Online-Appendix

A. Run time effects

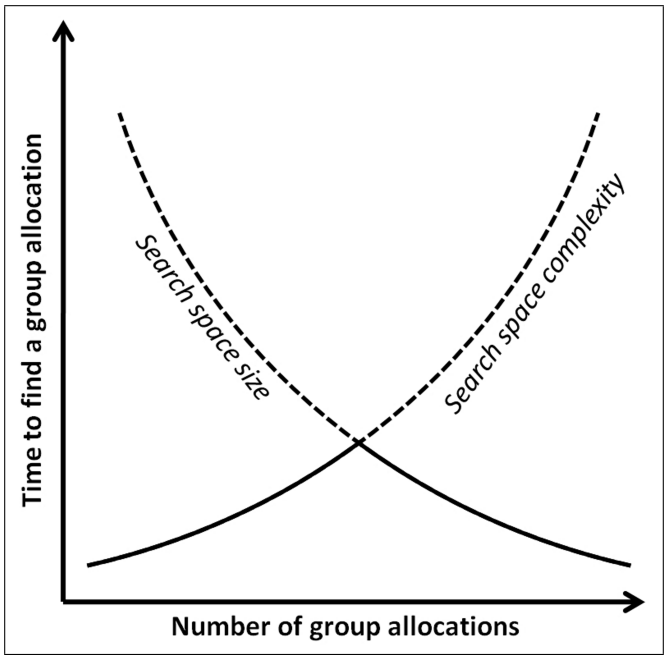

Figure A.1: The graph schematically illustrates two contrary effects which influence the run time for finding subsequent group allocations. Initially, a low run time is achieved when searching for a feasible group allocation since the search space is not constrained through matching histories. The longer the sequence of group allocations under the perfect stranger criterion, the more constraints have to be satisfied for further group allocations. While this increases the complexity of the search space, it simultaneously shrinks the size of the search space since many solutions can be excluded a-priori. Overall, this leads to an increase in run time for subsequent group allocations up to a certain point after which the run time decreases again.

B. Run time table

	Proposed Algorithm					**Brute-Force**				
N	G=2	G=3	G=4	G=5	G=6	G=2	G=3	G=4	G=5	G=6
4	~0ms	-	-	-	-	~0ms	-	-	-	-
5	-	-	-	-	-	-	-	-	-	-
6	~0ms	-	-	-	-	~0ms	-	-	-	-
7	-	-	-	-	-	-	-	-	-	-
8	~0ms	-	-	-	-	~0ms	-	-	-	-
9	-	~0ms	-	-	-	-	0,3ms	-	-	-
10	~0ms	-	-	-	-	3,1ms	-	-	-	-
11	-	-	-	-	-	-	-	-	-	-
12	0,1ms	~0ms	-	-	-	23ms	4,3ms	-	-	-
13	-	-	-	-	-	-	-	-	-	-
14	0,1ms	-	-	-	-	30ms	-	-	-	-
15	-	0,1ms	-	-	-	-	70ms	-	-	-
16	0,2ms	-	~0ms	-	-	61ms	-	~0ms	-	-
17	-	-	-	-	-	-	-	-	-	-
18	0,2ms	0,1ms	-	-	-	6,4s	1,2s	-	-	-
19	-	-	-	-	-	-	-	-	-	-
20	0,3ms	-	0,4ms	-	-	88s	-	0,7s	-	-
21	-	0,2ms	-	-	-	-	0,6s	-	-	-
22	0,4ms	-	-	-	-	1096s	-	-	-	-
23	-	-	-	-	-	-	-	-	-	-
24	0,5ms	0,5ms	0,9ms	-	-	x	5,1s	3,8s	-	-
25	-	-	-	0,1ms	-	-	-	-	0,3s	-
26	0,7ms	-	-	-	-	x	-	-	-	-
27	-	1ms	-	-	-	-	54s	-	-	-
28	0,9ms	-	2,6ms	-	-	x	-	3,7s	-	-
29	-	-	-	-	-	-	-	-	-	-
30	1,2ms	2ms	-	1,1s	-	x	x	-	x	-
31	-	-	-	-	-	-	-	-	-	-
32	1,8ms	-	8,6ms	-	-	x	-	0,5s	-	-
33	-	5,5ms	-	-	-	-	x	-	-	-
34	2,3ms	-	-	-	-	x	-	-	-	-
35	-	-	-	1,2s	-	-	-	-	x	-
36	3ms	8,8ms	25ms	-	0,2ms	x	x	102s	-	215s
37	-	-	-	-	-	-	-	-	-	-
38	4,3ms	-	-	-	-	x	-	-	-	-
39	-	19ms	-	-	-	-	x	-	-	-
40	7,2ms	-	72ms	1,7s	-	x	-	x	x	-

Table B.1: The Table shows the average computation time needed to generate a single complete sequence for specific configurations (n,g). Run time tests were conducted on an i5 2500k without parallel computing (only a single core was used). For large problem instances, it was not possible to measure the brute-force run time due to its substantial increase in computation time.

