Parallelization of Travelling Salesman Problem

Sanketh NS [221IT060]
Vrushank TS [221IT083]
Nithin S [221IT085]

Problem Statement

Problem Desctiption

- The Travelling Salesman Problem (TSP) is an NP-hard optimization problem.
- It involves a salesman who must visit a set of cities, each exactly once, and return to the starting city.
- The objective is to determine the shortest possible route that visits all cities.

Why TSP is Important ?

- TSP has wide applications in logistics, route planning, manufacturing, and circuit design.
- Due to its complexity, finding efficient solutions, especially for large datasets, remains a significant computational challenge.

Literature Survey

Year	Technique(s) Used	Key Finding(s)	Parameter(s) Analyzed	Research Gaps
2023	Branch and Bound (BnB) with OpenMP for parallelism	Parallel BnB speeds up TSP search	Runtime and core count	Needs verification on large problem instances
2017	Modified Balas' and Christofides' algorithm	Achieved better performance on TSP with up to 800 nodes.	Running time on different node degrees	High memory consumption for larger TSPs.
2014	Ant Colony Optimization (ACO) using OpenCL parallel implementation	Formula to calculate the probability to visit edge k after edge i.	Dependency of the probability on the pheromone content.	Need hybrid ACO on GPU and optimize GPU resource usage

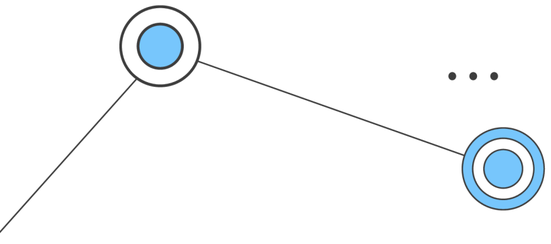
Sequential Solution For TSP

1. Brute Force Approach

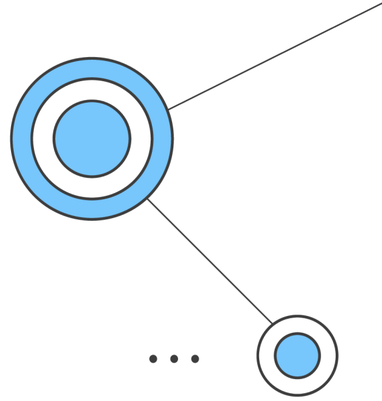
- In the brute force method, all possible permutations of city routes are generated and their total distances are calculated. The shortest one is selected as the optimal route.
- Time Complexity: O(n!) (Factorial growth makes it impractical for large n).

2. Dynamic Programming Approach

- The DP approach solves the problem by breaking it into smaller subproblems and using memoization to avoid redundant calculations.
- Time Complexity: O(n^2 * 2^n) (More efficient than brute force, but still exponential).



Methodology for Parallel Execution



Thread-Based Performance Improvement:

- The brute force method has factorial time complexity, and dynamic programming has exponential complexity. Using multiple threads to divide the workload allows each CPU core to handle a separate computation, reducing overall execution time.

Improved Scalability with Multithreading:

- As the number of cities increases, the number of computations grows rapidly. By utilizing multithreading, we can distribute these independent tasks (e.g., different paths or sub-problem evaluations) across multiple threads, ensuring better CPU utilization and scalability for larger graphs.

