Name: Krishna Biswakarma
Roll: 24CS60R71
Subject - ADA.

(1) Let the input be as following:

A complete weighted graph $G=(V,E)$ where,

V = set of cities. $|V|=n$

E = set of edges with weights $w(u,v)$ representing the distance between cities u and v.

A value 'K' which denotes the maximum allowed distance between any two consecutive cities in the tour.

State space defination for CTSP:

- It is defined as the tuple (s,c) where,
  $S \subseteq V$ is the set of cities visited so far, excluding the starting city

NOTE- S = start city (initially).

- $c \in V$ is the current city in the tour (starting at the given initial city),
- Given remaining cities to be visited are $V-S$.

Here, the state represents the partial tour with a specific set of cities visited with the current position in the tour.

Also, a constraint of 'K' should be enforced between city 'c' and the next city to visit must not exceed 'K'.

Heuristic Estimate (Lower Bound):

Let $h(s,c)$ be a heuristic function for CTSP.

We can use the MST as a guaranteed lower bound heuristic.

To create the heuristic we can follow the below steps:

i) $V-S$ be the remaining cities. Compute MST over the subgraph.

NOTE - Only edge with weight $\leq K$ will be considered.

$\therefore h(s,c)$ = Weight of MST → This gives the LB on the cost of completing the tour starting from the current state.

(2) The statement is "partially true".

Comparision-

A^*: Expands nodes with lowest cost i.e., $f(n) = g(n) + h(n)$

$g(n)$ = Actual distance

$h(n)$ = Heuristic cost.

It is proven the A^* gives optimal result with fewest nodes possible given an admissible i.e., non-overestimating heuristic.

DFBB: It explores the depth first until it either finds a sol^n or the path cost becomes too high. If the meanwhile we keep on updating the bound whenever it has the lowest cost.

Heuristic Estimate and Node Expansion:

If both uses same admissible heuristic then both should agree on the potential quality of solution in terms of cost. However DFBB may expand nodes that A^* won't expand; because it don't prioritize nodes based on $f(n)$ but goes on expanding the depth.

Cases where DFBB expands extra nodes:

DFBB may expand more nodes because it explores each path or prune it without choosing the lowest cost node first. This means it will also expand the path that A^* won't.

∴ DFBB may expand nodes in the path with relatively high $g(x)$ even if better paths are there.

Hence, It is partially true because DFBB could indeed expand every node that A^* expands if both algorithm are running on the same problem with similar heuristic.

However, DFBB may additionally expand some nodes that A^* won't due to its DFS behaviour and lack of storing global $f(x)$.

(3) The statement is partially true.

A* we saw previously how it expands the nodes based on global value of f(n) where, $f(n) = g(n) + h(n)$

g(n): Actual cost, h(n): Heuristic cost.

- IDA expands all nodes that A* will –

Since IDA is performing DFS with increasing cut-off based on f(n) it will eventually expand all nodes that A* will expand provided the search reaches the same state.

- IDA may expand some nodes that A* wont –

Since IDA uses DFS with thresholding mechanism, it may explore nodes in different order and may explore nodes that A* doesn't particularly if the nodes have f(n) value above the current threshold but still leads to other potentially optimal path. Specifically IDA* might expand nodes early in search that A* would have expanded later due to its greedy priority queue. This happens because IDA* explores all possibility in DFS manner without the guarantee of always expanding the least f(n) value node at each steps.

∴ The statement is partially true because IDA* will indeed expand what A* will but will also expand what A* wont because of its higher threshold at each iteration.

(2) (4) Genetic algorithm for TSP -

we represent each solution as a sequence of cities.

- Chromosome Representation:

Each chromosomes represent a tour of cities. i.e ordered list of integers.

Eg: 1, 3, 5, 2, 4 → cities.

- Fitness Function:

Fitness = - (Total Distance + Constraint Penalty)

Actual distance

Penalty added due to constraint violation.

- Population:

Permutation of cities.

Ensure cities does not repeat.

- Selection:

Tours with higher fitness value for reproduction. We may use Roulette / Tournament selection method.

- Crossover:

Combining two parent chromosomes to create new offspring we may either use, Ordered Crossover / Partially Mapped crossover for this.

- Mutation:

Introducing small random changes to maintain diversity in the population and avoid premature convergence.

Swap mutation - Randomly swap two cities.

Inversion mutation - Select two cities and reverse the order.

Insert mutation - Remove the city and reinsert it

- Constraint Handling -

Using penalty based approach - As we saw above adding penalty to fitness function.

- Terminating Condition -

i) Max. no. of generations reached.

ii) No improvement in best sol", iii) High fitness value is found.