Optimising the LINPACK 1000 using Parallelization

Sam Serrels
40082367@napier.ac.uk
Edinburgh Napier University
Concurrent and Parallel Systems (SET10108)

1 Introduction

The LINPACK benchmark Finding the optimal route for a delivery truck, delivering items of varying size to customers spread over a large area, is a complex and computationally intensive task. The problem is similar to the travelling salesman problem, where the task is to find the shortest route to visit every location within a set only once.
The Delivery truck problem extends this with the additional factors that the delivery trucks have a capacity limit to how much they can carry, and thus how many of the customers can be served by a single truck.

Algoirthm analysis The capacity limitations means that multiple routes must be generated. If delivery time is a factor, then multiple trucks could be driving simultaneously, conversely a single truck could go round one route, resupply at the depot, then drive around another route. This report is not taking delivery time into into account, only the number of routes, and the combined cost of all the routes. The truck capacity is assumed to be the same for all trucks.

Related Work Generating the optimal path for multiple routes is no longer a simple optimisation task, multiple approaches can be taken and often the solution chosen is not mathematically perfect, but chosen as a trade-off between efficiency and computation time. Assigning one customer to one route means taking into account the effects this will have for the other routes. A brute force method could be taken to process all the possible route combinations, or a saving algorithm could be used.

High-Performance Linpack The savings algorithm implemented and tested in this report is the Clarke-Wright algorithm, first described in the 1962 paper "Scheduling of vehicles from a central depot to a number of delivery points" [Clarke and Wright 1962]. The algorithm finds a solution to the problem in a heuristic manor, so the result will not be the perfect solution, but should be relatively close to perfect with substantially less computation time than a brute force attempt.

Project Scope The Algorithm first starts by calculating the "savings" of delivery to a single pair of customers, rather than each customer on two trips. This is done for every combination of customers, resulting in a list or every customer pair and the corresponding savings. This list is sorted by the savings, in descending order.

OpenMP

SIMD

2 Linpack Gaussian elimination

Initial analysis The generated routes were exported as a .csv data file, and a visualisation of the routes were also generated and saved as a .svg vector image format. These images are included in the report and can bee seen in the results section.

The Pivot loop A basic solution in which a single truck was allocated to every customer was created, forming the most ineffective solution possible. This was used as the baseline set of data that the Clarke-Wright solutions were compared against. The solution cost and total number of routes for each algorithm were compared to each other and to the baseline numbers.

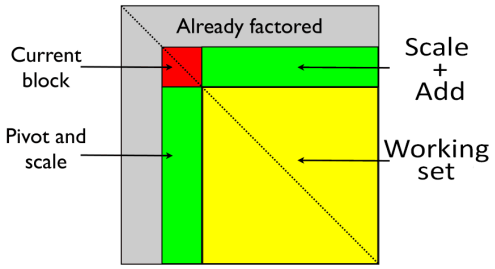

Figure 1: ***Sequential Algorithm with 50 Customers*** -

The Collumn loop Test were carried to measure the relative computation time for each algorithm with an increasing set of customers. The tests were carried out fifty times and the results averaged to mitigate external factors.
The program was executed on a linux based server, the Java Virtual machine was allocated an initial memory size of 2GB to avoid heap increase slowdowns and the JVM was also running in server mode. These conditions should result in the best possible testing environment in terms of repeatable and consistent results.

Daxpy A basic solution in which a single truck was allocated to every customer was created, forming the most ineffective solution possible. This was used as the baseline set of data that the Clarke-Wright solutions were compared against. The solution cost and total number of routes for each algorithm were compared to each other and to the baseline numbers.

3 Optimisation Method

Code Simplification Each solution produced by the algorithms was tested by looping through the routes, checking that each customer was visited and that the correct quantity was delivered. The total quantity of deliveries for each route was also calculated, assuring that no route was over the capacity of the truck.

Parallelised Daxpy Each solution produced by the algorithms was tested by looping through the routes, checking that each customer was visited and that the correct quantity was delivered. The total quantity of deliveries for each route was also calculated, assuring that no route was over the capacity of the truck.

SIMD Daxpy Each solution produced by the algorithms was tested by looping through the routes, checking that each customer was visited and that the correct quantity was delivered. The total quantity of deliveries for each route was also calculated, assuring that no route was over the capacity of the truck.

MDaxpy Each solution produced by the algorithms was tested by looping through the routes, checking that each customer was visited

and that the correct quantity was delivered. The total quantity of deliveries for each route was also calculated, assuring that no route was over the capacity of the truck.

