集合的性质不能用欧氏测度来衡量，但维数恰是此类集合尺度变化下的不变量，因此主张用维数来刻画这类集合。

定义 1（分形维数） 对于一个在区间$[r_{\min}, r_{\max}]$（无标度区间）内呈现统计自相似特征的 E 维数据集，数据点落在边长为 r 的 E 维单元格内的概率为 p_i，则分形维数为

$$D_q=\frac{1}{q-1}\frac{\partial \log \sum_i p_i^q}{\partial \log r} r\in[r_{\min}, r_{\max}] \quad (1)$$

当 $q=0$ 时为豪斯道夫维数，$q\to 1$ 时为信息维，而 $q=2$ 时为关联维。

计算一个 E 维数据集的分形维数时，首先将数据集划分为 L 层网格结构。第一层对每一维 2 等分，得到 2^E 个 E 维

到稿日期：2010-05-13 返修日期：2010-08-04 本文受国家自然科学基金(70871033,70801025)，国家高技术研究发展计划(863)(2007AA04Z116)，合肥工业大学校科学研究发展基金(2009HGXJ0040)资助。

倪志伟(1963－)，男，教授，博士生导师，主要研究方向为人工智能、机器学习；公维峰(1986－)，男，硕士生，主要研究方向为分形与数据挖掘；周之强(1987－)，男，硕士生，主要研究方向为联机分析挖掘。

网格；第二层对每一维4等分，得到2^{2E}个E维网格；直到第L层，对每一维L等分，得到2^{LE}个E维网格。然后计算每一层划分中数据点落入每一非空网格的概率$p_i=c_i/n$，其中c_i为第i个网格中落入的数据点数，n为总数据点数。根据式(1)拟合得到分形维数。

对于实时的、连续的、潜在无界的事件序列构成的数据流，由于不能将快速到达的数据全部存在主存中，因此与计算静态数据集分形维数相比，要求其算法时间和空间上更高效。此外，数据流上计算分形维数还要满足以下几点要求：

1)新到达的数据可能超出了原有的数据取值范围，要能够进行简单快速的调整使取值范围适应新数据。

2)能够满足人们的查询要求，根据输入的查询参数输出指定查询范围内的分形维数。

基于分形理论的数据挖掘算法一般都离不开分形维数的计算，研究者们在提出算法的同时也提出了不同分形维数的计算方法。同时有研究者专门针对分形维数的计算方法进行了研究。这些算法可以分为两大类：一类是确定型算法，另一类是随机型算法。

确定型算法是指在计算分形时能准确得到落入每个非空网格中的数据点的个数。文献[2-6]中的算法都属于确定性算法。文献[2-5]中的算法只适用于静态数据集，其中文献[2,3]中的方法比较具有代表性，前者利用树结构存储所有非空网格信息，后者利用基于Z-ordering编码的网格结构存储所有非空网格信息。文献[6]在树结构的基础上引入了滑动窗口，使算法适用于数据流环境，但只能支持较小的窗口。

这些确定型算法的共同点在于最终都得到了每个非空网格中的数据点数，区别主要是使用了不同的数据结构。每一层网格结构中，这类算法要保存非空网格中的数据点数，至少需要$O(n*\log n)$的空间，其中n为非空网格个数；还要保存每个网格的坐标，需要$O(E)$的空间，其中E为数据集维数。在海量的数据流中，非空网格的数量也将非常巨大，因此确定型算法计算数据流的分形维数只能通过一个较小的滑动窗口来实现，对数据流的考察范围比较有限。

随机型算法是指在划分出来的每一层网格结构中，使用一些随机算法来估计$\sum f_i^q$，其中f_i为落入每个非空网格中的数据点数。为达到这一目的，一般构建随机变量X，使$E(X)=\sum f_i^q$，$D(X)$较小，通过切比雪夫不等式可知，得到的估计值在较大概率下与真实值的误差可以满足阈值要求。文献[7]中的方法属于随机型算法，使用固定的空间估计关联维数，其理论来源为文献[8]的tug-of-war思想。但是算法不够灵活，只能计算整个数据流的分形维数，而且更新数据时要同时更新所有s_1*s_2个计数器。

