INTELLIGENT SYSTEMS PROJECT REPORT

Hrishikesh Deshpande

**Topic – Travelling Salesperson Problem (Problem No 8 from specified list)**

**Problem Description –**

The travelling salesperson problem(TSP) is a touring problem in which given a set of interconnected cities, the travelling salesperson must travel each city exactly once and reach the starting point such that his tour is the shortest. The problem is NP-hard and there is no method to know the shortest route beforehand.

We need a unit square in which the cities will be specified and the distance between each city will be calculated based on their coordinates.

Formulate the problem so that you can apply A\* algorithm with Minimum Spanning Tree as a heuristic.Compare the results obtained through A\* algorithm

Devise a hill climbing approach to solve the TSP.

**Objective –**

The project aims to implement A\* algorithm and apply it to solving the TSP and also to devise and implement a method with which hill climbing can be applied for solving the TSP and then compare the results for both.

**Model –**

User Interface – The user interface is developed in HTML, CSS with Bootstrap 3 Framework and JavaScript using the AngularJS framework.

Problem Solving Logic – The problem-solving logic is developed using JavaScript with the AngularJS framework.

**Approach/Method –**

1) For A\* algorithm –

Initial State – The agent is in the city from which the tour starts.

Goal State – The agent has visited all the cities and reached the initial state.

Successor Function – Generates all the cities that have not been visited yet.

Path Cost – the distance from the initial state to current state along the edges followed to reach it.

Heuristic – The total distance of the minimum spanning tree formed from the graph of unvisited cities as specified in the question. Minimum Spanning Tree is a subgraph which is a tree with minimum edge costs.

2) Hill climbing search

Takes the input of cities, and randomly generates numerous possible next states and selects the one with the minimum total distance. It does this process reiteratively. An upper bound of 2000 iterations is placed on the hill climbing so that the code does not end up in infinite loop.

**Outline of work performed**

1) Planned logic for implementation of user interface

2)Implemented JavaScript canvas and related logic for generating cities and edges

3)Implemented formation of minimum spanning tree from given graph.

4)Implemented A\* algorithm along with necessary helper functions like priority queue and tree.

5) Tested A\* algorithm

6)Integrated implementation of A\* algorithm with user interface.

7)Tested A\* with user interface

8)Developed logic for hill climbing search.

9)Tested hill climbing search.

10)Integrated it with user interface.

11)Tested hill climbing and A\* algorithm both with user interface.

**External Documentation of program –**

The program uses HTML, CSS with Bootstrap 3 and JavaScript with AngularJS framework.

The program is divided into 3 parts –

1) travelling\_salesman\_problem.html – This is an html file that contains the presentation part of the program, it specifies how the components of the UI are to be laid out.

The user interface contains a toggle switch to select the solving strategy – A\* or hill climbing. There are 2 more buttons –

a) Reset – resets the canvas for the user to input a new problem

b) Get Solution – get the solution using the strategy selected by the user

c) Canvas – depicted as a black square, the user can click anywhere within it to create cities which are part of the instance of TSP.

2)tsp.css – This is cascaded style sheet file which specifies the styles for the various components inside the html file specified in point (1). This again is a part of the user interface i.e. presentation.

3)controller.js – This is the JavaScript file which contains the AngularJS controller called tspCtrl. This holds all the component bound models, event handlers and functions that belong to the main logic of the problem solving. AngularJS uses two-way data binding, the model for which are specified in the controller and can be directly used in the html within the scope of the controller.

Important functions within the controller.js file –

1) $scope.onCanvasClick(event) – when the user clicks on the canvas, this function is triggered and it draws a city at the co-ordinates where the user clicked and also creates a corresponding object describing the city and also draws edges and creates model object for them in case multiple cities already exist.

2)$scope.searchWithAstarAlgorithm() – this function contains implementation of the A star algorithm applied to the TSP. This function uses additional functions which are documented inline in the source code.

3)$scope.hillClimbingSearch() – this function contains implementation of hill climbing search applied to the TSP. This function again uses additional helper functions which are documented inline in the source code.

4)$scope.getSolution() – this function determines which solving method should be used depending on the user selection provided in the user interface.

5)$scope.reset() – resets the canvas for the user to input a new problem.

The controller also contains implementations of a priority queue and tree which are required for implementation of the A\* algorithm.

Details about framework used –

Bootstrap 3 – provides inbuilt generalized styles that can be directly applied to html components as classes or other attributes which ease the process of the developing good user interfaces.

AngularJS – It is a JavaScript framework which simplifies JavaScript code and reduces it by providing variety of features like two-way data binding, dependency injection, model view whatever architecture. It also provides ease in DOM manipulation, dynamic user interfaces and single page user interfaces.

JQuery – It is a java library that provided numerous simplified utilities for JavaScript ranging from DOM manipulation, http requests, event handling etc. used internally by bootstrap for dynamic styles and DOM manipulation.

**Program Source Codes**

1)travelling\_salesman\_problem.html

<!DOCTYPE HTML>

<html>

<head>

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<link href="css/bootstrap.min.css" rel="stylesheet" type="text/css"/>

<link href="tsp.css" rel="stylesheet" type="text/css"/>

<script src="jquery-3.2.1.min.js"></script>

<script src="js/bootstrap.min.js"></script>

<script src="angular.min.js"></script>

<script src="controller.js"></script>

</head>

<body style="background-color:#34495e" ng-app="app" ng-controller="tspCtrl">

<div class="container" style="margin:0;padding:0;width:100%">

<div style="display:block;position:fixed;background-color:#2c3e50;width:100%;font-size:20px;z-index:11;color:white;">

<ul class="nav">

<li class="nav-item" style="text-align:center">

<a href="#" class="nav-link" style="color:white;

cursor:default;">Travelling Salesman Problem</a>

</li>

</ul>

</div>

<div class="row" style="margin-top:6%;font-size:20px">

<div class="col-md-5">

<div align="right">A\* algorithm</div>

</div>

<div class="col-md-2">

<div align="center">

<div class="onoffswitch" align="center" >

<input type="checkbox" name="onoffswitch" class="onoffswitch-checkbox" id="myonoffswitch" ng-model="isHillClimbing" ng-click="show()">

<label class="onoffswitch-label" for="myonoffswitch"></label>

</div>

</div>

</div>

<div class="col-md-4">

Hill climbing search

</div>

</div>

<div class="row" id="canvasParent" >

<div class="col-md-4">

<div align="center">

<Button style="width:50%;" class="btn btn-primary btn-block" ng-click="reset(true)">Reset</Button>

</div>

<div align="center" >

<Button style="width:50%;" class="btn btn-primary" ng-click="getSolution()">

Get Solution

</Button>

</div>

</div>

<div class="col-md-4">

<canvas height="400" width="400" id="my-canvas" ng-click="onCanvasClick($event)"></canvas>

</div>

<div class="col-md-4">

<div align="left" ng-show="isHillClimbing">

<p>Iterations : {{iterations}}</p>

</div>

</div>

</div>

</div>

<div class="footer">

Developed By<br>Hrishikesh Deshpande

</div>

</body>

</html>

2)tsp.css

body{

color:white ;

}

.nav>li>a:focus,.nav>li>a:hover{

background-color: inherit ;

cursor: none ;