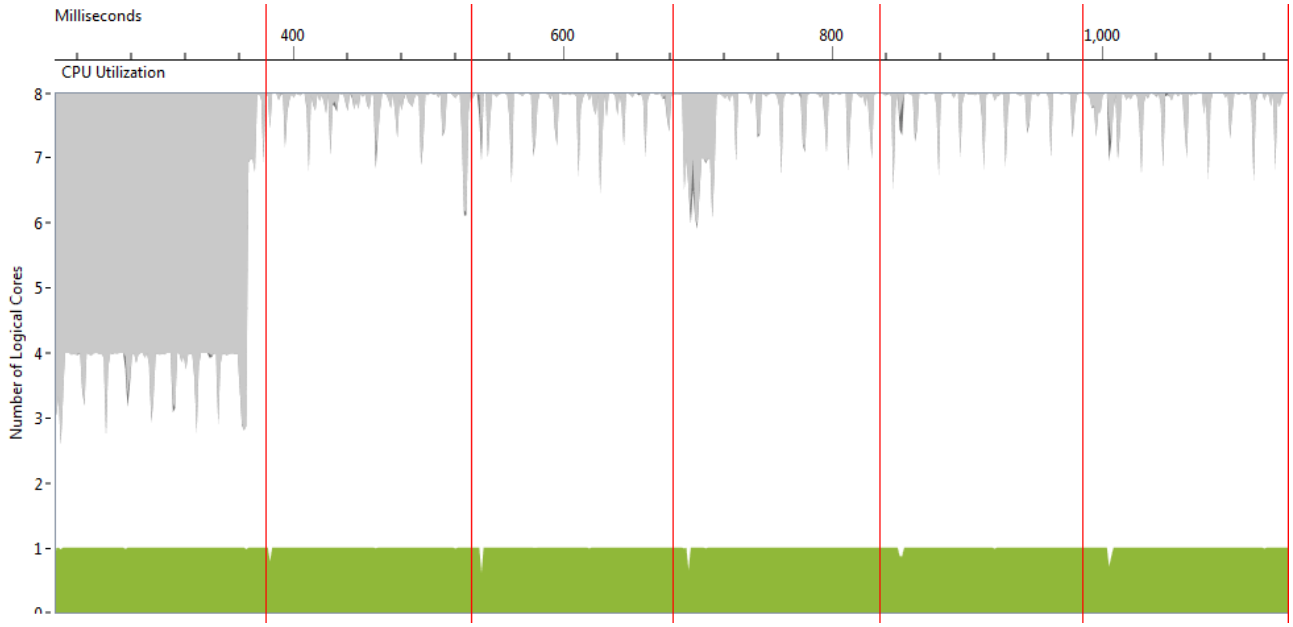

Figure 2: ***Single Threaded, 6 runs, simd256 Daxpy*** *- Overall system CPU utilisation*

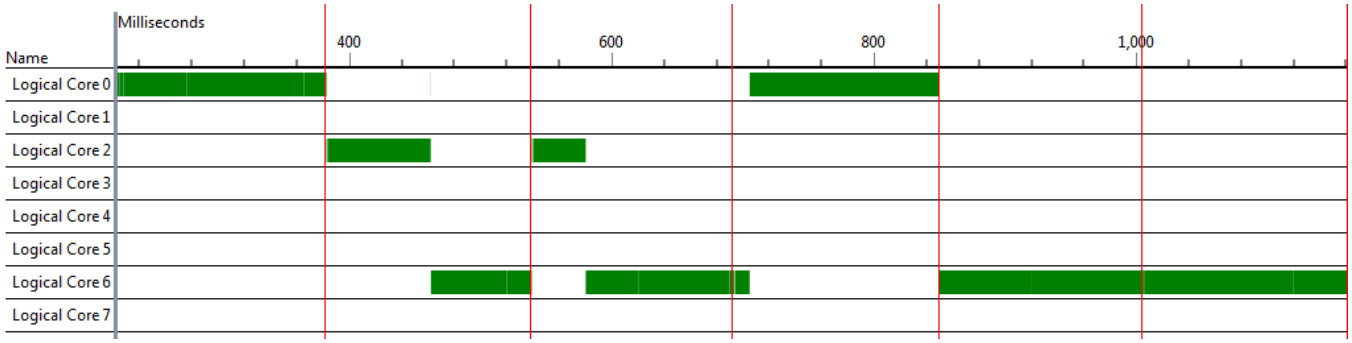

Figure 3: ***Single Threaded, 6 runs, simd256 Daxpy*** *- Thread to CPU Core allocation*

