APPENDICES

1. PSEUDO-CODE USED TO GENERATE THE SENDERS AND RECEIVERS

In this appendix, we describe the code we used to generate the senders and receivers in our queues of synthetic payments. As mentioned in Section 2, we used the method in (Soramäki et Cook 2013). The pseudo-code is given in Algorithm A1 below. The logic of the algorithm is that senders and receivers are selected randomly with a probability based on their *preferential attachment* h. When generating a new payment order, the larger h_i, the higher the probability of i being selected as a sender or a receiver.

In a nutshell, the method starts by adding n_0 core participants to the system, each with a preferential attachment $h_i = 1$. Then, it adds iteratively $(n - n_0)$ periphery participants, where n is the total number of participants. Every time a periphery participant is added $m = integer(V_p/(n - n_0))$ payments are generated. Senders and receivers are selected using a probability $P_i = h_i / \sum_{p \in n} h_p$. Once the sender i^* and the receiver j^* have been selected, a quantity α is added to their strength of preferential attachment, h_{i^*} and h_{j^*}. This process reinforces their likelihood of being selected again for future payment orders. The steps described above are the steps in the original method from (Soramäki et Cook 2013). These steps are located from lines 1 to 15 in Algorithm A1.

As described above, m is the integral part of $V_p/(n - n_0)$. This means that most of the time decimals are truncated. Therefore, the method generates fewer payments than the target number V_p. The number of missing payments is equal to $\left(V_p - m * (n - n_0)\right)$. We add a second section to the method to generate these few missing payments using the same preferential attachment logic. That is, senders and receivers are selected using the probability $P_i = h_i / \sum_{i \in n} h_i$. Once the sender i^* and the receiver j^* have been selected, a quantity α is added to their strength of preferential attachment, h_{i^*} and h_{j^*}. This second section is located from lines 16 to 22 in Algorithm A1.

Algorithm A1

Input: n_0, n, V_p, α
Output: *Queue (a queue of synthetic payments)*

1 **Initialize** $h[i] = 0$ for $i = 0, \dots, n-1$
2 **Set** $h[i] = 1$ for $i = 0, \dots, n_0 - 1$
3 **Set** $Queue = empty\ list$
4 **Set** $m = integer(V_p \,/\, (n - n_0))$
5 **for** k in range (n_0, $n-1$, 1) **do**
6 **for** l in range (0, $m-1$, 1) **do**
7 Randomly choose a sender i^* using probability based on h
8 // i.e. participant i has a probability $h[i]/\sum_p h[p]$ of being chosen
9 **Set** $h[i^*] = h[i^*] + \alpha$
10 Randomly choose a receiver $j^* \neq i^*$ using probability based on h
11 **Set** $h[j^*] = h[j^*] + \alpha$
12 **Append** payment (i^*, j^*) to $Queue$
13 **end**
14 **Set** $h[k] = 1$ // add a new periphery participant
15 **end**
16 **for** l in range (0, $(V_p - m * (n - n_0)$, 1) **do**
17 Randomly choose a sender i^* using probability based on h
18 **Set** $h[i^*] = h[i^*] + \alpha$
19 Randomly choose a receiver $j^* \neq i^*$ using probability based on h
20 **Set** $h[j^*] = h[j^*] + \alpha$
21 **Append** payment (i^*, j^*) to $Queue$
22 **end**

Algorithm A1: Pseudo-code of the method used to generate queues of synthetic payments. The section from lines 1 to 15 is the original method from (Soramäki et Cook 2013)*. The section from lines 16 to 22 was added to be able to match a particular number of payments using the same preferential attachment logic.*

2. TRUNCATED LOG-NORMAL DISTRIBUTION SAMPLING

In this appendix, we show the difference between (1) the target mean μ_v, the target standard deviation σ_v and the target maximum payment value max_v used to define the payment value distribution and (2) the realised mean μ_v^*, the realised standard deviation σ_v^* and the realised maximum payment value max_v^* when sampling from a truncated log-normal distribution. To compare target and realised values, we sample 100,000 values from a truncated log-normal distribution with parameters (μ_v, σ_v, max_v). Then, we compute (μ_v^*, σ_v^*, max_v^*) from the sample. All values can be found in Table A2.