确定型算法和随机型算法比较：

1)确定型算法计算出的分形维数更准确，但需要较高的时间复杂度和空间复杂度。

2)随机型算法只需要固定的空间和较低的时间复杂度，更适用于数据流环境。

3)确定型算法对于q的取值不敏感，使用相同的数据结构可以计算出q取不同值时的分形维数；随机型算法在$q\leqslant 2$时比较简单，随着q增大算法复杂度明显提高。

3 基于窗口计数器的多层CountSketch结构

为了可以根据输入的查询参数估算指定查询范围内的分形维数，本文引入了基于窗口计数器的多层CountSketch结构，使用到的基本定义如下。

定义2(k-universal哈希函数[9-11]) 首先定义$[n]=\{0,1,\cdots,n-1\}$，如果一组从$[n]$映射到$[m]$的哈希函数为k-universal哈希函数族，则满足以下条件：对任意不相等的$x_0,x_1,\cdots,x_{k-1}\in[n]$和任意的$v_0,v_1,\cdots,v_{k-1}\in[m]$，

$$\Pr_{h\in H}\{h(x_i)=v_i,\forall i\in[k]\}=1/m^k$$

定义3(窗口计数器) 窗口计数器是这样的一种数据结构：它包含计数器最后一次更新的时标T_u和循环列表List，并采用延迟更新机制来避免频繁的维护，即在更新计数器或输出结果时，将过期窗口对应的数据设为0，再将当前窗口对应的信息保存到List的相应位置。

定义4(改进的CountSketch结构) 改进的CountSketch结构是由$t\times m$个窗口计数器counter、t个包含m个桶的哈希表和t个从对象到$\{1,\cdots,m\}$的4-universal哈希函数组成的数据结构。其中t个哈希表的每个元素是一个窗口计数器。

与文献[12]中定义的CountSketch结构相比，该结构减少了t个从对象映射到$\{1,-1\}$的4-universal哈希函数，并且使用窗口计数器代替了整数计数器。

对于一个支持w个基本窗口的窗口计数器，其循环列表List包含w个元素，每个元素可以对应一个大小为$Size_b$的基本窗口。如果单独使用具有窗口计数器的CountSketch结构，面临的问题是如果基本窗口过小，要支持较大的查询粒度时就需要多个窗口，这使得计数器占用的空间变大，更新速度变慢。同样，如果使用的基本窗口过大，又降低了查询的精度。为了解决这一矛盾，提出了基于窗口计数器的多层CountSketch结构。其基本结构如图1所示。

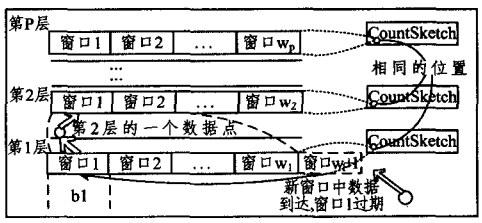

图1 多层CountSketch结构图

不难发现CountSketch结构具有可加性，即对于连续的数据流段$S_1,S_2,\cdots,S_n$，其所对应的CountSketch结构为$CS_1,CS_2,\cdots,CS_n$。如果其参数都相同，则将$CS_1,CS_2,\cdots,CS_n$对应位置相加，得到新的CountSketch结构CS，则CS与数据流$S=\{S_1,S_2,\cdots,S_n\}$产生的具有相同参数的CountSketch结构相同。因此可以使用分层结构，根据输入的参数建立起P层具有相同参数的CountSketch的分层结构。每一个CountSketch结构中使用包含w_i个大小为b_i的基本窗口的窗口计数器，其中$i\in\{1,2,\cdots,P\}$。只有最底层的CountSketch需要t个哈希函数处理原始数据流，其它层不需要哈希函数。在除最高层以外的任何一层中，如果一个窗口过期，则作为上一层的一个点将这个窗口中的数据加到上一层对应位置的计数器中。当需要输出结果时，将查询参数N分解，得到所有包含在查询范围内的窗口。将各层查询范围内窗口中的数据相加，合并成一个单一的CountSketch结构。