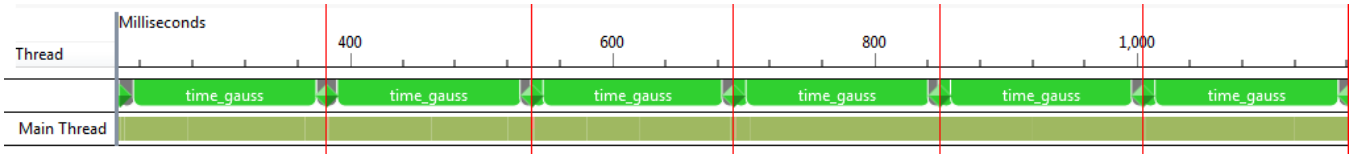

Figure 4: ***Single Threaded, 6 runs, simd256 Daxpy*** *- Thread Status*

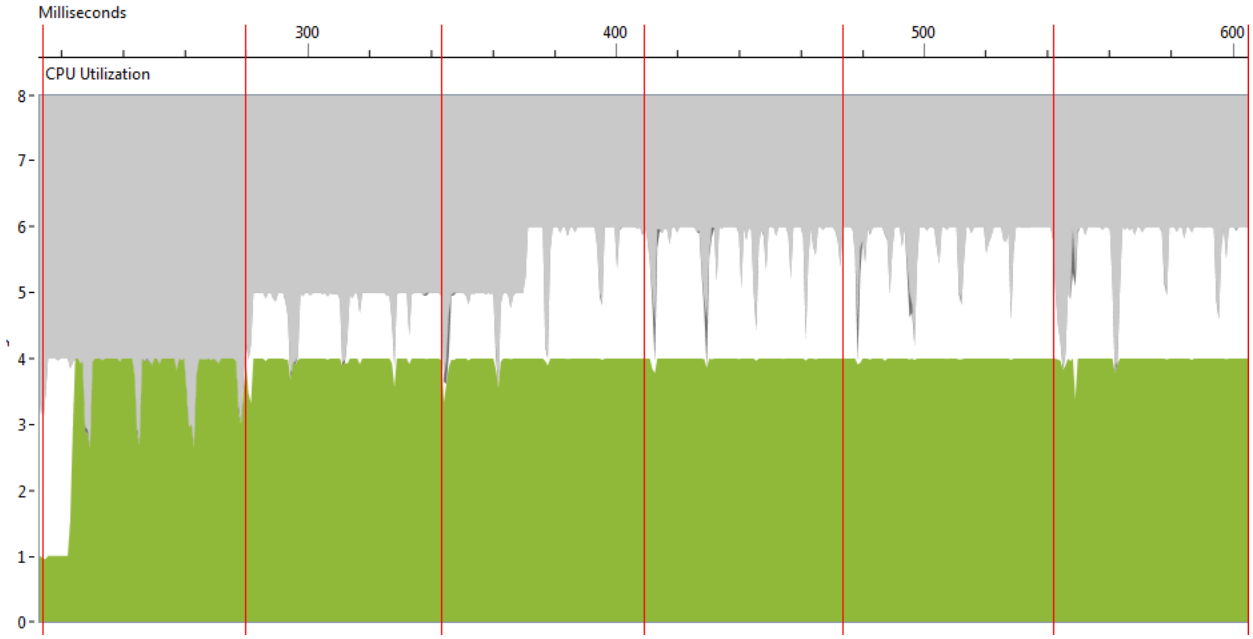

Figure 5: ***4 Threads, 6 runs, simd256 Daxpy*** *- Overall system CPU utilisation*

