Unbelievable Quick Morris' Cluster Sampling

Haowei Li, MIS 1801, Business School, CSU

摘要

Morris方法为初步计算实验提供了行之有效的采样方法。然而，其生成簇采样矩阵的方法仍有效率提升的空间。本文提出了一种效率更优的簇采样矩阵方式，使得该簇采样矩阵可以以极优的时间复杂度生成，为后续的科学计算实验提供支持。

关键词

Morris方法 簇采样 递推算法 时间复杂度分析

Elementary Effect与簇采样矩阵分解

Elementary Effect（以下简成ee）用于表示不同因子之间的关系。当映射为采样0-1矩阵时，令每一矩阵行为一次抽样，每一矩阵列代表一个因子，则不同行之间形成ee可以表示为：两矩阵行之间有且只有1列因子不同。如下例：

$$\begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}$$

其中第1行$(0,0)$与第2行$(0,1)$在第2个因子上形成了1个ee，与第3行$(1,0)$在第1个因子上形成了1个ee，与第4行不形成ee。上述例子对每个因子共各自形成了2个ee。
分析0-1矩阵M的形成，约定M_k表示列数为k的矩阵。对矩阵M_k按1的数量分块，约定$M_{k,m}$表示k列情况下，每行有m个1的子矩阵。如上例的矩阵可以表示为M_2，可分解成如下$M_{2,0}$，$M_{2,1}$，$M_{2,2}$矩阵：

$$\begin{pmatrix} 0 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 \end{pmatrix}$$

由此，可得一有k+1个子矩阵构成的序列：$M_{k,0}$,$M_{k,1}$,...,$M_{k,k}$。根据ee的定义，可知子矩阵内部不同行之间必不可能产生ee，非相邻子矩阵之间也不可能产生ee，对全局ee情况的讨论转化为对相邻子矩阵之间ee情况的讨论，M各列形成的ee表示为相邻子矩阵ee数的总和。
进一步，0-1选取问题可映射为组合问题，令每列代表1个物品，则每个子矩阵可理解为从k个物品中抽取m个的所有组合情况。由组合数公式，可知$M_{k,m}$的行数=$C_k^m = \frac{k!}{k!-m!}$。
对组合情况子矩阵进行分析，可得$M_{k,m}$与$M_{k,m+1}$之间形成的每列ee数=C_{k-1}^m，简易证明如下：

证明：考虑$M_{k,m}$情况。
对$M_{k,m}$的每一行，有m个1，k-m个0，可以与$M_{k,m+1}$的k-m行形成ee；
又$M_{k,m}$有C_k^m行，则共形成$C_k^m \times (k-m)$个ee。
显然两个子矩阵之间形成的ee对每列是均匀的，
则每列各形成$C_k^m \times (k-m) \div k$=$C_{k-1}^m$。

由此，对k+1个子矩阵构成的序列$M_{k,0}$,$M_{k,1}$,...,$M_{k,k}$，其相邻两子矩阵所形成的ee数为C_{k-1}^0,C_{k-1}^1,...,C_{k-1}^{k-1}。

连续子矩阵子序列选取算法

对采样效率的优化问题可以映射成簇采样0-1矩阵的形成方式的优化。我们提出簇采样优化0-1矩阵M^*，优化的目标是在因子数k的情况下，指定要生成的各因子（各列）ee数量r，形成矩阵M^*，使得矩阵M^*满足r约束条件且采样数（行数）尽可能少。而根据上述分析，对子矩阵选取求ee可进一步转化为组合数连续子序列的选取，且对k列总矩阵而言，所能抽取的每列ee数位于区间$[1, sum(C_{k-1}^0,C_{k-1}^1,...,C_{k-1}^{k-1})]$中。基于上述讨论，本文先**假定**$r$的取值恰好为组合数某连续子序列之和，并进行算法求解。
观察组合数序列C_{k-1}^0,C_{k-1}^1,...,C_{k-1}^{k-1}，可得如下规律：

1. 所有序列元素均为正整数；
2. 序列呈现一定的单调性：从C_{k-1}^0,C_{k-1}^1,...,$C_{k-1}^{\lfloor \frac{k-1}{2} \rfloor}$，序列单调递增；从$C_{k-1}^{\lceil \frac{k-1}{2} \rceil}$,$C_{k-1}^{\lceil \frac{k-1}{2} \rceil+1}$,...,$C_{k-1}^{k-1}$，序列单调递减；
3. 序列左右对称。

根据序列规律，由目标数r，令k'=k-1，设计贪心线性搜索算法求最接近目标值r的连续子序列和：

算法：贪心线性搜索子序列
输入：目标数r，因子数k'，数组$c[k']$，其中$c[i]$=$C_{k'}^{i-1}$

输出：数组左端点下标$left$，右端点下标$right$，最接近目标数r的$total$

1. 初始化 $left$，$right$，$total$

2. $tempLeft \leftarrow 1, tempRight \leftarrow 1, tempTotal \leftarrow c[1]$ // 数组下标从1开始