}

.footer {

position: fixed;

left: 0;

bottom: 0;

width: 100%;

background-color: #2c3e50;

color: white;

text-align: center;

font-size:20px;

}

.onoffswitch {

position: relative; width: 119px;

-webkit-user-select:none; -moz-user-select:none; -ms-user-select: none;

}

.onoffswitch-checkbox {

display: none;

}

.onoffswitch-label {

display: block; overflow: hidden; cursor: pointer;

height: 36px; padding: 0; line-height: 36px;

border: 2px solid #2c3e50; border-radius: 36px;

background-color: #95A5A6;

transition: background-color 0.2s ease-in;

}

.onoffswitch-label:before {

content: "";

display: block; width: 36px; margin: 0px;

background: #FCFCFC;

position: absolute; top: 0; bottom: 0;

right: 81px;

border: 2px solid #2c3e50; border-radius: 36px;

transition: all 0.2s ease-in 0s;

}

.onoffswitch-checkbox:checked + .onoffswitch-label {

background-color: #95A5A6;

}

.onoffswitch-checkbox:checked + .onoffswitch-label, .onoffswitch-checkbox:checked + .onoffswitch-label:before {

border-color: #95A5A6;

border: 2px solid #2c3e50;

}

.onoffswitch-checkbox:checked + .onoffswitch-label:before {

right: 0px;

}

.modal {

display: block; /\* Hidden by default \*/

position: fixed; /\* Stay in place \*/

z-index: 200; /\* Sit on top \*/

left: 0;

top: 0;

width: 100%; /\* Full width \*/

height: 100%; /\* Full height \*/

overflow: auto; /\* Enable scroll if needed \*/

background-color: rgb(0,0,0); /\* Fallback color \*/

background-color: rgba(0,0,0,0.8); /\* Black w/ opacity \*/

}

3)controller.js

//define the app to be a angular app and register it with angular framework with the required dependencies.

var app = angular.module('app',[]) ;

//Controller for the AngularJS starts here

app.controller('tspCtrl',function($scope,$window,$timeout){

//Model for toggle switch

$scope.isHillClimbing = false ;

$scope.canvas = document.getElementById('my-canvas') ;

$scope.canvasContext = $scope.canvas.getContext('2d') ;

console.log($scope.canvas) ;

$scope.heuristic = null ;

$scope.canvasContext.fillStyle = "#000000";

$scope.canvasContext.fillRect(0,0,400,400);

$scope.distanceMatrix = [] ;

$scope.loading = false ;

$scope.getLoading = function() {

return $scope.loading ;

}

$scope.iterations = 0 ;

//Draw a line on canvas. Take initial and final x and y co ordinates ;

$scope.drawLine = function(x1, y1, x2, y2) {

var ctx = $scope.canvasContext ;

ctx.fillStyle="#FFFFFF" ;

ctx.strokeStyle="#FFFFFF" ;

ctx.beginPath();

ctx.moveTo(x1, y1);

ctx.lineTo(x2, y2);

ctx.stroke();

}

//Reset the canvas and reset the cities and edges optionally based on the input parameter

$scope.reset = function(shoudlResetCities) {

$scope.distanceMatrix = [] ;

$scope.beenReset = true ;

if(shoudlResetCities) {

$scope.cities = [] ;

$scope.roads = [] ;

}

$scope.canvas = document.getElementById('my-canvas') ;

$scope.canvasContext = $scope.canvas.getContext('2d') ;

$scope.canvasContext.clearRect(0,0,400,400) ;

$scope.canvasContext.fillStyle = "#000000";

$scope.canvasContext.fillRect(0,0,400,400);

}

//draw circle with center at x,y and radius r

$scope.drawCircle=function(x,y,r) {

var ctx = $scope.canvasContext ;

ctx.beginPath();

ctx.arc(x,y,r,0,2\*Math.PI);

ctx.fillStyle = "#FFFFFF" ;

ctx.fill() ;

ctx.closePath() ;

ctx.stroke();

}

//fill text specified at position x,y

$scope.fillText = function(x,y,text) {

var context = $scope.canvasContext ;

context.moveTo(x,y) ;

var font = 10 +"px serif";

context.font = font;

context.textBaseline = "top";

//context.textAlign = "center"

context.fillStyle = 'red' ;

var t

if(text.toString().length == 1) {

t = '0'+text ;

}else{

t = text ;

}

context.fillText(t,x,y);

}

//draw an ellipse with center at x,y and specified width and height

$scope.drawEllipse = function(centerX, centerY, width, height) {

$scope.canvasContext.fillStyle = "#FFFFFF";

var context = $scope.canvasContext ;

context.beginPath();

context.moveTo(centerX, centerY - height/2); // A1

context.bezierCurveTo(

centerX + width/2, centerY - height/2, // C1

centerX + width/2, centerY + height/2, // C2

centerX, centerY + height/2); // A2

context.bezierCurveTo(

centerX - width/2, centerY + height/2, // C3

centerX - width/2, centerY - height/2, // C4

centerX, centerY - height/2); // A

context.fill();

context.closePath();

}

$scope.cities = [] ;

$scope.roads = [] ;

//calculate distance between points x1,y1 and x2,y2

$scope.distance = function(x1,y1,x2,y2) {

var xd = x2-x1 ;

var yd = y2-y1 ;

var zd = xd\*xd + yd\*yd ;

return Math.sqrt(zd) ;

}

//handler for a click on canvas , captures co-ordinates of the click and draws a city at that point and connects it to other cities already present on the canvas

$scope.onCanvasClick = function(event) {

$scope.beenReset = false ;

var c = new $scope.city(event.offsetX,event.offsetY,$scope.cities.length) ;

$scope.cities.push(c) ;

c.displayVertex() ;

var cities = $scope.cities ;

for(var i=0 ; i<cities.length ; i++) {

$scope.distanceMatrix[i] = [] ;

for(var j=0 ; j<cities.length ; j++) {

var v1 = $scope.cities[i] ;

var v2 = $scope.cities[j] ;

var d = $scope.distance(v1.x,v1.y,v2.x,v2.y) ;

if(i!=j) {

$scope.roads.push(new $scope.edge(v1,v2,Math.floor(d)))

}

$scope.distanceMatrix[i][j] = Math.floor(d) ;

}

}

for(var road=0 ;road<$scope.roads.length ; road++) {

$scope.roads[road].displayEdge() ;

}

for(var k=0 ; k<$scope.cities.length ; k++) {

$scope.cities[k].refillText() ;

}

}

//get the total route distance for the tour specified by the parameter

$scope.calculateTourDistance = function (route) {

var totalDistance = 0;

for (var i = 1; i < route.length; i++) {

totalDistance += Math.sqrt(Math.pow((route[i][0] - route[i-1][0]),2) + Math.pow((route[i][1] - route[i-1][1]),2));

}

//distance back to beginning

totalDistance += Math.sqrt(Math.pow((route[route.length-1][0] - route[0][0]),2) + Math.pow((route[route.length-1][1] - route[0][1]),2));

return totalDistance;

}

//find number possible of next states from the current tour as specified by the second parameter

$scope.generateRandomNextStates = function (tour, numNextStates) {

var newRoute = tour.slice(0);

for (var i = 0; i < numNextStates; i++) {

var route1 = Math.floor(Math.random() \* newRoute.length);

var route2 = Math.floor(Math.random() \* newRoute.length);

var temp = newRoute[route1];

newRoute[route1] = newRoute[route2];

newRoute[route2] = temp;

}

return newRoute;

}

//Generates the next possible states using the helper function and selects the best next state.