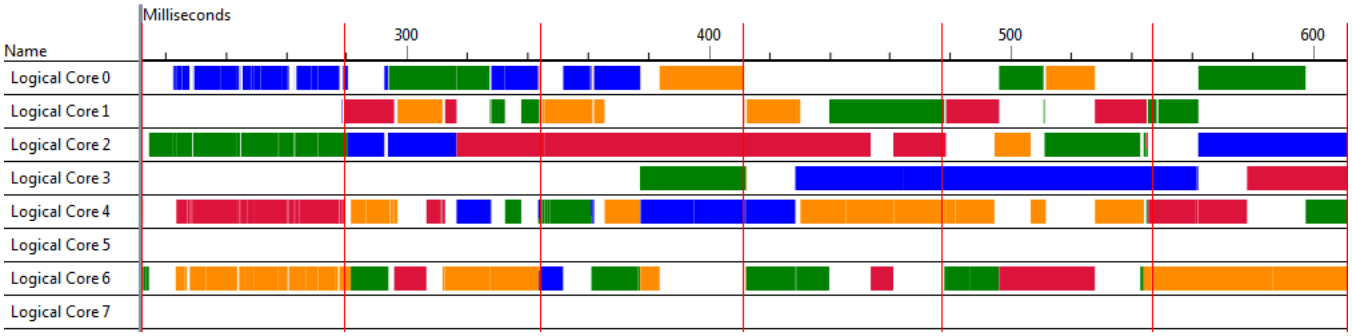

Figure 6: ***4 Threads, 6 runs, simd256 Daxpy*** *- Thread to CPU Core allocation*

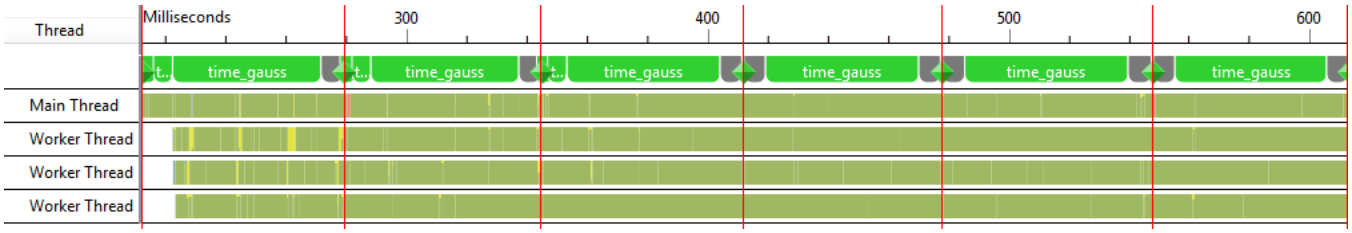

Figure 7: ***4 Threads, 6 runs, simd256 Daxpy*** *- Thread Status*

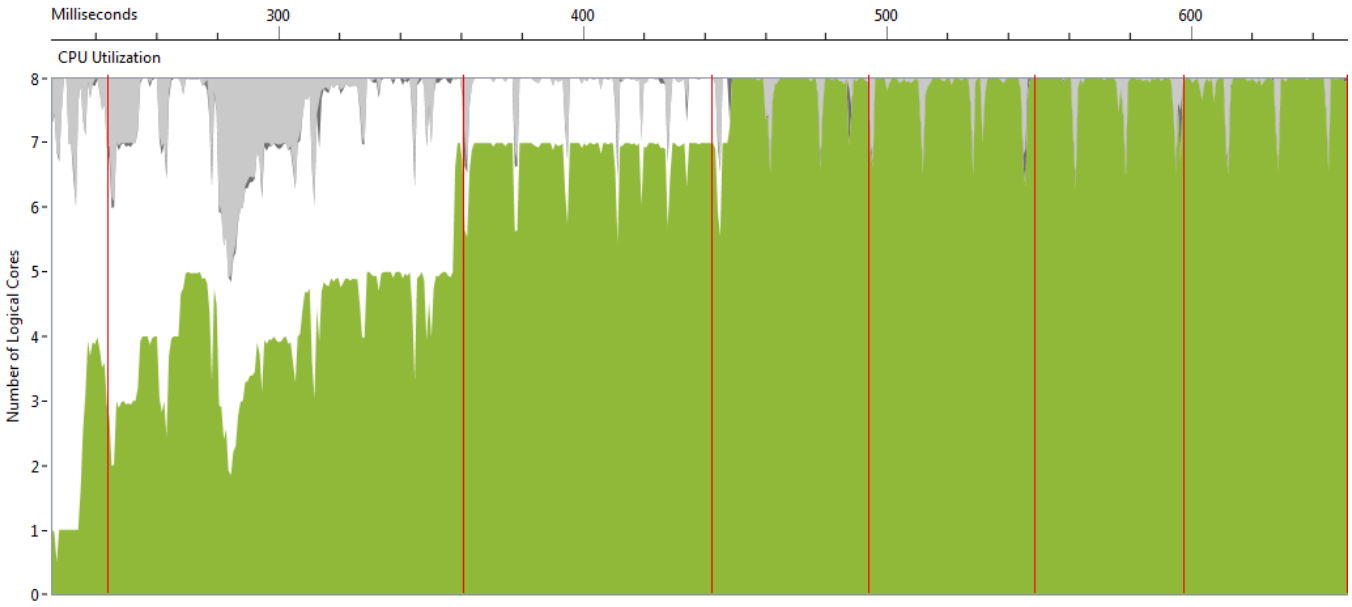

Figure 8: ***8 Threads, 6 runs, simd256 Daxpy*** *- Overall system CPU utilisation*

