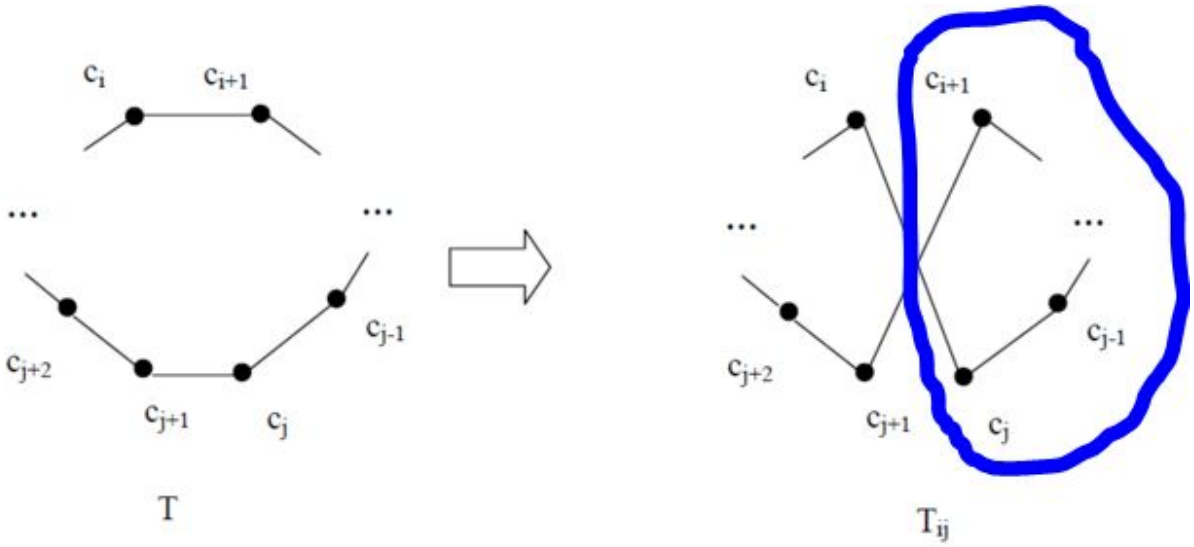
Robert Jones & Travis E Michael
Prof Lupo
CPE 458
Lab1 Report

Algorithm:

The full algorithm we used was inspired from the graphic accompanying the initial algorithm description contained in the lab specification. We first began by trying to simplify the concept of travelling nodes in such a way that we ensured only a single, complete cycle existed. This resulted in requiring that our graph be directional, therefore defining our graph map to require both a “before” and “after” element for each city. For the sake of simplifying communication and allocation of graph maps we condensed this information into a one dimensional array.

Applying a random mutation to a “tour” using the path switching method shown below then became an issue because direction then became distorted with each mutation.



The solution at this point was to traverse all nodes between C_{i+1} and C_j (circled in blue) and swap their “before” and “after” numbers. This ensured that the total distance could be easily calculated and the cycle maintained with a relatively small amount of complexity, all by simply maintaining directionality.

After being able to calculate a new tour total distance we decided to sync the results of each iteration with the root process and determine whether or not an improvement was found. If found the new best path would be broadcasted to the rest of the nodes so that no iterations would be wasted on sub par path configurations.

Problems:

We ran into plenty of problems with MPI_Bcast() where the flow of execution would continue where we thought the broadcast would block until all processes had reached the function call. This proved to be an incorrect assumption and therefore the use of a sequential MPI_Barrier() immediately following the bcast was required. Other MPI functions ran as expected after reading their documentation.

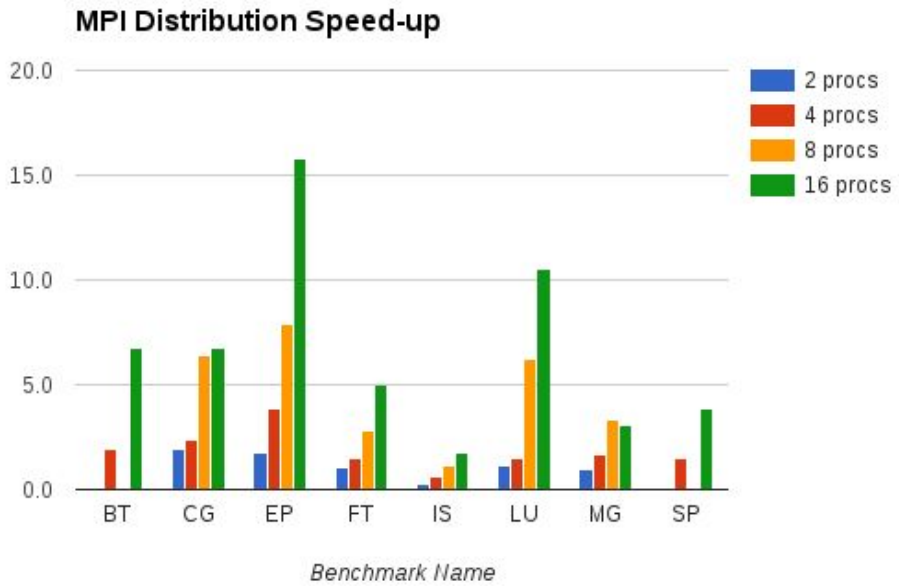
Another major problem faced was MPI_Finalize() causing a segmentation fault. This was happening when we were using a single node to parse the input file and then pass the retrieved data to all other nodes. To fix this, we allowed all nodes to parse the file individually.

Results:

Part A:

# Processes	BT	CG	EP	FT	IS	LU	MG	SP
1	271.80	139.86	80.11	68.12	2.33	214.18	7.78	181.81
2		73.28	45.30	66.92	10.26	186.12	8.03	
4	141.37	58.81	20.82	45.27	3.83	144.18	4.62	124.75
8		21.80	10.10	24.02	2.11	34.60	2.32	
16	40.04	20.77	5.06	13.68	1.31	20.43	2.53	47.38

Note: Benchmarks BT and SP can only be run with # processes equal to a square number



Part B:

Below are examples of the best found solutions for some of the data sets. To test, the program was run with 1000000 iterations until there were three runs in a row on a data set that did not produce a better result.

Input Name	Best Distance	# of Iterations Required
a280	2729.73	70064
ali535	2192.41	605303
att48	34084.36	3061
att532	95060.61	485345
berlin52	7834.18	1955
bier127	126125.47	15949
burma14	30.88	37
ch130	6718.67	19318
ch150	6980.52	25134
d1291	58864.42	2430551
eil101	692.37	17062
fl417	12901.29	275439
gil262	2643.10	64994
gr202	518.10	45999
kroA100	22164.58	8923
kroB100	24063.36	7461
kroC100	21797.43	11348
kroD100	22772.84	12654
kroE100	22736.79	9203
lin105	15404.42	9848
pcb442	56716.18	182477
pr299	52880.00	85287
rat195	2472.33	29116
rd400	16770.16	196340
ts225	130959.24	52842
tsp225	4077.60	66904
u574	39310.77	262269
ulysses22	75.31	337

Note: Not all provided data sets are shown/supported