$scope.generateAndSelectNextState = function(route, numNextStates) {

var nextStates = [route];

for (var i = 0; i < numNextStates; i++) {

nextStates.push($scope.generateRandomNextStates(route, Math.ceil(i/2)));

}

var routeLengths = [];

for (var j = 0; j < nextStates.length; j++) {

routeLengths.push({

route: nextStates[j],

distance: $scope.calculateTourDistance(nextStates[j])

});

}

routeLengths.sort(function(a,b){

if (a.distance > b.distance) return 1;

if (a.distance < b.distance) return -1;

return 0;

});

return routeLengths[0].route;

}

//Draws the current tour on the canvas

$scope.refreshCanvas = function(coords, iteration) {

if($scope.beenReset) {

return ;

}

$scope.coords = coords ;

var canvas = document.getElementById("my-canvas"); //canvas, context, other vars etc

var canvasWidth = canvas.width;

var canvasHeight = canvas.height;

var ctx = canvas.getContext("2d");

ctx.clearRect(0, 0, canvas.width, canvas.height); //clear screen to redraw

ctx.fillStyle="#000000" ;

ctx.fillRect(0,0,400,400);

ctx.fillStyle="#FFFFFF" ;

var mult = 1 ;

for(var i=0 ; i<$scope.cities.length ; i++) {

$scope.cities[i].displayVertex() ;

}

for (var i = 0; i < coords.length; i++) {

ctx.beginPath();

if (i == coords.length-1) {

//last path

ctx.moveTo(coords[i][0]\*mult, coords[i][1]\*mult);

ctx.lineTo(coords[0][0]\*mult, coords[0][1]\*mult);

ctx.stroke();

}

else {

ctx.moveTo(coords[i][0]\*mult, coords[i][1]\*mult);

ctx.lineTo(coords[i+1][0]\*mult, coords[i+1][1]\*mult);

ctx.stroke();

}

$timeout(function () {

$scope.iterations = iteration ;

}, 500);

}

}

//initiates and controls the hill climbing search

$scope.hillClimbSearch = function(route, ctr, originalRoute) {

if($scope.isHillClimbing) {

if (typeof route === 'undefined') {

var route = [] ;

for(var i=0 ; i<$scope.cities.length ; i++) {

route.push([$scope.cities[i].x,$scope.cities[i].y]) ;

}

ctr = 0;

originalRoute = route.slice(0) ;

}

var newBest = $scope.generateAndSelectNextState(route, 50);

$scope.refreshCanvas(newBest, ctr);

if (ctr < 1000 && !$scope.beenReset) {

if (ctr > 0 && ctr % 1000 === 0) {

setTimeout(function() {

$scope.hillClimbSearch(originalRoute, ctr+1, originalRoute);

}, 0);

}

else {

setTimeout(function() {

$scope.hillClimbSearch(newBest, ctr+1, originalRoute);

}, 0);

}

}

else {

}

}

}

//prototype of city object

$scope.city = function(x,y,name){

this.name = name;

this.x=x;

this.y=y;

this.parent;

this.children = [];

this.asChildVisited = false;

this.refillText = function() {

$scope.fillText(this.x-5,this.y-5,this.name) ;

}

this.displayVertex = function(){

$scope.drawCircle(this.x, this.y,12);

$scope.fillText(this.x-5,this.y-5,this.name) ;

} ;

}

//city object ends

//prototype for edge connecting two cities vertex1 and vertex2

$scope.edge = function(vertex1,vertex2,distance){

this.vertex1 = vertex1;

this.vertex2 = vertex2;

this.distance = distance;

this.name = this.vertex1.name.toString()+"-"+this.vertex2.name.toString();

this.show = true;

this.color = 255;

this.displayEdge = function(){

if(this.show == true){

//stroke(this.color);

//strokeWeight(2);

$scope.drawLine(this.vertex1.x+5, this.vertex1.y+5, this.vertex2.x+5, this.vertex2.y+5);

}

}

}

//hide all the edges in the parameter specified

$scope.hideEdges = function(edges) {

for(var i=0;i<edges.length;i++){

edges[i].show = false;

}

}

//find the index of edge between vertex1 and vertex2

$scope.findEdgeIndex = function(vertex1,vertex2,edges){

for(var i=0;i<edges.length;i++){

if(edges[i].name == vertex1.name.toString()+"-"+vertex2.name.toString()){

return i;

}

}

}

//refresh the canvas and redraw the new state

$scope.redraw = function(vertices,edges) {

$scope.canvas = document.getElementById('my-canvas') ;

$scope.canvasContext = $scope.canvas.getContext('2d') ;

$scope.canvasContext.fillStyle = "#000000";

$scope.canvasContext.fillRect(0,0,400,400);

for(var i=0 ; i<vertices.length ; i++) {

vertices[i].displayVertex() ;

}

for(var i=0 ; i<edges.length ; i++) {

edges[i].displayEdge() ;

}

}

//generate all the edges between all the cities specified in the parameter

$scope.generateEdges = function(vertices) {

var distanceMatrix = [] ;

edges = [] ;

for(var i=0 ; i<vertices.length ; i++) {

distanceMatrix[i] = [] ;

for(var j=0 ; j<vertices.length ; j++) {

var v1 = vertices[i] ;

var v2 = vertices[j] ;

var d = $scope.distance(v1.x,v1.y,v2.x,v2.y) ;

if(i!=j) {

edges.push(new $scope.edge(v1,v2,Math.floor(d)))

}

distanceMatrix[i][j] = Math.floor(d) ;

}

}

return edges ;

}

//calculate the MST for the given vertices and edges

$scope.calculateMST = function(vertices,edges) {

if(edges === null || edges === undefined) {

edges = $scope.generateEdges(vertices) ;

}

$scope.hideEdges(edges);

var reached = [];

var unreached = [];

for (var i = 0; i < vertices.length; i++) {

unreached.push(vertices[i]);

}

reached.push(unreached[0]);

unreached.splice(0, 1);

while (unreached.length > 0) {

var record = 100000;

var rIndex;

var uIndex;

for (var i = 0; i < reached.length; i++) {

for (var j = 0; j < unreached.length; j++) {

var v1 = reached[i];

var v2 = unreached[j];

var d = Math.floor($scope.distance(v1.x, v1.y, v2.x, v2.y));

if (d < record) {

record = d;

rIndex = i;

uIndex = j;

}

}

}

var ans = $scope.findEdgeIndex(reached[rIndex],unreached[uIndex],edges);

edges[ans].show = true;

reached.push(unreached[uIndex]);

unreached.splice(uIndex, 1);

}

var d = 0 ;

for(var i=0 ; i<edges.length ; i++) {

if(edges[i].show === true) {

d+=edges[i].distance ;

}

}

console.log('mst distance ' + d) ;

console.log(edges) ;

console.log(reached) ;

console.log(unreached) ;

$scope.redraw(vertices,edges) ;

return d ;

}

$scope.Node = function (data, priority) {

this.data = data;

this.priority = priority;