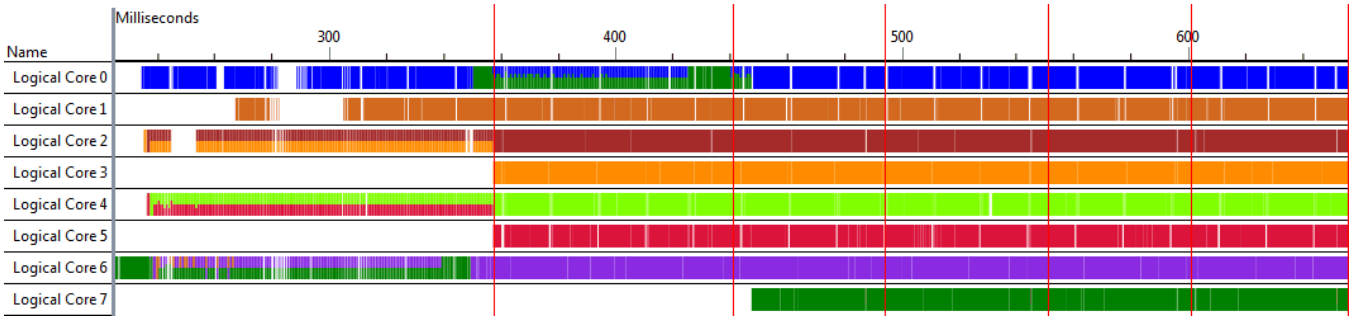

Figure 9: ***8 Threads, 6 runs, simd256 Daxpy*** *- Thread to CPU Core allocation*