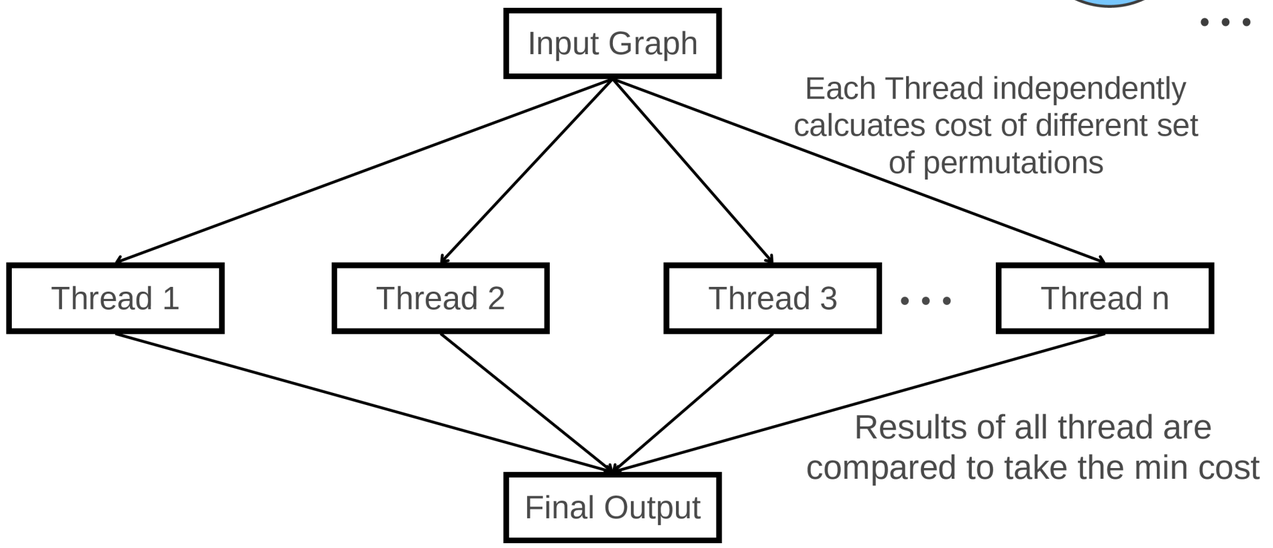
Parallel Sub-problem Evaluation in Dynamic Programming:

- In the dynamic programming approach (Held-Karp), the recursive steps involve solving for subsets of cities. These sub-problems are independent of each other. By paralleling these sub-problem evaluations, multiple threads can work simultaneously on different subsets, leading to faster computation. Threads can share the memoization table and update it with results as they complete their tasks.

Concurrent Regions for Parallel Execution

Algorithm 1 Brute Force Serial TSP

Input: *city_list*, *cost_matrix*
optimal_cost $\leftarrow \infty$
optimal_path $\leftarrow$ *null*
city_list $\leftarrow$ *list of cities excluding* $city_1$
while *next_permutation*(*city_list*) **do**
  temp_cost $\leftarrow$ *get_path_cost*(*city_list*, *cost_matrix*)
  {Cost of travelling cities in order of cities in *city_list*}
  if *temp_cost* $<$ *optimal_cost* **then**
    optimal_cost $\leftarrow$ *temp_cost*
    optimal_path $\leftarrow$ *city_list* {with $city_1$ appended at start & end}
  end if
end while
Output: *optimal_path*, *optimal_cost*