4 基于多层CountSketch结构的随机型关联维数算法

随机型算法计算关联维数时，划分完L层网格结构后，

需要计算每一层划分中$\sum_i p_i^2$的值。由于$\sum_i p_i^2=\sum_i \frac{C_i^2}{N^2}=\frac{1}{N^2}\sum_i C_i^2$，$C_i$为每个非空网格中落入的数据点数，$N$为总点数。其中$N$可以通过计数的方式获得，因此需要估计的是$F_2=\sum_i C_i^2$。4.1节介绍在每一层网格结构中，如何基于窗口计数器的多层CountSketch结构估计F_2。4.2节介绍如何通过F_2的估计算法估计数据流的关联维数。

4.1 F_2估计算法

定义5(F_2的(ε,δ)估计) 对于给定的参数$0<\varepsilon<1$和$0<\delta<1$，得到F_2的估计值$\hat{F}_2$，使得$\Pr(|\hat{F}_2-F_2|>\varepsilon F_2)<1-\delta$，叫做$F_2$的$(\varepsilon,\delta)$估计。即以大于$\delta$的概率保证估计值$\hat{F}_2$与真实值$F_2$之间的误差不超过$\varepsilon F_2$。

通过输入不断到达的数据的坐标，可以得到F_2的(ε,δ)估计。具体过程如下：

Step1 建立多层CountSketch结构，参数为t和m。h_1，h_2，…，h_t为相互独立的从坐标到$\{1,\cdots,m\}$的4-universal哈希函数。

Step2 插入数据，执行Step3；输出结果，执行Step4。

Step3 t个哈希函数h_j，$j\in(1,t)$分别将数据坐标$g_i=\{g_{i1},g_{i2},\cdots,g_{iE}\}$映射到$\{1,2,\cdots,m\}$，更新最底层CountSketch结构中的$t$个计数器，使$counter_{h_j(g_i)}+=1$。

Step4 多层结构输出结果，即所有包含在查询范围内的窗口合并成一个CountSketch结构。由其中的t个哈希表根据式(2)计算出t个估计值X_1，X_2，…，X_t，返回它们的中位数作为估计结果。

$$X=\frac{m+1}{m}\sum_{i\in[m]} counter_i^2-\frac{1}{m}\left(\sum_{i\in[m]} counter_i\right)^2 \tag{2}$$

式中，$counter_i$为哈希函数映射到哈希表第i个位置的元素个数。用a，b代表任意可能的非空网格，f_a，f_b为其中落入的点数，可知

$$\begin{aligned}\sum_{i\in[m]} counter_i^2&=\sum_{h(a)=h(b)} f_a f_b\\&=\sum_{a=b} f_a^2+\sum_{a\neq b\wedge h(a)=h(b)} f_a f_b\left(\sum_{i\in[m]} counter_i\right)^2\\&=\sum_{a,b} f_a f_b\\&=\sum_{a=b} f_a^2+\sum_{a\neq b\wedge h(a)=h(b)} f_a f_b+\sum_{h(a)\neq h(b)} f_a f_b\end{aligned}$$

于是$X=F_2+\sum_{a\neq b\wedge h(a)=h(b)} f_a f_b-\frac{1}{m}\sum_{h(a)\neq h(b)} f_a f_b$。根据k-universal哈希函数的概念，可以得出[9]

$$E(X)=F_2,\ D(X)=\frac{2}{m}(F_2^2-F_4)$$

$$\Pr(|X-F_2|>\varepsilon F_2)<\frac{D(X)}{\varepsilon^2 F_2^2}<\frac{2F_2^2}{m\varepsilon^2 F_2^2}$$