3. $while\ j \leq k^*\ do$

4. $\quad if\ tempTotal < r$

5. $\quad then\ tempRight \leftarrow tempRight + 1$

6. $\quad\quad tempTotal \leftarrow tempTotal + c[tempRight]$

7. $\quad else\ if\ tempTotal > r$

8. $\quad then\ tempTotal \leftarrow tempTotal - c[tempLeft]$

9. $\quad\quad tempLeft \leftarrow tempLeft + 1$

10. $\quad else$

11. $\quad\quad [left, right, total] \leftarrow [tempLeft, tempRight, tempTotal]$

12. $\quad\quad break$

13. $\quad if\ tempTotal$比$total$更接近r

14. $\quad then\ [left, right, total] \leftarrow [tempLeft, tempRight, tempTotal]$

15. $return\ left, right, total$

其中最接近目标数的$total$既可以是$\leq r$但尽可能接近r的下界值，也可以是$\geq r$但尽可能接近r的上界值。如果r并非刚好为连续子序列和，后续算法需要对界限值与r值之差进行增加或减少的调整。

该算法的正确性可简易证明如下：

证明：考虑每一步决策情况。

对连续子序列的端点调整，有4种决策：左端点左移、左端点右移、右端点左移、右端点右移；

每步决策可能面临三种情况：当前值$< r$，当前值$> r$，当前值$= r$。

若当前值$= r$，说明已找到最优连续子序列组合，算法结束；

若当前值$< r$，有4种决策：

若左端点左移，即把左侧元素选入区间。因为算法的每一步决策步只有在子序列和超过目标值时才会选择丢弃左端元素，因此以左侧元素为端点的所有可行解必定已被遍历，是无意义的重复遍历；

若左端点右移，即丢弃当前左端点，所生成的新子序列必然比原子序列更差；

若右端点左移，同理新子序列必然更差；

若右端点右移，即选取右侧端点，所形成的新子序列可能更优，可能非法，比较新子序列和维护的最优解数据并更新最优解。因此右端点右移可保证必定遍历到全局最优值。

若当前值$> r$，讨论类似，从略。

由上，通过一次线性遍历，算法区间$[left, right]$，即求得一组组合数连续子序列$C_{k'}^{left}, C_{k'}^{left+1}, \dots, C_{k'}^{right}$，即选定$M_{k,left}, M_{k,left+1}, \dots, M_{right+1}$连续子矩阵序列所构成的矩阵。

基于组合规律的子矩阵编码递推

对于具体的子矩阵，将每列理解为1个物品，可以利用组合规律进行递推：
观察$M_{k,m-1}$，可以表示k个物品中选取m-1个的全部组合情况，$M_{k-1,m}$表示k-1个物品中选取m个的全部组合情况。则对于$M_{k,m}$子矩阵，它的第k列有选取和不选取两种情况。若选取，则相当于$[M_{k,m-1}, \mathbf{1}]$矩阵；若不选取，则相当于$[M_{k-1,m}, \mathbf{0}]$矩阵。综上，可得递推式：

$$M_{k,m} = \begin{cases} \begin{pmatrix} M_{k-1,m} & \mathbf{0} \\ M_{k-1,m-1} & \mathbf{1} \end{pmatrix}, & k > m > 0 \\ \mathbf{1}, & k = m \\ \mathbf{0}, & m = 0 \end{cases}$$

其中$\mathbf{0}, \mathbf{1}$均为对应维度的列向量。根据递推式，子矩阵可以高效地求出。进一步分析，矩阵递推可通过进制编码转换为十进制递推。把每一个0-1向量视为一个二进制串，可一一映射地编码为对应的十进制数。而对于矩阵扩充的递推情况，在原行向量右侧增加一位，相当于原十进制数$\times 2$后再$+0$或1。综上，可得十进制编码递推式：

$$key_{k,m} = \begin{cases} \begin{pmatrix} key_{k-1,m} \times 2 \\ key_{k-1,m-1} \times 2 + 1 \end{pmatrix}, & k > m > 0 \\ \mathbf{1}, & k = m \\ \mathbf{0}, & m = 0 \end{cases}$$

同样地，$\mathbf{0}, \mathbf{1}$均为对应维度的列向量。通过对子矩阵的编码递推，可以高效而简洁地完成递推求解。在求得对应子矩阵的压缩列向量后，只需将十进制编码还原为二进制0-1串即可得到原子矩阵。

