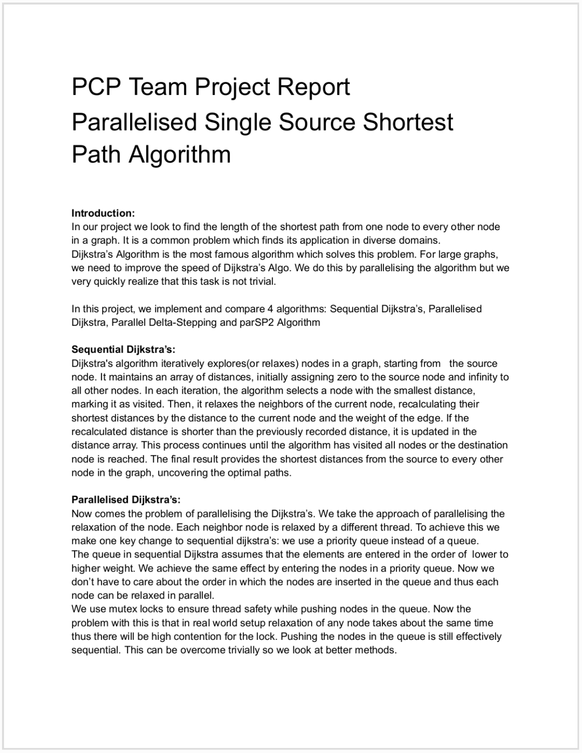
Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author:	Aman Panwar
Assignment title:	Final Project Report
Submission title:	Term Project
File name:	Turnitin_Prjt-CS20BTECH11004.pdf
File size:	190.78K
Page count:	6
Word count:	1,468
Character count:	7,213
Submission date:	03-Dec-2023 09:09PM (UTC+0530)
Submission ID:	2246017456



PCP Team Project Report
Parallelised Single Source Shortest Path Algorithm

Introduction:
In our project we look to find the length of the shortest path from one node to every other node in a graph. It is a common problem which finds its application in diverse domains.
Dijkstra's Algorithm is the most famous algorithm which solves this problem. For large graphs, we need to improve the speed of Dijkstra's Algo. We do this by parallelising the algorithm but we very quickly realize that this task is not trivial.

In this project, we implement and compare 4 algorithms: Sequential Dijkstra's, Parallelised Dijkstra, Parallel Delta-Stepping and parSP2 Algorithm

Sequential Dijkstra's:
Dijkstra's algorithm iteratively explores(or relaxes) nodes in a graph, starting from the source node. It maintains an array of distances, initially assigning zero to the source node and infinity to all other nodes. In each iteration, the algorithm selects a node with the smallest distance, marking it as visited. Then, it relaxes the neighbors of the current node, recalculating their shortest distances by the distance to the current node and the weight of the edge. If the recalculated distance is shorter than the previously recorded distance, it is updated in the distance array. This process continues until the algorithm has visited all nodes or the destination node is reached. The final result provides the shortest distances from the source to every other node in the graph, uncovering the optimal paths.

Parallelised Dijkstra's:
Now comes the problem of parallelising the Dijkstra's. We take the approach of parallelising the relaxation of the node. Each neighbor node is relaxed by a different thread. To achieve this we make one key change to sequential dijkstra's: we use a priority queue instead of a queue.
The queue in sequential Dijkstra assumes that the elements are entered in the order of lower to higher weight. We achieve the same effect by entering the nodes in a priority queue. Now we don't have to care about the order in which the nodes are inserted in the queue and thus each node can be relaxed in parallel.
We use mutex locks to ensure thread safety while pushing nodes in the queue. Now the problem with this is that in real world setup relaxation of any node takes about the same time thus there will be high contention for the lock. Pushing the nodes in the queue is still effectively sequential. This can be overcome trivially so we look at better methods.