}

$scope.Node.prototype.toString = function(){return this.priority}

// Implementation of priority queue

$scope.PriorityQueue = function(arr) {

this.heap = [];

if (arr) for (i=0; i< arr.length; i++)

this.push(arr[i].data, arr[i].priority);

}

$scope.PriorityQueue.prototype = {

push: function(data, priority) {

var node = new $scope.Node(data, priority);

//this.bubble(this.heap.push(node) -1);

this.heap.push(node) ;

this.sort(this.heap) ;

},

sort:function(a) {

for(var i=0 ; i<this.heap.length ; i++) {

var swapp;

var n = a.length-1;

var x=a;

do {

swapp = false;

for (var i=0; i < n; i++)

{

if (x[i].priority > x[i+1].priority)

{

var temp = x[i];

x[i] = x[i+1];

x[i+1] = temp;

swapp = true;

}

}

n--;

} while (swapp);

return x;

}

},

// removes and returns the data of highest priority

pop: function() {

var data = this.heap[0].data ;

this.heap.splice(0,1) ;

return data ;

//var topVal = this.heap[1].data;

//this.heap[1] = this.heap.pop();

//this.sink(1); return topVal;

},

// bubbles node i up the binary tree based on

// priority until heap conditions are restored

bubble: function(i) {

while (i > 1) {

var parentIndex = i >> 1; // <=> floor(i/2)

// if equal, no bubble (maintains insertion order)

if (!this.isHigherPriority(i, parentIndex)) break;

this.swap(i, parentIndex);

i = parentIndex;

} },

// does the opposite of the bubble() function

sink: function(i) {

while (i\*2 < this.heap.length) {

// if equal, left bubbles (maintains insertion order)

var leftHigher = !this.isHigherPriority(i\*2 +1, i\*2);

var childIndex = leftHigher? i\*2 : i\*2 +1;

// if equal, sink happens (maintains insertion order)

if (this.isHigherPriority(i,childIndex)) break;

this.swap(i, childIndex);

i = childIndex;

} },

// swaps the addresses of 2 nodes

swap: function(i,j) {

var temp = this.heap[i];

this.heap[i] = this.heap[j];

this.heap[j] = temp;

},

// returns true if node i is higher priority than j

isHigherPriority: function(i,j) {

return this.heap[i].priority < this.heap[j].priority;

},

//returns if queue is empty

isEmpty: function() {

return !this.heap.length > 0 ;

}

}

//implementation of tree

$scope.treeChild = function(value) {

this.value = value;

this.children = [];

this.parent = null;

this.setParentNode = function(node) {

this.parent = node;

}

this.getParentNode = function() {

return this.parent;

}

this.addChild = function(node) {

node.setParentNode(this);

this.children[this.children.length] = node;

}

this.getChildren = function() {

return this.children;

}

this.removeChildren = function() {

this.children = [];

}

this.getChildAt = function(index) {

return this.children[index] ;

}

this.getChildCount = function() {

return this.children.length ;

}

}

//Implementation of prototype that represents a state in the state space

$scope.stateNode = function(city,pathCost,heuristic) {

this.city = city ;

this.pathCost = pathCost ;

this.heuristic = heuristic ;

this.totalCost = this.pathCost + this.heuristic ;

this.setTotalCost = function() {

this.totalCost = this.pathCost + this.heuristic ;

}

this.setCustomTotalCost = function(pathCost,heuristic) {

this.totalCost = pathCost + heuristic ;

}

}

//find index of node provided as first parameter in the list provided as the second parameter

$scope.getVisitedIndex = function(currentNode,visited) {

for(var i=0 ; i<visited.length ; i++) {

if(visited[i].value.city.name === currentNode.value.city.name) {

return i ;

}

}

return -1 ;

}

//find index of city provided as first parameter in the list provided as the second parameter

$scope.getCityIndexFromVisitedCityNodes = function(city,cities) {

for(var i=0 ; i<cities.length ; i++) {

if(city.name === cities[i].value.city.name) {

return i ;

}

}

return -1 ;

}

//generate next possible states in the A star algorithm and attach it as children to the currentNode

$scope.getNextState = function(currentNode,visitedCities,cities) {

var citiesCopy = [] ;

for(var i=0 ; i<cities.length ; i++) {

var index = $scope.getCityIndexFromVisitedCityNodes(cities[i],visitedCities) ;

if(index === -1) {

citiesCopy.push(cities[i]) ;

}

}

//cities.splice($scope.getVisitedIndex(currentNode,cities),1) ;

console.log(citiesCopy) ;

for(var i=0 ; i<citiesCopy.length ; i++) {

var newStateNode = new $scope.stateNode(citiesCopy[i],null,$scope.calculateMST(citiesCopy,null)) ;

var newNode = new $scope.treeChild(newStateNode) ;

newNode.value.pathCost = $scope.getPathCostToNode(newNode,visitedCities) ;

newNode.value.totalCost = newNode.pathCost + newNode.heuristic ;

currentNode.addChild(newNode) ;

}

console.log(currentNode) ;

}

//calculate path cost of the current node from the root node in the order of tour taken

$scope.getPathCostToNode = function(childNode,visited) {

var pathCost = 0 ;

var prevNode = visited[0] ;

for(var i=1 ; i<visited.length ; i++) {

var x1 = visited[i].value.city.x ;

var y1 = visited[i].value.city.y ;

pathCost += $scope.distance(x1,y1,prevNode.value.city.x,prevNode.value.city.y) ;

prevNode = visited[i] ;

}

return (pathCost + $scope.distance(childNode.value.city.x,childNode.value.city.y,prevNode.value.city.x,prevNode.value.city.y)) ;

}

$scope.depth = 0 ;

$scope.nodesGenerated = 1 ;

//checks if the current state is goal state

$scope.isGoalState = function(visited) {

if(visited.length === $scope.cities.length) {

var contains = true ;

for(var i=0 ; i<$scope.cities.length ; i++) {

for(var j=0 ; j<visited.length;j++) {

if($scope.cities[i].name === visited[j].value.city.name)

{

contains=true ;

break ;

}

contains=false ;

}

if(!contains) {

return contains ;

}

}

return true ;

}else{

return false ;

}

}

//implements the state space search with A star algorithm

$scope.searchWithAstarAlgorithm = function() {

$scope.visited = [] ;

$scope.nodes = new $scope.PriorityQueue() ;

$scope.loading = true ;

var initNode = new $scope.stateNode($scope.cities[0],0,$scope.calculateMST($scope.cities,null)) ;

var stateSpace = new $scope.treeChild(initNode) ;

var finalCityIndex = $scope.cities[$scope.cities.length-1] ;

var visitedFirstCity = true ;

//check if the length is 0 ;

$scope.nodes.push(stateSpace,initNode.totalCost) ;

//while there are nodes in priority queue , search the state space

while(!$scope.nodes.isEmpty()) {

var currentNode = $scope.nodes.pop() ;

$scope.visited.push(currentNode) ;

//if goal state is reached , stop the search and display solution

if($scope.isGoalState($scope.visited)) {

$scope.loading = false ;

alert('Goal State Reached') ;

var cityCopy = [] ;

for(var i=0 ; i<$scope.visited.length ; i++) {

var city = $scope.visited[i].value.city ;

cityCopy.push(city) ;