We can see that the statistics of the sampled data match well the target statistics. The difference between μ_v and μ_v^* is less than 1% for all values of σ_v. The difference between σ_v and σ_v^* is less than 2% for all values of σ_v. Finally, the realised maximum max_v^* is always smaller than max_v.

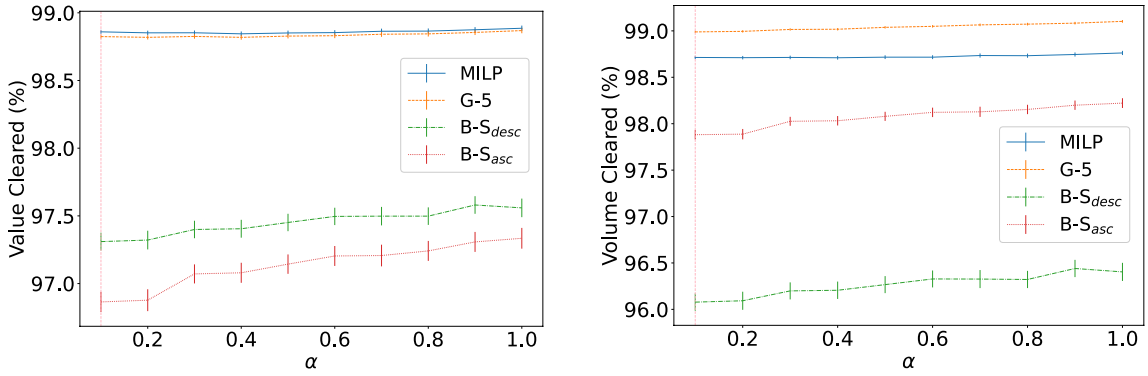
σ_v	μ_v^*	σ_v^*	max_v^*	Error on the mean (%)	Error on the std (%)
0.05	1,937,127.46	0.05	2,373,619.50	**-0.00**	**0.05**
0.10	1,937,436.24	0.10	2,920,497.62	**0.02**	**-0.50**
0.20	1,933,736.40	0.20	4,129,834.47	**-0.18**	**-0.59**
0.30	1,936,961.48	0.30	6,990,150.51	**-0.01**	**-0.10**
0.40	1,944,428.28	0.40	9,735,584.29	**0.38**	**1.28**
0.50	1,932,845.32	0.50	12,941,281.18	**-0.22**	**-0.05**
0.60	1,939,360.60	0.60	17,285,879.75	**0.12**	**-0.13**
0.70	1,933,540.48	0.69	19,298,776.59	**-0.19**	**-1.10**
0.80	1,931,878.74	0.79	19,158,582.56	**-0.27**	**-1.60**

Table A1: Difference between target statistics and realised statistics. The target mean μ_v was set at £1,937,131.63 for all scenarios (more on how this value was determined can be found in Section 2). Different scenarios had different relative standard deviations σ_v. Each σ_v used during the simulations is a row in the table. The maximum payment value allowed was set at $10 * 1{,}937{,}131.63 = £19{,}371{,}316.30$.

3. SIMULATION RESULTS: VARYING THE STRENGTH OF PREFERENTIAL ATTACHMENT

In this appendix, we provide the results for the fourth set of simulations (where the parameter of interest is the *strength of preferential attachment*) as described in Section 4. For these simulations, we consider 10 scenarios where α varies between 0.1 and 1. All other parameters are assigned their default values (listed in Table 1). These scenarios investigate the effect of the heterogeneity in participant sizes on the performances of the algorithms. The statistics were computed over 1,000 independent queues.

Figure A1(a) shows the evolution of the average value cleared (as a percentage of the total value in the queue) and Figure A1(b) shows the average volume cleared (as a percentage of the total number of payments in the queue). All averages have an error bar representing their 99% confidence intervals. We can see that α has a relatively marginal impact on the performances of both the G-5 and the MILP algorithms as their curves remain approximately flat between 98.8% and 98.9% of value cleared, whereas the B-S algorithms tend to slightly improve as the participants become more heterogeneous in size. Additionally, the ranking of the algorithms remains the same for all values of α (e.g., for the value cleared, MILP $\geq$ G-5 $>$ B-S_{desc} $>$ B-S_{asc}). It is also worth noting that α does not have any impact on the computing time of the algorithms (MILP remains around 1 second for all values of α and the other algorithms remain below 0.2 seconds).