贪心重排序回溯算法调整解矩阵

综上所述，当采样目标值r恰好为某连续子序列和时，可以通过灵活的递推和编码实现极快的求解。若r不刚好为某连续子序列和，则不得不使用其他算法手段对最终解矩阵进行调整。**显然**，在原有连续子序列的基础上，只需对序列的左侧或右侧进行调整，视乎求得的连续子序列取$\geq r$的上界和或$\leq r$下界和，对序列侧进行裁剪或增加处理。由于靠近序列中央的采样集簇程度更高，应尽可能裁剪更靠近一端的行向量而尽可能增加靠近中央的行向量。
具体到生成某一调整向量时，为保证最终生成的子矩阵对各列的ee是均匀的，每次生成调整向量时都尽可能贪婪地令已有ee数较少的列形成新的ee。由此，可构造微调残缺子矩阵M'的求解算法。

此处的M'含义视乎上界/下界子序列策略而相应地成为矩阵裁剪部分或矩阵增加部分。把每一列的0-1选取（或是否产生ee）视为一决策步，当求得一组0-1组成的行向量时，验证是否曾经选取，若不曾选取则加入到M'中。每次求得一可行向量，都需要更新各因子ee计数并重新从小到达排序，然后从头开始回溯，通过这样的方法保证ee对各因子抽样的均匀性。可抽象算法如下：

子算法：$ReBack(step)$
输入：$step$，当前决策步
输出：$finished$，是否取得一可行解
1. if 当前分枝继续求解已不可能满足m'的取值条件　　// 剪枝
2. $then\ return$
3. if 当前$vector$符合k,m要求　　// 回溯解
4. $then\ if\ HashFind(vector) = true$　　// 哈希表中已有该行向量
9. 　　$then\ return\ finished \leftarrow false$
7. 　　$HashInsert(vector)$
7. 　　将$vector$加入到M'中
7. 　　更新$factors$
7. 　　对$factors$按形成的ee数从小到大排序
8. 　　$return\ finished \leftarrow true$
11. $vector[factors[step]] \leftarrow true$
12. $finished \leftarrow Reback(step + 1)$
13. $if\ finished = true$
13. $then\ return\ finished$
11. $vector[factors[step]] \leftarrow false$
12. $finished \leftarrow Reback(step + 1)$
13. $return\ finished$

算法：生成残缺子矩阵
输入：调整目标数r'，因子数k，每行ee数m'
输出：残缺子矩阵M'
1. 初始化$factors[]$，记录各因子编号及对应已生成的ee数
10. $while$ 各列还没有全部满足r'条件
11. 　　$ReBack(1)$

此外，观察子矩阵规律，对于位于序列左侧的$M_{k,left-1}$。因为$M_{k,left-1}$每行选取的“物品”数比$M_{k,left}$少1，因此对于$M_{k,left-1}$的每1行，将其0元素所处的位置替换成1，即为与其形成ee的

$M_{k,left}$上的行。即$M_{k,left-1}$可以与$M_{k,left}$在元素为0的列上形成ee。据此特征，可以很容易将0-1选取情况预处理成ee选取情况，并将对应参数传入上述的回溯求解框架中。

时间复杂度分析

以下对算法各步骤进行分析：

线性贪心搜索最接近子序列最坏情况下$left$移动k-1次，$right$移动k-1次，由于组合数子序列具有对称性，因此即使是最坏情况，左右端点移动次数各不会超过$\frac{k-1}{2}$次，因此最坏时间复杂度$=O(k)$；

对于子矩阵递推求解，可以自杨辉三角顶部开始自顶向底递推。最坏情况下全部k个子矩阵全部选取，即需求解$\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$次，因此最坏时间复杂度$= O(k^2)$；

对于贪心回溯求解子矩阵，每次有k个决策步，每次排序时间复杂度为$O(klogk)$，每轮回溯时间复杂度$= O(2^k)$，最坏情况下对共k个因子回溯$\frac{r'k}{m}$轮，总回溯时间复杂度$= O(\frac{r'k2^k}{m})$，总排序时间复杂度为$\frac{r'k^2logk}{m}$。

根据以上分析，贪心搜索与矩阵递推的时间复杂度都取决于k值，而k与目标ee数r相比通常十分小，配合向量进制压缩编码，**求解效率极高**，实现了算法性能的巨大提升。然而，若选取的r值并非恰好为连续子序列时，贪心回溯求解子矩阵算法具有非多项式的时间复杂度，k值只要稍微增加，便可能面临“指数爆炸”的风险，非常不建议指定“不规则”的r值目标。若令r为某一连续子序列和之值，可以以不可思议的速度实现对Morris方法的簇采样矩阵的极速生成，为后续的科学计算提供有力的支持。

结论

本文通过对Morris方法簇采样矩阵的算法重新设计和求解，并对该算法的采样效率和时间复杂度进行了初步分析，证明了该算法能以相当高效的求解速度求得采样效率较高的簇采样矩阵，为科学计算提供上游算法效率的支持。此外，对如何在可接受时间复杂度内求解最优采样效率的簇采样矩阵仍待进一步探索。