}

$scope.reset() ;

console.log(cityCopy) ;

//draw the solution

for(var i=0 ; i<cityCopy.length-1 ; i++) {

cityCopy[i].displayVertex() ; $scope.drawLine(cityCopy[i].x,cityCopy[i].y,cityCopy[i+1].x,cityCopy[i+1].y) ;

}

var i = cityCopy.length-1 ; $scope.drawLine(cityCopy[i].x,cityCopy[i].y,cityCopy[0].x,cityCopy[0].y) ;

cityCopy[i].displayVertex() ;

return ;

//if goal state is not reached then explore the state space by getting the children from current node

}else{

$scope.getNextState(currentNode,$scope.visited,$scope.cities) ;

for(var i=0 ; i<currentNode.getChildCount() ; i++ ) {

var childNode = currentNode.getChildAt(i) ;

var visitedIndex = $scope.getVisitedIndex(childNode,$scope.visited) ;

if(visitedIndex === -1) {

//if child was never visited before

childNode.value.pathCost = $scope.getPathCostToNode(childNode,$scope.visited) ;

childNode.value.setTotalCost() ;

$scope.nodes.push(childNode,childNode.value.totalCost) ;

}else{

//if the child was visited before then check was it a better solution

/\*if($scope.visited[visitedIndex].value.pathCost

< childNode.value.pathCost) {

//if visited child is better next state than others

var v = $scope.visited.splice(visitedIndex,1) ;

$scope.nodes.push(v,v.value.totalCost) ;

}\*/

}

}

}

}

$scope.loading = false ;

alert('No solution found!')

//reset canvas for a new problem instance

$scope.reset(true) ;

}

//Determines whether the solution is to be found using A star or hill climb based on the user input

$scope.getSolution = function() {

if($scope.isHillClimbing) {

$scope.hillClimbSearch() ;

}else{

$scope.searchWithAstarAlgorithm() ;

}

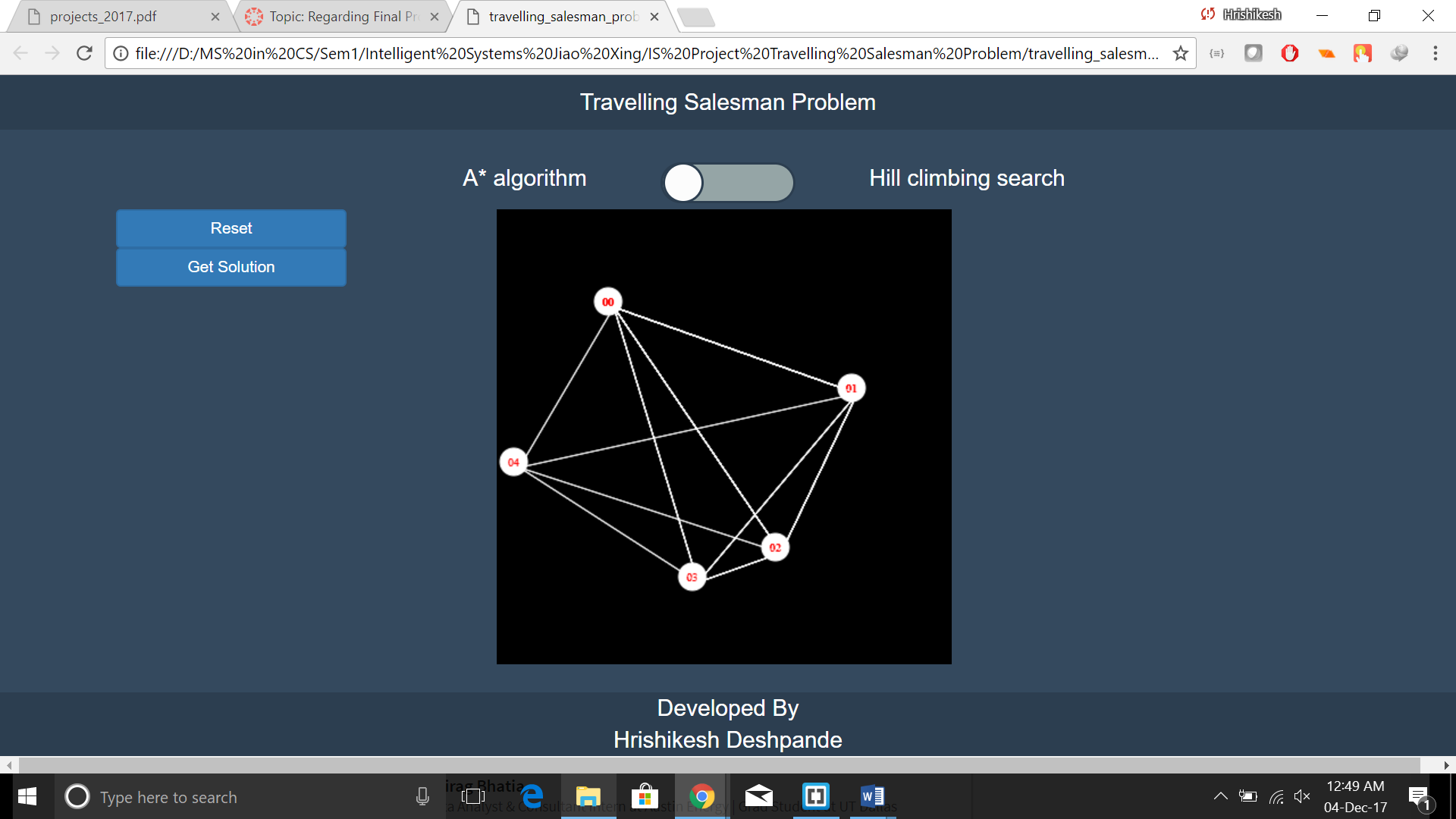
}

}) ;