令$m=\frac{2}{\varepsilon^2(1-\delta)}$，可以得到$X$为$F_2$的$(\varepsilon,\delta)$估计。

4.2 随机型关联维数算法

为了计算关联维数，首先将E维数据流X划分为L层网格结构。设X的第j维的最大值为$\max_j$，最小值为$\min_j$，则X第j维的取值空间长度$R_j=\max_j-\min_j$。对于一个新到的数据点$x_i=\{x_{i1},x_{i2},\cdots,x_{iE}\}$，计算其在每一层网格结构中的坐标。在第$k$层划分中，$k\in\{1,2,\cdots,L\}$，网格的边长$r_{kj}=R_j/2^{k-1}$，$x_i$坐标编码为$(g_1,g_2,\cdots,g_E)$，$g_i$根据式(3)得到。将计算出的数据点在每一层网格结构中的坐标作为F_2估计算法的输入，计算出每层网格结构中F_2的估计值，进而根据式(1)拟合得到关联维数的估计值。

$$g_i=\left\lceil\frac{x_{ij}-M_j}{r_{kj}}\right\rceil \tag{3}$$

式中，$M_j=\min_j+\frac{R}{2}$。

数据流中，计算一个新到达的数据点在每一层网格结构中的坐标时，可能出现数据点取值超出原有取值范围的情况，这需要对取值空间进行调整。具体方法如下：

输入：每一维的最大值$\max_j$，最小值$\min_j$，数据点x_i

输出：调整后的$\max_j$，$\min_j$，L

```
While(点超出取值空间)
{
For j=1 to E
  If(x_ij>max_j or x_ij<min_j)
    L=L+1;
    For k=1 to E
        min_k=min_k-R_k/2;
        max_k=max_k+R_k/2;
        R_k=2*R_k
    End for
  End if
End for
}
```

综上所述，基于多层CountSketch结构的随机型关联维数算法的具体过程如下：

Step1 初始化算法，得到每一维初始的最大值和最小值。

Step2 将E维空间划分为L层网格结构。

Step3 为每一层网格结构构建估计F_2所需的多层CountSketch结构。

Step4 对于每一个新到达的数据点，检查其是否超出取值范围，超出取值范围时做相应调整。计算出数据点在每一层网格结构中的坐标。

Step5 将坐标插入每一层网格结构的多层CountSketch结构中。

Step6 计算出每一层网格结构中F_2的估计值。

Step7 由每一层网格结构中F_2的估计值计算出$\sum_i p_i^2$，根据式(1)拟合得到结果。

5 实验验证

实验部分将本文的算法与文献[6]中的SID-meter算法、文献[3]中的ZBMFD算法以及文献[7]中的TOW算法进行了对比。其中TOW算法属于随机型算法，其余两种算法属于确定型算法。5.1节主要进行计算结果精度方面的对比。SID-meter和ZBMFD属于确定型算法，它们输出的分形维数相同。这一结果是应用中最常用到的。通过将这一结果与本文算法代表的随机型算法的计算结果进行比较，证明了本文的随机型算法的计算结果与确定型算法的计算结果很接近。5.2节主要进行几种算法的时间空间效率的对比。

5.1 结果精度对比实验

实验1 在Sierpinkski三角数据集对算法进行了测试，并与确定型算法进行了比较。首先生成5000个点的Sierpinkski数据集，并将所有点包含在窗口内。随机生成5个4-universal哈希函数。输出的5组估计值和确定型算法输出的

估计值如图 2 所示。从图中可以看出确定型算法输出值的实线被多条虚线覆盖，只有一条虚线产生了较大的偏差。对 5 组输出数据取中位数，去除了偏差曲线的影响，输出的分形维数为 1.5580，SID-meter 和 ZBMFD 算法输出的结果相同，为 1.5578，TOW 算法得到的结果为 1.520，Sierpinkski 理论上的分形维数为 $\log_2 3$。几种算法的计算结果与理论值相差不大。

实验 2　为了进一步测试对数据流分形维数的计算，合成了这样一个数据流，其前 100000 条数据产生于科赫曲线，后 100000 条数据产生于 Skierpinkski 三角。每隔 40000 计算一次分形维数，考察分形维数随数据流动而产生的变化。实验结果如图 3 所示。

