|  |  |
| --- | --- |
| Shri Vile Parle Kelavani Mandal's  **INSTITUTE OF TECHNOLOGY**  **DHULE (M.S.)**  **DEPARMENT OF COMPUTER ENGINEERING** | |
| **Name:** Anushka Paras Jain **Roll No.:** 01  **Subject :** Artificial Intelligence Lab **Subject Code :** BTCOL707  **Class:** Final Year Comp. Engg. **Expt. No. :** 07  **Title :** Solve traveling salesman problem. | |
| **Problem Statement:**  **Software Required:**  **Theory:**  **Conclusion:** | Solve traveling salesman problem.    Prolog  A salesman is given a list of locations in the Traveling Salesman issue (TSP), a classic optimization issue. His task is to determine the shortest route that visits each city exactly once and returns to the beginning city. The optimal solution of TSP for a large number of cities can be computationally costly. In this example, I'll give you a Prolog code that uses a brute-force method to solve a simple TSP instance. Remember that large instances of the problem are inefficient using this code.  % Define the cities and distances between them  distance(city1, city2, 10).  distance(city1, city3, 15).  distance(city1, city4, 20).  distance(city2, city3, 35).  distance(city2, city4, 25).  distance(city3, city4, 30).  % Create a list of cities  cities([city1, city2, city3, city4]).  % Predicate to calculate the total distance of a tour  tour\_distance([], 0).  tour\_distance([\_], 0).  tour\_distance([City1, City2 | Rest], TotalDistance) :-  distance(City1, City2, Dist),  tour\_distance([City2 | Rest], RestDistance),  TotalDistance is Dist + RestDistance.  % Predicate to find the shortest tour  shortest\_tour(ShortestTour, ShortestDistance) :-  cities(CityList),  permutation(CityList, Tour),  append(Tour, [Tour], ClosedTour),  tour\_distance(ClosedTour, Distance),  (ShortestDistance =< 0; Distance < ShortestDistance),  ShortestTour = Tour,  ShortestDistance = Distance.  % Entry point to solve the TSP  solve\_tsp :-  shortest\_tour(Tour, Distance),  write('Shortest tour: '), write(Tour), nl,  write('Shortest distance: '), write(Distance), nl.  % Start the solver  :- solve\_tsp.  Using the distance/3 predicate, we define the cities and their respective distances from one another.  A list of cities to visit is defined by the cities/1 predicate.  The tour\_distance/2 predicate is used to determine a tour's total distance.  The shortest\_tour/2 predicate determines the distance for each tour, creates all feasible combinations of the cities, and maintains track of the shortest tour's length.  The shortest tour and its distance are found and printed by the solve\_tsp predicate.  This code illustrates how to solve the TSP for a limited number of cities using a simple brute-force method. Near-optimal solutions are usually found for bigger instances using heuristics like the Christofides algorithm or the closest neighbor technique, or more efficient algorithms like dynamic programming and branch and bound.  Prolog's declarative nature can be used to represent the TSP and apply search algorithms to find solutions. Distances between cities are defined using predicates, and different permutations are explored using Prolog's backtracking mechanism. While this approach is simple, it becomes inefficient for large problems. |