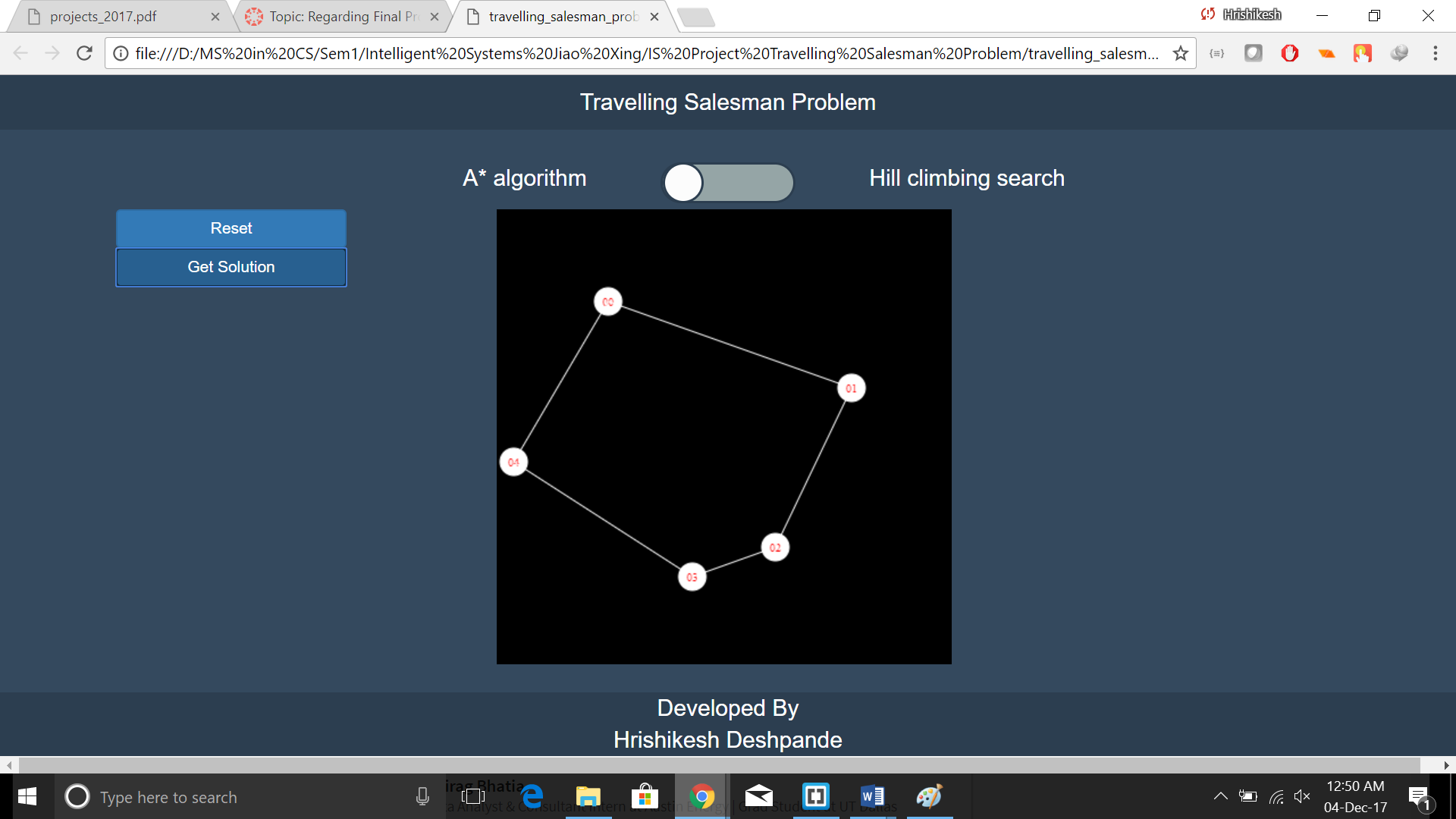
**Examples for testing:**

1)For A \* -

Click on the points specified in the screenshots below

