CompSci 367/761 ASSIGNMENT 2: TRAVELLING SALESPERSON PROBLEM (TSP) : FINDING OPTIMAL TOUR

Due 27 August 11:59PM worth 5%.

1 Introduction

This assignment will be released in stages. This phase of the assignment will introduce you to the problem you are to solve. The next phase will introduce you to example road networks. These example road networks will provide you with test cases for your code. The final phase will provide the wrapper prolog code that will use your code to find the optimal tour for a TSP. Phase 2 will be available on Monday 5 August and the final phase will be available on Wednesday 7 August. You can begin trying to understand the assignment now. However, the asignment should become clearer as you learn prolog in the tutorial and the lectures this week. Please use Piazza to ask questions about and to discuss the assignments.

1. Declarative Programming

   The goal of declarative programming is to separate the *what* from the *how*. When you were "programming" the logic puzzle in assignment 1, you were doing, to a large extent, declarative programming. A large part of your program described *what* relationships must exist between the values of different variables. You did not describe *how* to find those values.

   In this assignment, you will be describing *what* it means for a list of cities to be a tour for a given start city and a road network. You should NOT be describing *how* that tour is found.

2. Context

   In this assignment you need to write prolog code to generate solutions for the asymmetric travelling salesman problem (see `https://en.wikipedia.org/wiki/Travelling_salesman_problem#Description`). The goal is to write a "prototype" solver for travelling salesman problems. Your assignment is make the code as declarative and as obviously correct as possible. Efficiency is of no concern here. Basically your code will "define" which it means for a list of cities to be a tour of a road network starting from and ending at a specified city.

3. Inputs and Outputs

   You must write the code for:
   solution(+Path, +RoadNetwork, -SolutionCost, -SolutionPath).
   See "Notation of Predicate Descriptions" (`https://www.swi-prolog.org/pldoc/man?section=preddesc`) for descriptions of the meaning of "+" and "-" in front of the argument names.
   The inputs are:

Path a list of cities, in reverse order of being visited (e.g., the last city in the list is the first city in the tour

RoadNetwork a description of all the directed roads between cities and their cost

The outputs are:

SolutionCost the cost of a tour of network;

SolutionPath the tour for which SolutionCost was computed.

2 Domain Description

1. Road Network

   The road network is a directed asymmetric weighted graph, the graph is not nessarily complete (i.e., there need not be edges between every two nodes in the graph). The interpretation is that the nodes are cities, the edges are one-way roads, and an edge weight is the distance between the two edge nodes (i.e., cities) in that direction. For example, the "road" between city a and city b might have cost 5 and the "road" between city b and city a might have cost 2 (or there might not a "road" going from city b to city a).

2. Tour Definition

   Given a starting city and a road network, a tour is defined as follows. A tour is defined to be a list of cities, such that:

   - every city in the network appears in the list at least once;
   - only the starting city can appear in the list more than once;
   - the starting city is both the first and the last element in the list;
   - the starting city appears in the list exactly twice.
   - the list contains exactly one more element than the number of distinct cities in the road network.

   For example, for a road network that only contains one city, there would only be one tour with two elements, that city appearing twice, e.g., let the city be a, then the tour is the list $[a,\ a]$. Note that since the graph need not be complete, not all road networks have tours. In this case, *solution/4* (the predicate *solution* with 4 arguments) should simply fail.

   The cost of a tour is the sum of the edge costs between the successive nodes in the tour list. The distance between any city and itself is zero. Therefore the cost of the tour $[a,\ a]$ is 0. If there were two cities, a and b, and the distance from a to b is 4 and the distance from b to a is 3, and a is the starting city then the only legal tour is the list $[a,\ b,\ a]$, which has a cost of 7. If there were no road (edge) from b to a then *solution/4* will fail.