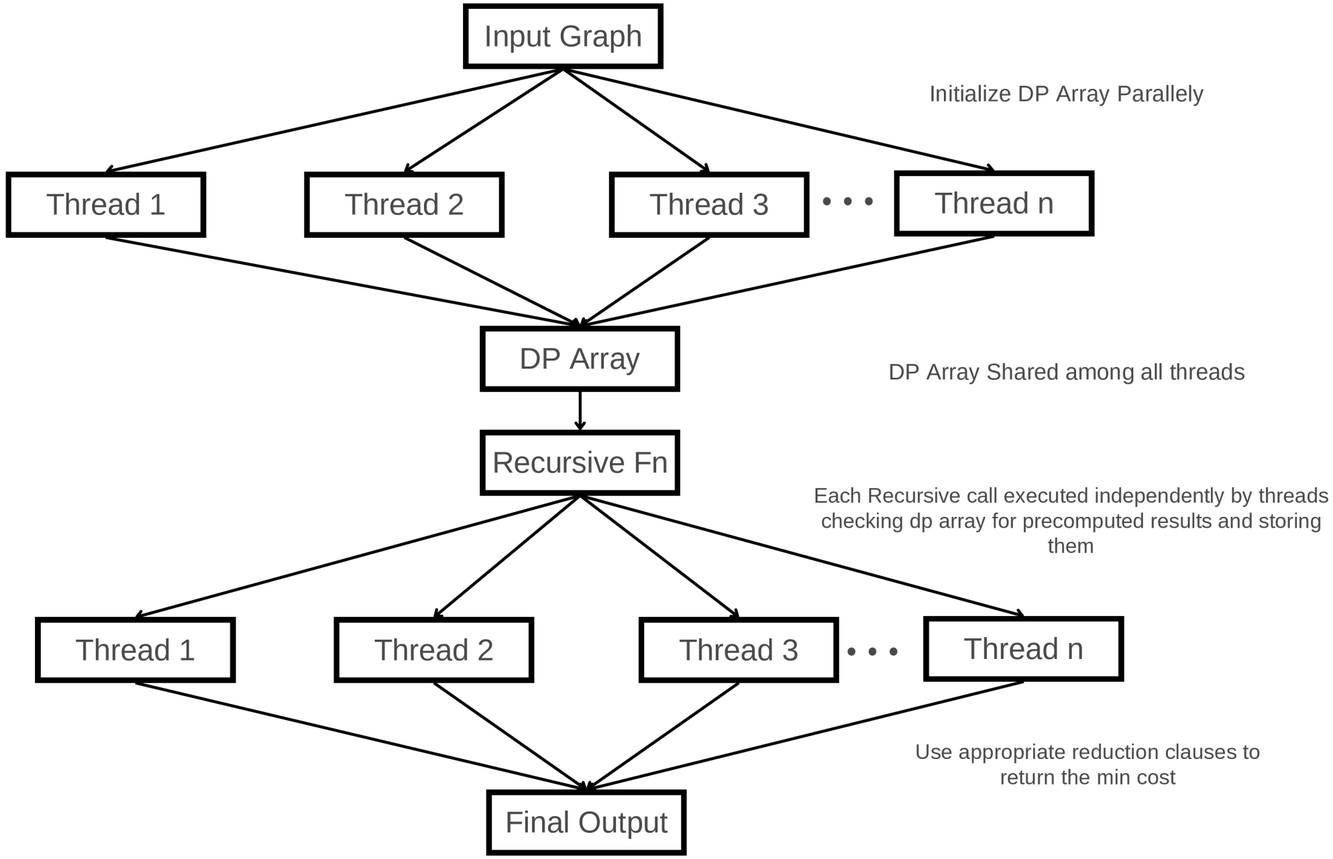
Concurrent Regions for Parallel Execution

Algorithm 1: Dynamic Programming algorithm for the original TSP

Data: A set of locations V, an arbitrary location $v_0 \in V$ and cost function c

Result: A shortest tour that visits all locations in V

```
1  Initialize D_TSP with values ∞ ;
2  Initialize a table P to retain predecessor arcs ;
3  foreach w ∈ V do
4      D_TSP({w}, w) ← c(v_0, w) ;
5  for i = 2, ..., |V| do
6      for S ⊆ V where |S| = i do
7          foreach w ∈ S do
8              foreach u ∈ S \ {w} do
9                  v ← D_TSP(S \ {w}, u) + c(u, w) ;
10                 if v < D_TSP(S, w) then
11                     D_TSP(S, w) ← v ;
12                     P(S, w) ← (u, w) ;
13 return path obtained by backtracking over arcs in P starting at P(V, v_0) ;
```