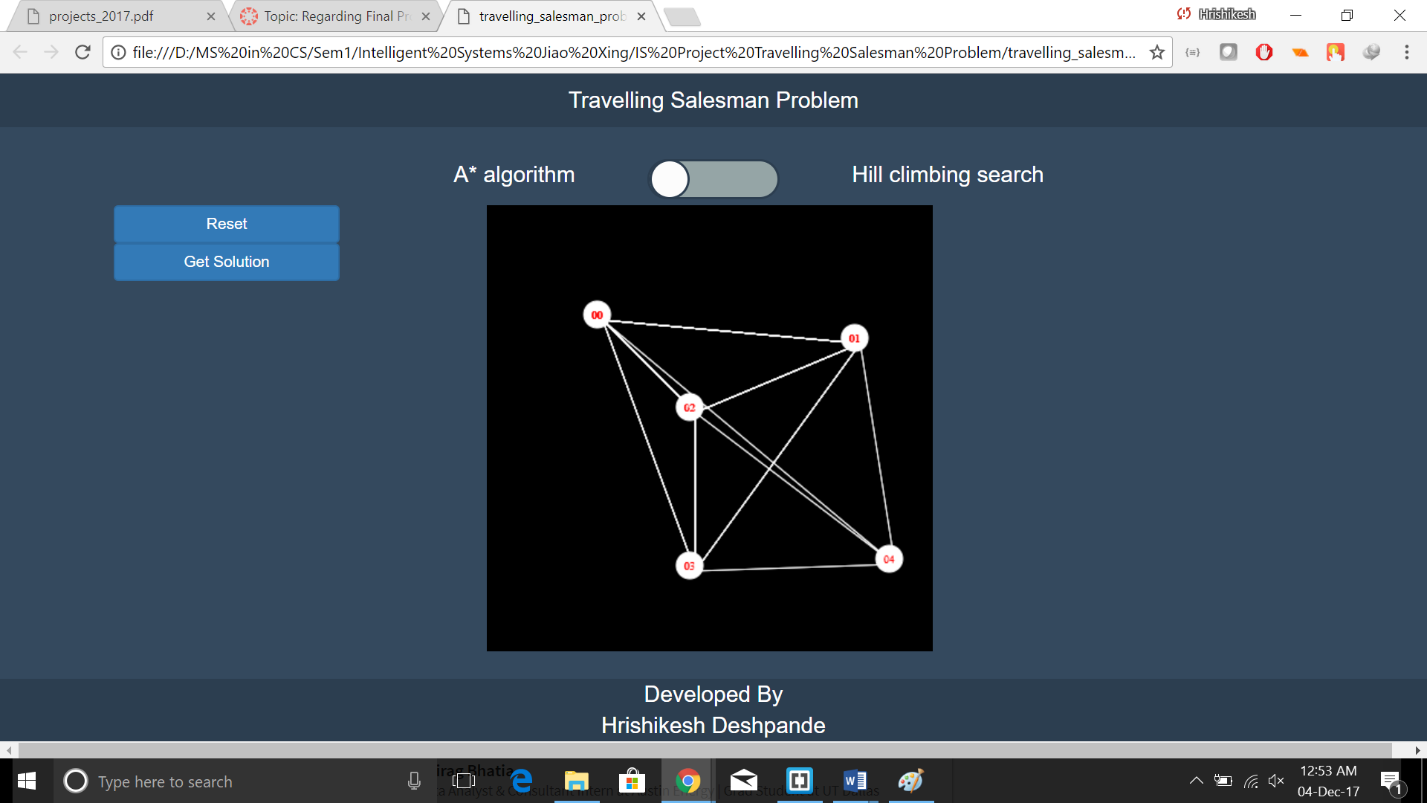


Click get solution button

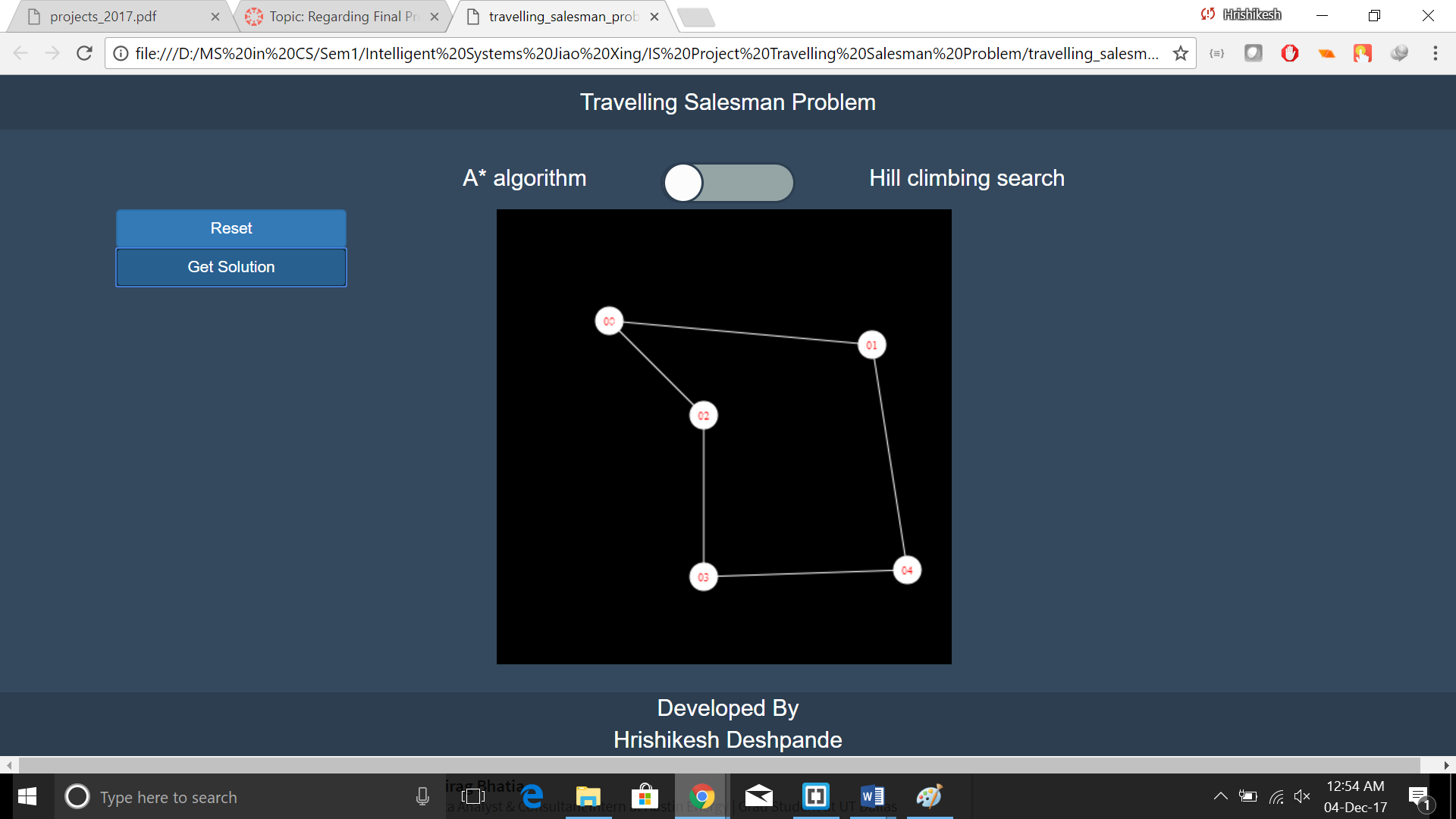


Click on reset to clear canvas

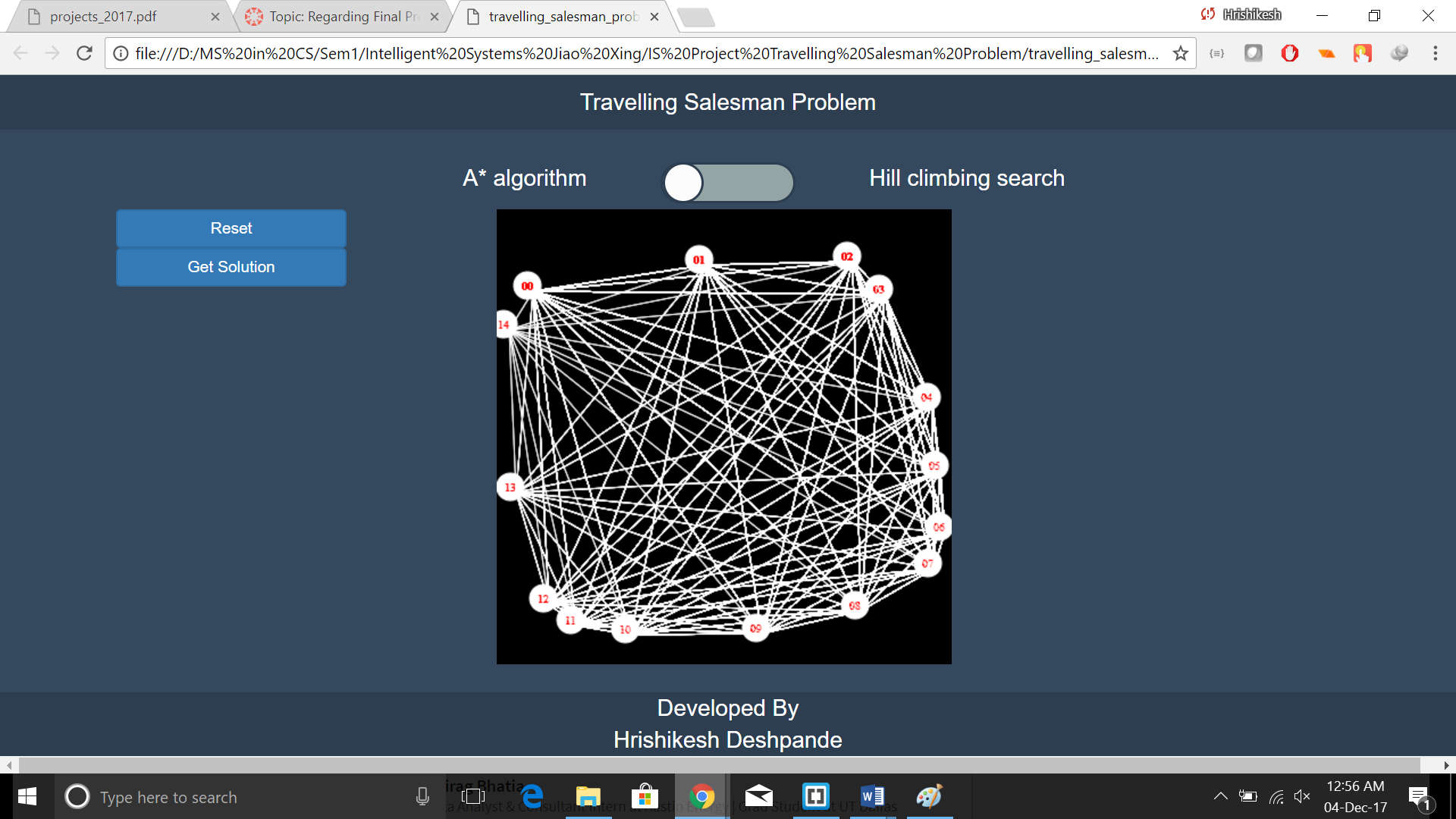