(a) *(b)*

Figure A1: (a) Evolution of the average value cleared (in %) as α increases. (b) Evolution of the average volume cleared (in %) as α increases. Error bars represent the 99% confidence intervals of the averages. The pink vertical line in each figure shows the value used in the default scenario given in Table 1.

4. UNFEASIBLE SOLUTIONS

In this appendix, we list the scenarios in which MILP solutions became unfeasible as a result of rounding the algorithm output (as explained in the last paragraph of Section 4). We can see that instances when solutions become unfeasible are rare. There does not seem to be any obvious relationship between the scenarios and the number of unfeasible solutions. Capacity = 10% is the scenario where we counted the most unfeasible solutions (9). However, this number remains small compared to the total number of queues simulated for that scenario (0.9%).

Scenario	**# unfeasible solutions**	**# simulated queues**
$\alpha = 0.8$	1	1,000
Capacity = 10%	9	1,000
Capacity = 40%	1	1,000
Capacity = 50%	1	1,000
Capacity = 60%	1	1,000
Capacity = 80%	1	1,000
$V_p = 5{,}000$	2	1,000
$V_p = 7{,}000$	1	1,000
$\sigma = 5\%$	3	1,000
$\sigma = 10\%$	1	1,000
$\sigma = 30\%$	2	1,000
$\sigma = 40\%$	1	1,000
$\sigma = 60\%$	1	1,000
$\sigma = 70\%$	1	1,000

Table A2: Scenarios where unfeasible solutions happened. Example read (first row): in the simulation where α was the varying parameter, when $\alpha = 0.8$, one MILP solution became unfeasible after rounding over the 1,000 queues that were simulated.

5. TWO-OBJECTIVES PERFORMANCES

In section 5.4, we discussed the results of the Value-Delay optimisation task where payment delays were distributed following a truncated log-normal distribution. In this appendix, we show the results of the same task with payment delays being distributed uniformly, as described in Section 3. Results are given in Table A3.

We can see that the results are similar in both the uniform and truncated log-normal case. MILP-VD still has a much lower sum of delays outstanding than the B-S alternative (10,714.045 ± 268.732 seconds (178 minutes) vs. 15,962.380 ± 447.629 seconds (266 minutes), *that is a 31% decrease in delays outstanding*). However, the difference is smaller than in the truncated log-normal case. MILP-VD also has a higher value and volume cleared than B-S. So, it seems to remain a better alternative on average.

With a uniform distribution, MILP however accumulated a much higher sum of delays (which is normal as it does not take delays into account when optimising). Comparing MILP and MILP-VD, we can see that MILP-VD has 59,491 seconds (991 minutes) less of sum of delays outstanding (-84.61%). This large improvement in delays outstanding came at the cost of clearing £24,992,575.52 (-0.43%) less. This corresponds to a cost of *£25,219.55* per saved minute of delay outstanding.

	B-S	MILP	MILP-VD
volume cleared (%)	97.081 ± 0.067	98.699 ± 0.017	98.460 ± 0.019
volume cleared	2,912.435 ± 1.986	2,960.983 ± 0.498	2953.807 ± 0.540
value cleared (%)	97.070 ± 0.066	98.841 ± 0.015	98.411 ± 0.019
value cleared (£)	5,641,582,409.33 ± 4,224,032.28	5,744,520,354.35 ± 1,898,664.48	5,719,527,778.83 ± 1,993,435.45
delay remaining (s)	15,962.380 ± 447.629	70,205.350 ± 1,031.345	10,714.045 ± 268.732
delay remaining (%)	0.295 ± 0.009	1.300 ± 0.020	0.198 ± 0.006
computing time (s)	0.089 ± 0.0005	0.991 ± 0.0036	0.879 ± 0.005

Table A3: Comparison of performances between the B-S algorithm sorted by descending delays, the MILP and MILP-VD algorithms. Delays are distributed uniformly. The values shown are averages +/- 99% confidence intervals computed over 1,000 queues of synthetic payments. Example read (cell in row 1, column 1): the B-S algorithm was able to clear 97.081% +/- 0.067% of payments on average.