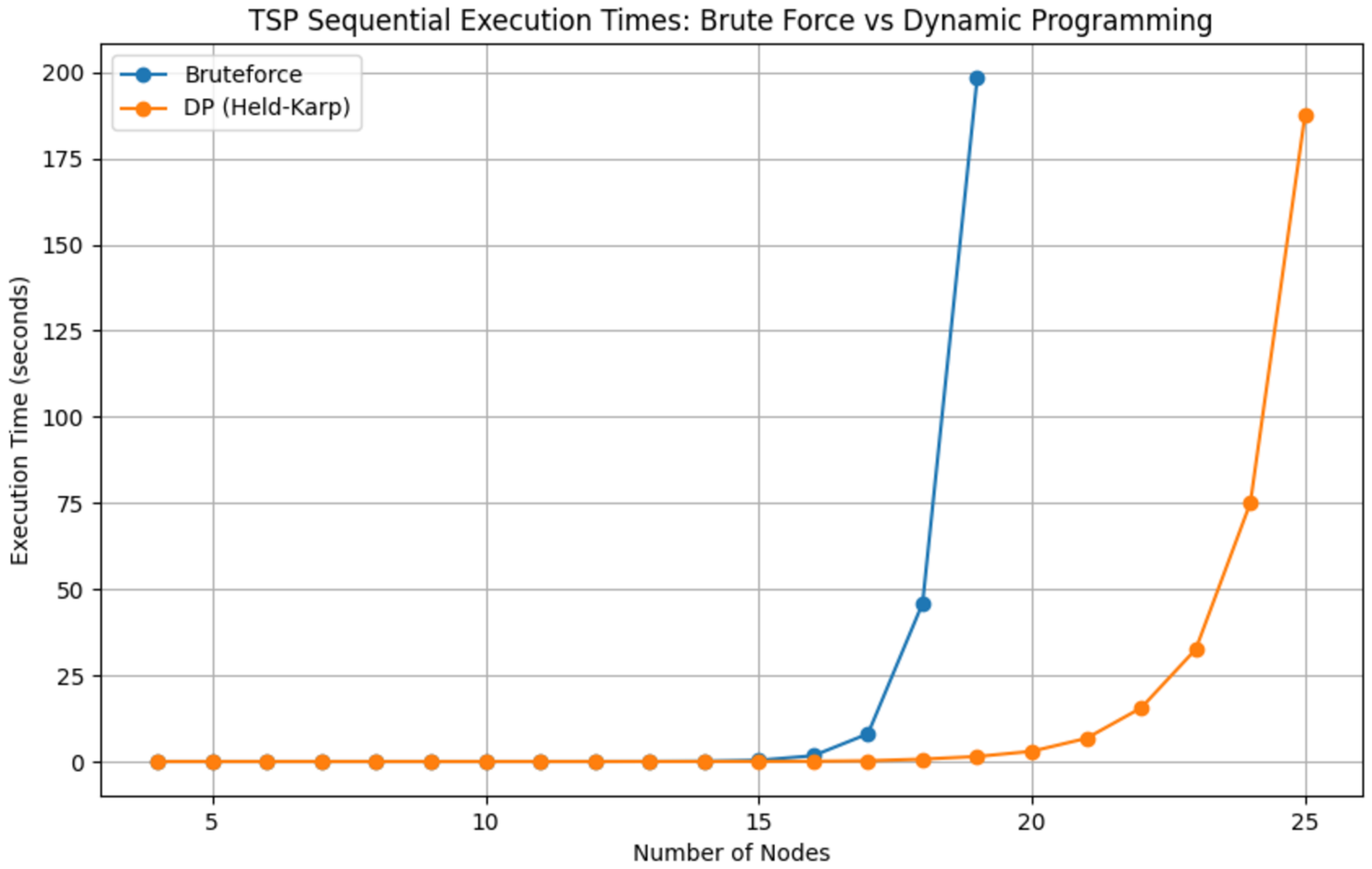
Sequential Brute Force Execution Results

```
vrushank@vrushank:~/All Codes/Projects/Parallel_TSP/Parallel-TSP/Sequential/Bruteforce$ gcc TSP_sequential_bruteforce.c
vrushank@vrushank:~/All Codes/Projects/Parallel_TSP/Parallel-TSP/Sequential/Bruteforce$ ./a.out
Nodes: 3, Execution Time: 0.000004 seconds, Min Cost: 142
Nodes: 4, Execution Time: 0.000005 seconds, Min Cost: 113
Nodes: 5, Execution Time: 0.000007 seconds, Min Cost: 49
Nodes: 6, Execution Time: 0.000014 seconds, Min Cost: 149
Nodes: 7, Execution Time: 0.000034 seconds, Min Cost: 134
Nodes: 8, Execution Time: 0.000108 seconds, Min Cost: 194
Nodes: 9, Execution Time: 0.000377 seconds, Min Cost: 172
Nodes: 10, Execution Time: 0.001569 seconds, Min Cost: 253
Nodes: 11, Execution Time: 0.005524 seconds, Min Cost: 205
Nodes: 12, Execution Time: 0.016696 seconds, Min Cost: 248
Nodes: 13, Execution Time: 0.055953 seconds, Min Cost: 210
Nodes: 14, Execution Time: 0.266343 seconds, Min Cost: 233
Nodes: 15, Execution Time: 1.095573 seconds, Min Cost: 272
Nodes: 16, Execution Time: 5.894770 seconds, Min Cost: 281
Nodes: 17, Execution Time: 26.892775 seconds, Min Cost: 314
Nodes: 18, Execution Time: 82.153411 seconds, Min Cost: 294
Nodes: 19, Execution Time: 511.470190 seconds, Min Cost: 269
vrushank@vrushank:~/All Codes/Projects/Parallel_TSP/Parallel-TSP/Sequential/Bruteforce$
```

