Algorithms for generating design sequences

The algorithm described in Appendix 1 of Nonyane and Theobald (2007) is intended for finding the complete set of design sequences for a specified number of symbols n with the properties A and B defined in Section 2 of that paper: it has been used to find these sets for n equal to 6 and 7. We have not run it to completion for larger values of n: instead we generate sequences chosen by a random process from the complete set, although this process is not guaranteed to give every valid sequence the same probability of selection. In the first of the following algorithms, the initial sequence comprises symbols which are chosen at random.

Thus there are two versions of the program: one (the 'random' version) generates a new random initial sequence whenever a valid sequence is found, and the other (the 'systematic' version) continues until *all* valid sequences are found, if run for a sufficiently long period. In both programs, it is convenient to use symbols numbered from 0 to $n-1$ rather than from 1 to n, and to increase each symbol by 1 before displaying the sequences.

Random algorithm

We follow the rules listed below in order, except where a 'go to' or 'stop' instruction is given. We use `j` to index a position in the sequence (`j` = `0`, `1`, ..., $\texttt{n}^2$), and `s(j)` to denote the symbol (`0`, `1`, ... or `n-1`) at that position. The sequence is regarded as comprising an initial symbol `0`, followed by `n` 'blocks' each of length `n`, the first of which is (`0`, `1`, ..., `n-1`). Runs of the program use different random initial sequences (except by chance), so a re-run of the program will not give the same set of valid sequences in the same order.

When the program starts:

Get from the user the number of symbols (`n`) and the time (in seconds) that the program is allowed to run (`maxtime`).

Set to 0 the number of valid sequences found so far (`count`) and the number of them with the lowest values of criteria (4) and (5) of Nonyane and Theobald (2007) (`nfound`).

Set the lowest criterion values found so far (`mincrit4` and `mincrit5`) to huge values.

Set the start of the current block (`bmark`) to `n+1`.

Create an array `found()` of size `2n` $\times$`(`$\texttt{n}^2$`+1)` to store the best valid sequences that have been found. The number of these is assumed to be no more than `2n`.

Now do the search:

RES. Generate a random initial sequence `a()` of $\texttt{n}^2$`+1` symbols as

`0`, `0`, `1`, ..., `n-1`, `n-1`, `a(n+2)`, `a(n+3)`, ..., `a(`$\texttt{n}^2$`)`

in which `a(n+2)`, `a(n+3)`, ..., `a(`$\texttt{n}^2$`)` are selected with probabilities $\texttt{n}^{-1}$ from the symbols `0`, `1`, ..., `n-1` independently of each other and of previous initial sequences. Copy `a()` to the working array `s()`.

Start at position `j = n+2`, i.e. the second position in the second block.

PT1. If `s(j)` is equal to `s(j-1)+1`, or `s(j)` already appears in the current block, or the pair (`s(j-1)`, `s(j)`) already appears in the sequence, go to step PT2. [The first check is applied because all pairs of the form (`s(j)`, `s(j)+1`) are in the first block.]

If we have not reached the end of a block (i.e. if `j modulo n` $>$ `0`), increment `j`, set `s(j)` = `a(j)` and go to step PT1.

If we have reached the end of the last block (i.e. `j` = $\texttt{n}^2$), we have found a valid sequence. Go to step FOU.

We have reached the end of a block that is not the last block. If `s(j)` does not appear at the end of any earlier block, set `s(j+1)` = `s(j)`, increment `j`, mark `j` as the start of the current block, and go to step PT1.

PT2. Replace `s(j)` by {`s(j)+1`} `modulo n`. If `s(j)` $\neq$ `a(j)` then go to step PT1.

`s(j)` has now gone back to its starting value `a(j)`. So we need to move back a place and start again. Decrease `j` by `1`.

If `j` = `n+1`, we are at the beginning of the second block, so we have not found any possible valid sequences: go to step FIN.

If `j modulo n` = `1`, we are at the beginning of a block, so set `j-n` as the start of the current block, set `s(j-1) = a(j-1)`, and decrease `j` by `2`.

Go to step PT2.

FOU. We have found a valid sequence. Increase the number of such sequences that have been found (`count`). Calculate the criterion values (`crit4` and `crit5`).

If the new sequence is better than any that have been found so far (i.e. either (`crit5` $<$ `mincrit5`) or (`crit5` = `mincrit5` and `crit4` $<$ `mincrit4`)), store the sequence in `found(0,*)`, set `nfound` = 1, and go to step TIM.

If the new sequence is as good as the best found so far (i.e. `crit5` = `mincrit5` and `crit4` = `mincrit4`) then see if the sequence has been stored in `found()`: if it has not, store it and increment `nfound`.

TIM. If there is time left to generate more sequences, go to step RES.

FIN. Increase each symbol by 1 in the `found()` array. Display in the output the best criterion values, the contents of the `found()` array, and the number of valid sequences that have been found.

Stop.

Systematic algorithm

The program for systematic searches works in a similar way, except that `a(n+2)`, `a(n+3)`, ..., `a(`$\texttt{n}^2$`)` are replaced by 0. When a valid sequence is found the program displays it with its criterion values, and continues searching without creating a new initial sequence. Unless a time restriction is imposed, the program runs until all valid sequences have been found.