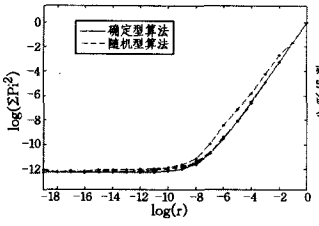

图 2　Sierpinkski 数据集 log-log 比较图

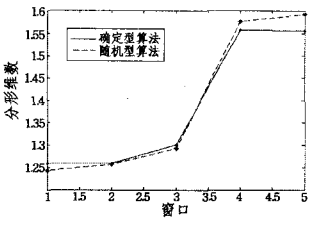

图 3　合成数据流分形维数变化比较图

实验 3　使用中国深市 1991 年 4 月 3 日至 2009 年 10 月 26 日每天开盘、收盘、最高价、最低价、交易量、交易金额组成的 6 维数据集进行了测试，由确定型算法和随机型算法分别计算出每一层网格结构 F_2 的估计值，考察 F_2 的估计值和真实之间的差距，结果如图 4 所示。可以看出 2 条曲线基本重合，两者之间差距很小。每隔 500 条数据计算一次分形维数，结果如图 5 所示。

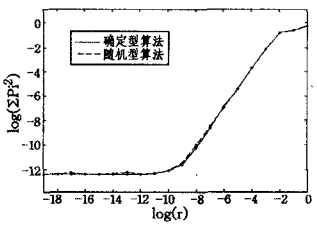

图 4　股票数据集 log-log 比较图

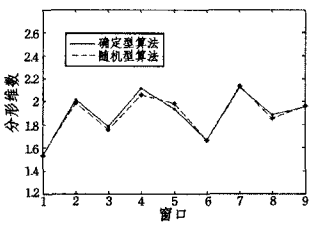

图 5　股票数据集分形维数变化比较图

实验 4　使用 UCI 数据集 Character Trajectories 前 400 个样本中的 65000 条数据进行实验的结果如图 6 所示。实验分别取数据集中的前两个属性和全部 3 个属性进行，可以看出，随着属性的增加，分形维数增大。

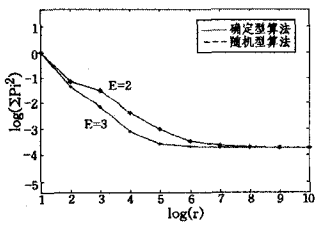

图 6　Character Trajectories 数据集 log-log 比较图

5.2　算法效率分析

在算法效率方面，随机型算法时间和空间效率明显高于确定型算法，且本文算法优于已有的随机型算法 TOW 算法。

空间上，若最大划分层次为 L，L 一般较小，第 L 层网格结构中非空网格个数最多为数据点数 n。SID-meter 算法需要保存整棵分形树，分形树每一层的结点个数最多为 n，需要的空间为 $O(n*L)$。ZBMFD 保存的是最底层的网格结构，相当于保存了分形树最底层的结点，需要的空间为 $O(n)$。这两种确定型算法所需空间随着非空网格个数的增加而增大。而非空网格个数最多等于总数据点数，因此只能通过较小的滑动窗口来保存有限的数据点。本文算法需要的空间为 $O(t*m*P*L)$，其中 P 为多层结构的层次，t 和 m 为 Count Sketch结构的两个参数，相对于 n 而言，所需的空间为 $O(1)$，与保存的数据点的个数无关。TOW 算法所需的空间为为 $O(s_1*s_2*L)$，其中 s_1*s_2 为所用的计数器的个数，相对于 n 而言所需的空间同样为 $O(1)$。