Sequential DP Execution Results

```
vrushank@vrushank:~/All Codes/Projects/Parallel_TSP/Parallel-TSP/Sequential/DP$ gcc tsp_dp_sequential.c
vrushank@vrushank:~/All Codes/Projects/Parallel_TSP/Parallel-TSP/Sequential/DP$ ./a.out
Nodes: 4, Execution Time: 0.000007 seconds, Min Cost: 140
Nodes: 5, Execution Time: 0.000011 seconds, Min Cost: 171
Nodes: 6, Execution Time: 0.000020 seconds, Min Cost: 171
Nodes: 7, Execution Time: 0.000049 seconds, Min Cost: 217
Nodes: 8, Execution Time: 0.000097 seconds, Min Cost: 198
Nodes: 9, Execution Time: 0.000237 seconds, Min Cost: 150
Nodes: 10, Execution Time: 0.000673 seconds, Min Cost: 99
Nodes: 11, Execution Time: 0.001512 seconds, Min Cost: 134
Nodes: 12, Execution Time: 0.003516 seconds, Min Cost: 152
Nodes: 13, Execution Time: 0.013156 seconds, Min Cost: 114
Nodes: 14, Execution Time: 0.039563 seconds, Min Cost: 144
Nodes: 15, Execution Time: 0.094203 seconds, Min Cost: 162
Nodes: 16, Execution Time: 0.217278 seconds, Min Cost: 189
Nodes: 17, Execution Time: 0.608848 seconds, Min Cost: 214
Nodes: 18, Execution Time: 1.647951 seconds, Min Cost: 194
Nodes: 19, Execution Time: 4.755034 seconds, Min Cost: 164
Nodes: 20, Execution Time: 11.522789 seconds, Min Cost: 183
Nodes: 21, Execution Time: 32.928224 seconds, Min Cost: 135
Nodes: 22, Execution Time: 63.838463 seconds, Min Cost: 185
Nodes: 23, Execution Time: 110.202514 seconds, Min Cost: 212
```

Comparison Graph of Brute Force and DP Execution Times



References

- **An efficient parallel approach for solving Travelling Salesman Problem using GPUs**
- https://ijeecs.iaescore.com/index.php/IJEECS/article/view/31763

- **A Comparative Analysis of Brute Force and Dynamic Programming Approaches for Solving the Travelling Salesman Problem**
- https://www.ijcaonline.org/archives/volume157/number7/26847-2017912780/

- **A Hybrid Parallel Algorithm for the Travelling Salesman Problem Using OpenMP and CUDA**
- https://www.sciencedirect.com/science/article/pii/S187705091732375X

- **Travelling Salesman Problem: GPU Accelerated Methods**
- https://hgpu.org/?p=12430

- **A Parallel Algorithm for Solving TSP Based on OpenMP**
- https://ieeexplore.ieee.org/document/6596453

Thank you!