Trace Generator

Input

n: the total number of tasks in a trace

m: the number of task A

relatedness: the ratio of the number of pairs with occurrence patterns to the total number of pairs which can be calculated using permutation as n*(n-1). For example for n=4, 4P2=4*3=12.

Output: The **trace** that fulfills the conditions of the input.

Other definitions:

r: the number of tasks with occurrence patterns in task A-transactionalized trace.

f: the number of pairs with occurrence patterns, then relatedness can be written as $\frac{f}{n*(n-1)}$.

Example 1:

For a simple trace with all the tasks alphabetically arranged:

A B C C D

A B C C

A B C C D

n=4 m=3

The total number of pairs of different tasks is 12—(i.e. <A, B>, <A, C>, …)

The number of tasks with occurrence patterns in task A-transactionalized trace r is 2 which are task B and task C, so r=2.

The number of pairs with occurrence patterns f is 6—<A B>; <B A>; <A C>; <C A>; <B C>; <C B> which is the permutation of ${}_{(r+1)}P_2$ or r*(r+1). As a side note, the reason why it is r+1 is that previously we did not count task A in the value of r while we need to account for it here.

Relatedness: 6/12=0.5

Situation 1:

Given the inputs n, m, and relatedness, the most essential step is to generate a trace with "n*(n-1)*relatedness" occurrence patterns.

If this number n*(n-1)*relatedness can be written as the multiplication of two consecutive integers, then we can build the relationship: n*(n-1)*relatedness=r*(r+1) where we can get the number of tasks with occurrence patterns in task A-transactionalized trace "r". For the rest (n-r-1) tasks that do not have occurrence pattern in task A-transactionalized trace, we just generate them a random number of times in each row such that there are no resulting occurrence patterns.

Example 2:

Given that n=4, m=3, and relatedness=0.5, according to n*(n-1)*relatedness=r*(r+1), we know that r =

2, thus task B and C have occurrence patterns in A-transactionalized trace, and task D has no pattern so we just generate it randomly. The result could be:

A B C C D D D
A B C C D D
A B C C D

Situation 2:

If the value of "n*(n-1)*relatedness" cannot be written as the multiplication of two consecutive integers, then we need to find other ways to generate the trace.

Firstly, we find the largest r that satisfies the inequality: =r*(r+1) < n*(n-1)*relatedness, then we could generate r*(r+1) pairs with occurrence patterns.

Subsequently, there should be "n*(n-1)*relatedness - r*(r+1)" more pairs with occurrence patterns.
Define a new parameter k where k = n*(n-1)*relatedness - r*(r-1), then we discuss how to add k pairs with occurrence patterns from two scenarios.

Scenario 1:
If k is an even number and suppose n=6 (task A—task F), m=4, relatedness=0.8, then we need to generate 0.8*6*5=24 pairs with occurrence patterns. For the largest r that makes r*(r+1) <24, we get r= 4, which means there are 4 tasks (task B C D and E) that have occurrence patterns in task A. For task F, just generate it randomly in each row. Now we have the basic trace whose number of pairs with occurrence patterns is "5*4"=20 as follows:

Example 3:
A B C C D D E F F
A B C C D D E F F F
A B C C D D E
A B C C D D E F

However, there are "24-20"=4 pairs with occurrence patterns still left. Now we try to add 4 more pairs with occurrence pattern:

Step 1: For each row, remove the first task without occurrence pattern in task A-transactionalized trace which can be calculated as task "r+2". In this case it is task F.
A B C C D D E
A B C C D D E
A B C C D D E
A B C C D D E

Step 2: Insert one task "r+2" right after the task "k/2" which is task B in this example except the last row. For the last row, insert task "r+2" at the end of the task.

A B **F** C C D D E
A B **F** C C D D E
A B **F** C C D D E
A B C C D D E **F**

Then task F will have an occurrence pattern in A-transactionalized trace and also B-transactionalized trace. Inversely, task A also has an occurrence pattern in F-transactionalized trace and so does task B.

The reason why the last task F is inserted at the end of the trace is to ensure that the rest of the task such as task C, D and E doesn't have pattern in F-transactionalized trace. In detail, task F in the last row has always been there to ruin the pattern of task C as 2 2 2 4, or else task C will also have an occurrence pattern in F-transactionalized trace as 2 2 2 2 and the same for task D and task E.

Scenario 2:

For the last situation—k is an odd number. Suppose n=6, m=4, relatedness=0.5 and then we need to generate 0.5*6*5=15 pairs with occurrence patterns. For the largest r that makes r*(r+1) <15, we get r= 3, which means there are 3 tasks (task B C and D) that have occurrence patterns in task A. We can now generate the trace whose number of pairs with occurrence patterns is "4*3"=12 as follows:

A B C C D D E F F
A B C C D D E E F
A B C C D D F F F F
A B C C D D E F F

Step 1: For each row, remove the first task without occurrence pattern in task A-transactionalized trace which can be calculated as task "r+2". In this case it is task E.

A B C C D D F F
A B C C D D F
A B C C D D F F F F
A B C C D D F F

Step 2: Insert one task "r+2" in each row after the task "(k+1)/2" which is task B in this example except the last row. In the last row, insert at the end of the task.

A B **E** C C D D F F
A B **E** C C D D F
A B **E** C C D D F F F F
A B C C D D F F **E**

Then task E will have an occurrence pattern in A-transactionalized trace and also B-transactionalized trace and vice versa which are <A E> <E A> <B E> <E B>. Similarly as the previous scenario, since there is a task E at the end of the trace, task C, D and E will have no patterns in E-transactionalized trace.

Step 3: Except for the first row, insert one task "r+2" after each task A.

A B **E** C C D D F F

A **E** B **E** C C D D F

A **E** B **E** C C D D F F F F

A **E** B C C D D F F **E**

Therefore, the pattern <E A> will be removed because the pattern that task E shows in each row change to 1, 2, 2, 2 which is of no pattern while the other pairs with occurrence patterns are not lost. However, some of the pattern has changed. Like in E-transactionalized trace, the pattern of task A used to be 1, 1, 1 and now it is 1, 0, 1, 0, 1, 0. So does task B in E-transactionalized trace.