时间上，计算分形维数分为两个阶段：第一阶段为插入数据点，第二阶段为计算分形维数。若第 L 层网格结构中非空网格个数为 n，在插入数据点过程中，ZBMFD 算法插入一个数据点时，需要查找要插入的点是否落入现有的非空网格。如果落入现有的非空网格，则直接更新该网格信息，否则建立一个新的非空网格。这一过程需要的时间为 $O(\log n)$。SID-meter 算法每一层网格结构中都需要做这样的判断，且每一个网格包含的下一层非空网格最多为 n，因此需要的时间为 $O(L*\log n)$。TOW 算法插入一个数据点需要计算 s_1*s_2*L 个哈希函数的值，假定计算哈希函数的时间为 $O(1)$，则 TOW 算法需要的时间为 $O(s_1*s_2*L)$，对于 n 而言需要的时间为 $O(1)$。本文算法插入一条数据需要计算 $t*L$ 个哈希函数的值，需要的时间为 $O(t*L)$，对于 n 而言需要的时间同样为 $O(1)$。同时由于 t 小于 s_1*s_2，因此本文算法优于 TOW 算法。在第二阶段计算分形维数过程中，ZBMFD 算法需要逐层合并出上层网格，由当前网格映射到上层网格需要的时间为 $O(n+n*\log n)$，可以认为是 $O(n*\log n)$，映射的次数取决于划分的层数，因此需要的时间为 $O(L*n*\log n)$。SID-meter 算法只需要对分形树遍历进行累加，分形树最多有 $L*n$ 个结点，需要的时间为 $O(L*n)$。TOW 算法需要对 s_1*s_2 个计数器进行统计，本文算法需要对 $t*m$ 个计数器进行统计，两者效率相当，但计数器的个数明显少于数据点数，因此两种随机型算法的效率要高于两种确定型算法。

因此无论在空间效率还是时间效率，随机型算法 TOW 和本文算法都优于确定型算法 SID-meter 和 ZBMFD。TOW 算法和本文算法在空间效率上相当，在时间效率上本文算法优于 TOW 算法。

结束语　现有的分形维数计算方法大体可以分为确定型算法和随机型算法两类。本文着重对随机型算法进行研究并提出了一种灵活高效的随机型分形维数算法。随机型分形维数算法通过构建一组期望等于各非空网格中数据点数的 q 次幂之和且方差较小的随机变量，得到分形维数的估计值。这一估计值以很高的概率与确定型算法算出的分形维数十分接近，从而可以有效地估计分形维数且时间复杂度和空间复杂度明显低于确定型分形维数计算方法，更适合快速、潜在无限的数据流的挖掘。研究广义分形维数的估算方法和应用随机型分形维数计算方法挖掘数据流中隐藏的规律，将是今后研究的重点。

参 考 文 献

[1]　倪丽萍，倪志伟，吴昊，等. 基于分形维数的数据挖掘技术研究综述[J]. 计算机科学，2008，35(1)：187-189

（下转第 229 页）

$T_P=\max\{0.36028,0.25\}=0.36028(s)$

剩下的任务加上这个并行块是顺序的，因此由式(3)可计算出第3个实例子图的响应时间为

$T_3=0.25+0.2+0.36028+0.053=0.86328(s)$

同样地，其它3个实例子图的响应时间为

$T_1=0.795(s)$，$T_2=0.82(s)$，$T_4=1.153(s)$

最后，整个模型的完成时间为

$T=0.12\times0.795+0.28\times0.82+0.18\times0.86328+0.42\times1.153=0.9646504\approx0.96(s)$

模型的性能分析完毕。

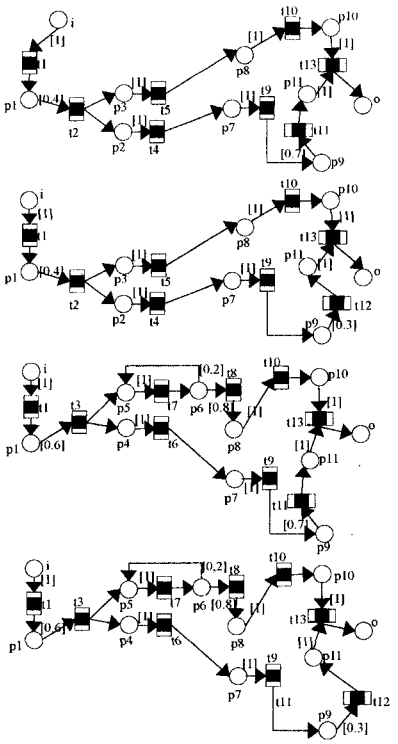

图4 实例子图

结束语 本文将WF-net概念做了延伸，把Poisson分布的到达时间和指数分布的服务时间同每个任务联系在一起，给出了随机良构工作流网的定义。结合排队网络的知识将资源到达情况及任务执行情况结合起来描述一个任务，推导出了每个任务的响应时间、每种工作流模式响应时间的计算公式，结合模型的分解算法推导出了实例子图的响应时间，最终结合路由率定义计算出了整个SWWF-net模型的完成时间。本文最后结合实例给出模型分解算法和相应的分析，推导出了随机良构工作流网模型的平均响应时间。在今后的研究中将继续探讨及改进分析算法。