C. Run time comparison

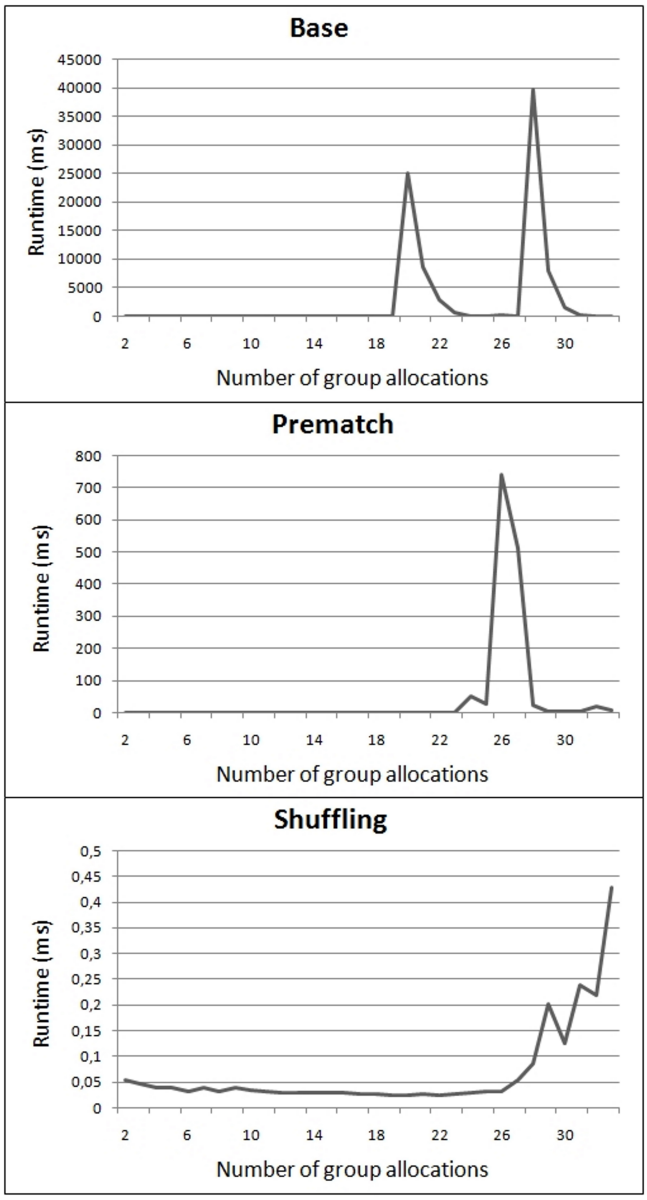

Figure C.2: For this figure, group allocations for the configuration (n,g) = (36,2) have been computed. Thereby, the time needed to find the next group allocation within a sequence was measured. Run time measurements where conducted on an i5 2500k. The top graph displays the computation time for the proposed algorithm without random shuffling. For the graph in the middle, the first 17 group allocations have been generated with a systematic pattern before applying the proposed algorithm. The bottom graph displays results of the proposed algorithm, combined with random shuffling.

D. Pseudocode

Algorithm 1 This algorithm generates a complete sequence under the perfect stranger criterion. Input parameters are the number of participants *n* and the group size g. Each of the *n* participants has a listing of participants, which they did not meet in previous group allocations, that is updated after each successful group allocation. Executing the function *AllocationSequence* returns a complete PSM sequence.

```
//possiblePartners[i,] symbolizes participants which are unknown to i
possiblePartners ← n × n matrix of ones with zeros on its diagonal

function ALLOCATIONSEQUENCE (n, g)
    sequence ← list of group allocations
    participantList ← list of n participants

    //Searching for a complete sequence
    while True do
        //Find new group allocation
        groupList ← empty list of groups
        groupList ← FINDALLOCATION (participantList, groupList,n, g)

        //allocate groups, if a match has been found
        if groupList != Null then
            add groupList to sequence
            ∀r, c ∈ [1, n]: possiblePartners[r, c] = 0 if c and r are grouped
            //optionally, shuffling of all lists can be inserted here
        else
            Break
        end if
    end while

    return sequence
end function
```

```
function FINDALLOCATION (unusedElem, groupList, n, g)
    if number of groups in groupList == (n/g) then
        return groupList
    end if
    memory ← empty group
    pivotElement ← first element of unusedElem
    posMatches ← unusedElem ∩ (participants at possiblePartners[pivotElement, ])
    if size of posMatches >= (g-1) then
        return Null
    end if

    //iterate over possible groups to add to groupList
    while True do
        //find group for given pivotElement
        curGroup ← new group with only pivotElement included
        curGroup ← FINDGROUP (posMatches, curGroup,g,memory)

        //stop if no group building was possible, else next recursion
        if curGroup == Null then
            return Null
        else
            add curGroup to groupList
            newUnusedElem ← unusedElem \ curGroup
            newGroupList ← FINDALLOCATION (newUnusedElem, groupList, n, g)
        end if

        //Store latest group in memory when group allocation failed
        if newGroupList == Null then
            memory ← curGroup
            remove curGroup from groupList
        else
            return newGroupList
        end if
    end while

    return Null
end function
```

```
function FINDGROUP (availElem, group, g, memory)
    size ← number of elements in group

    //end condition
    if memory is not empty then
        recursively build the group from memory
        clear memory and skip the rebuilt group in recursion
    end if

    if g == size then
        return group
    end if

    if number of elements in availElem < (g-size) then
        return Null
    end if

    //fetch new partner for group
    while number of elements in availElem > 0 do
        newPartner ← first element of availElem
        add newPartner to group
        posPartners ← participants at possiblePartners[newPartner, ]
        newAvailElem ← availElem ∩ posPartners
        newGroup ← FINDGROUP (newAvailElem,group,g,memory)

        if newGroup == Null then
            remove newPartner from availElem
            remove newPartner from group
        else
            return newGroup
        end if
    end while

    return Null
end function
```