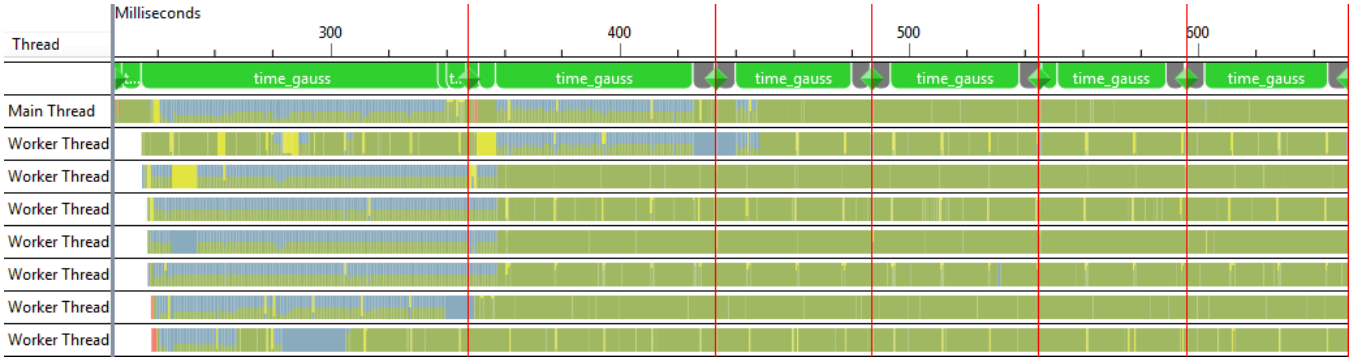

Figure 10: ***8 Threads, 6 runs, simd256 Daxpy*** *- Thread Status*

Number of Customers	Dumb solution cost	Sequential Cost	Parralel Cost	Sequential Routes	Parralel Routes	Sequential Time (Seconds)	Parralel Time (Seconds)
10	3781	1955	2027	2	2	0.000909157	0.001013164
20	7931	3749	3019	7	3	0.001404089	0.001417152
30	10302	5576	4717	11	5	0.000896818	0.000497884
40	15775	7271	4562	15	5	0.000736218	0.000602604
50	20073	9746	5753	21	7	0.001405751	0.00112024
60	23194	11918	6183	27	7	0.002299655	0.001860301
70	25925	12783	7293	31	8	0.003710384	0.002752865
80	28757	14238	7813	35	10	0.005782794	0.004625466
90	34580	17543	9202	41	11	0.011383535	0.008565139
100	38704	19375	9854	46	13	0.012820712	0.011544076
150	58323	29787	14632	71	18	0.042564373	0.035033029
200	76929	38920	17809	96	23	0.112553256	0.087635926
250	94218	47177	19586	119	28	0.236658078	0.19809618
300	116481	58912	22549	145	32	0.458011761	0.3729932
350	132529	67566	25634	171	37	0.795852936	0.655195371
400	153609	78127	31412	196	46	1.242856913	1.10756995
450	174708	88128	32206	220	51	1.909523115	1.674798974
500	188740	96343	35368	246	58	2.837620963	2.831905538
550	208973	105802	37923	270	63	4.026630142	4.657466973
600	228962	116788	43113	296	71	5.720755786	5.927701178
650	246924	126412	42089	322	72	7.251799771	7.534708523
700	266675	135277	44760	345	79	10.09592432	10.61137241
750	283723	143957	48740	370	82	12.4131069	12.93193556
800	311675	158686	51313	396	88	16.92304974	15.81973182
850	329742	168032	55694	422	97	21.56937135	18.40950886
900	347699	177212	59699	446	106	25.80634481	23.36173086
950	361993	184769	59529	472	105	30.69257549	29.51002702
1000	390783	199258	61701	497	109	39.20094816	35.98109894

Table 1: *Results of all tests carried out on both versions of the Clarke Wright algorithm*

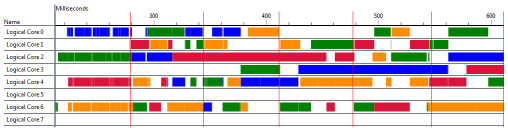

Figure 11: ***Parallel Algorithm with 20 Customers*** -

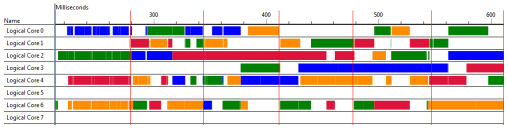

Figure 12: ***Sequential Algorithm with 20 Customers*** -

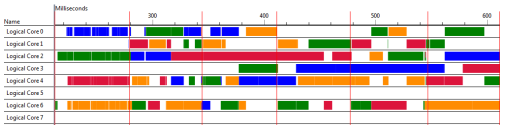

Figure 13: ***Parallel Algorithm with 50 Customers*** -

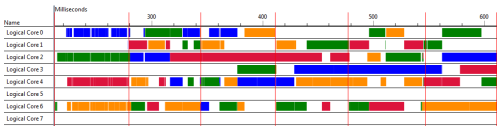

Figure 14: ***Sequential Algorithm with 50 Customers*** -

4 Conclusions

Computation time Both Implementations of the Clarke-Wright algorithms produced expected results, with the parallel version producing larger and fewer routes. As for the time taken to calculate, the performance is roughly equal. The discrepancies shown in Figure **??** when the amount of customers increases beyond 800 is possibly due to optimisations carried out by the Java virtual machine. The total operations carried out is roughly the same for each algorithm, however the arrays are accessed and modified at different times, this is a possible cause for the difference in processing time.

Solution Quality The Parallel solution produced a large saving of up to a 600% increase against the baseline cost, shown in Figure **??**. The Sequential solution produced a constant saving of around 200%. These results are also shown in Figure **??**, showing the number of routes.

Edge Cases It is possible that a customer can be left out of all routes due to capacity constraints; this is checked for at the end of the calculation. If a customer is left over, it is seen if it would be possible to add it to any existing route and then if it would be more efficient than sending out a new truck. This can be seen in Figure **??**, the customer in the top left falls in to this edge case category.

Conclusion The implementation written for this report successfully computes optimised and usable data, the processing cost increases in a quadratic relation to the size of data. The specific implementation could be optimised to produce quicker results. One possible optimisation route could be a custom sort method, as profiling reported that 40% of the processing time is taken by the initial sort of the customer pairs.
Overall this report produced repeatable and meaningful data, and can be seen as a successful investigation into the Clark-Wright Algorithm.

References

CLARKE, G., AND WRIGHT, J. 1962. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research 12*, 4, 568–581.

LYSGAARD, J. 1997. Clarke and wright's savings algorithm http://pure.au.dk/portal-asb-student/files/36025757/bilag_e_savingsnote.pdf. *Department of Management Science and Logistics, The Aarhus School of Business*.

5 Appendix: Code

5.1 ClarkeWright.java

5.2 VRSolution.java

Lines 20 to 28

5.3 Experiment.java