参考文献

[1] Tan Zhangxi, Lin Chuang, Yin Hao, et al. Approximate Performance Analysis of Web Services Flow Using Stochastic Petri Net[C]// Proc. of Third International Conferencem Grid and Cooperative Computing(GCC 2004). 2004:193-200

[2] Son J H, Kim J S, Kim M H. Extracting the workflow critical path from the extended well-formed workflow schema[J]. Journal of Computer and System Sciences, 2006, 70(1):86-106

[3] van der Aalst W, van Hee K. Workflow Management: Models, Methods, and Systems[M]. The MIT Press, 2002:271-272

[4] Aalst W. Workflow Verification: Finding Control-flow Errors Using Petri-net-based Techniques[M]. Springer-Verlag, 2000:161-183

[5] Sadiq W, Orlowska M. Applying Graph Reduction Techniques for Identifying Structural Conflicts in Process Models[C]// Proceedings of the 11th International Conference on Advanced Information Systems Engineering. 1999:195-209

[6] Workflow Management Coalition Terminology and Glossary(WFMC-TC-1011)[R]. Workflow Management Coalition, Brussels, 1996

[7] Sadiq W, Orlowska M E. Analyzing process models using graph reduction techniques[J]. Information Systems, 2000, 25(2):117-134

[8] Li Jianqiang, Fan Yushun, Zhou Mengchu. Performance Modeling and Analysis of Workflow[J]. IEEE Transactions on Systems, Man, and Cybernetics, 2005, 34(2):229-242

[9] Kleinrock L. Queueing Systems: Computer Applications[M]. NewYork: Wiley, 1974

[10] Trivedi K S. Probability and Statistics with Reliability, Queuing, and Computer Science Applications[M]. Prentice-Hall, Inc., 1982

(上接第212页)

[2] 鲍玉斌，王琢，孙焕良，等. 一种基于分形维的快速属性选择算法[J]. 东北大学学报：自然科学，2003，24(6)：527-530

[3] 闫光辉，李战怀，党建武. 基于多重分形的聚类层次优化算法[J]. 软件学报，2008，19(6)：1283-1300

[4] Traina C, Traina A, Wu L, et al. Fast feature selection using fractal dimension[C]// Proc. XV Brazilian Symposium on Databases. 2000

[5] Barbará D, Chen P. Using the Fractal Dimension to Cluster Datasets [C]// Proceedings of the Sixth ACM SIGKDD. International Conference on Knowledge Discovery and Data Mining. 2000

[6] de Sousa E P M, Traina A J M, Traina J C. SID: Calculating the Intrinsic Dimension of Data Streams [C]// Proceedings of the 2006 ACM Symposium on Applied Computing. 2006

[7] Wong A, Wu L J, Gibbons P B, et al. Fast Estimation of Fractal Dimension and Correlation Integral on Stream Data[J]. Information Processing Letters, 2005, 93(2):91-97

[8] Alon N, Matias Y, Szegedy M. The space complexity of approximating the frequency moments[C]// 1996 ACM Symposium on Theory of Computing. Philadelphia, 1996:22-24

[9] Thorup M, Zhang Yin. Tabulation Based 4-Universal Hashing with Applications to Second Moment Estimation[C]// Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms. 2004, 18:608-617

[10] Anna O, Rasmus P. Uniform Hashing in Constant Time and Linear Space[C]// Proceedings of the Annual ACM Symposium on Theory of Computing. 2003:622-628

[11] Carter J L, Wegman M N. Universal Classes of Hash Functions [J]. Journal of Computer and System Sciences, 1979, 18(2):143-154

[12] Charikar M, Chen K, Farach-Colton M. Finding Frequent Items in Data Streams[J]. Theoretical Computer Science, 2004, 312(1):3-15