More examples for testing

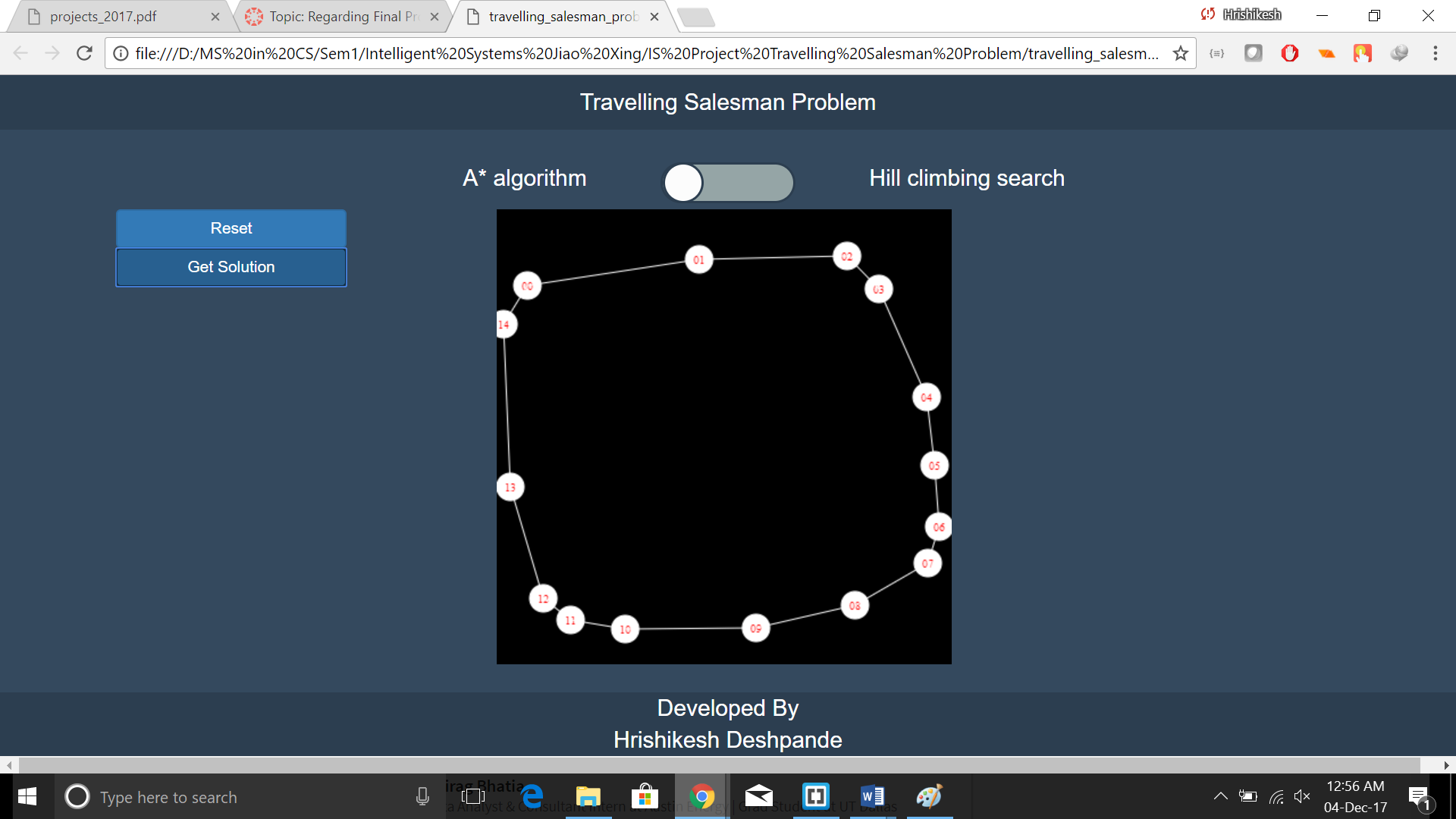


Solution



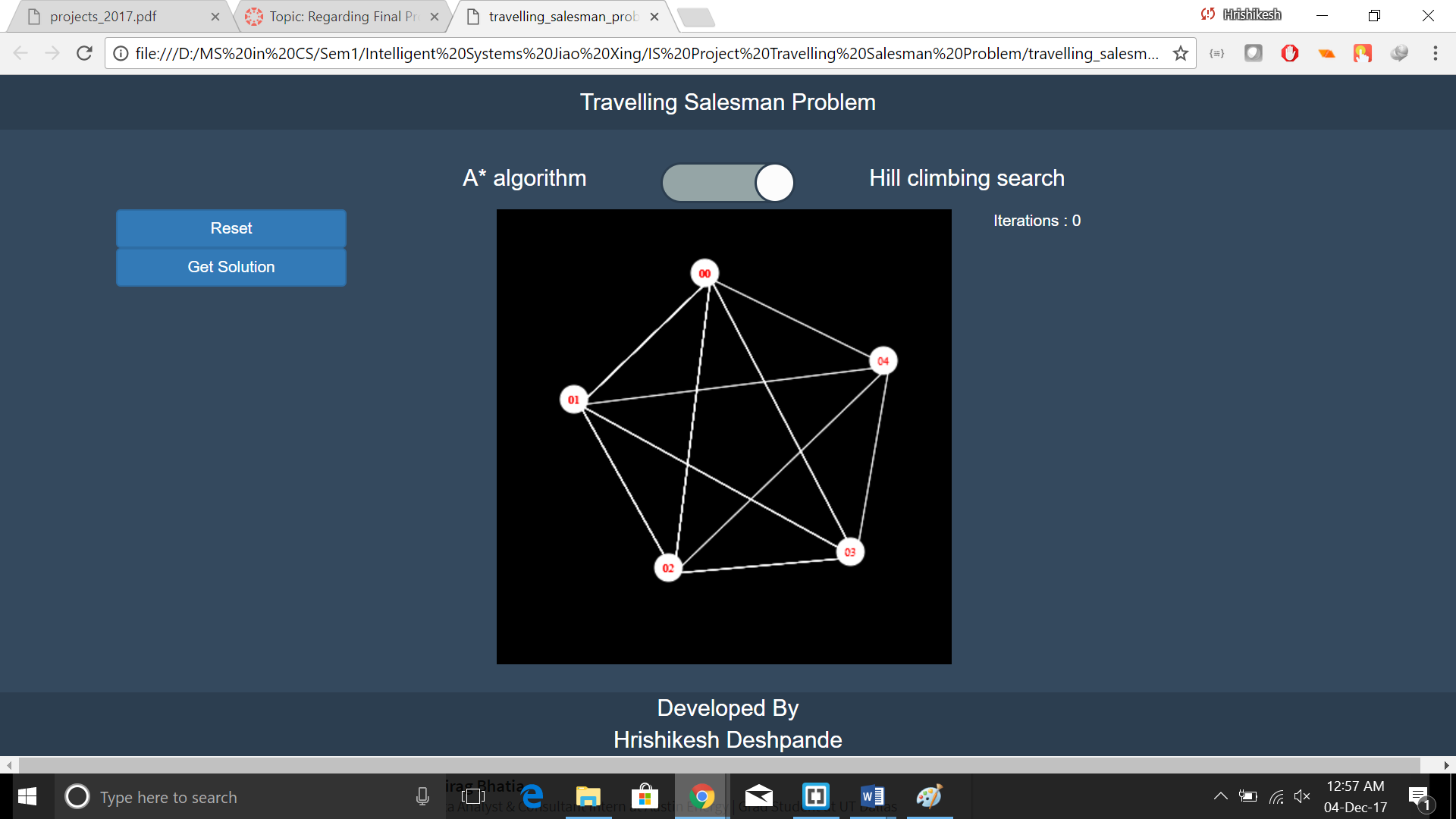
Another sample



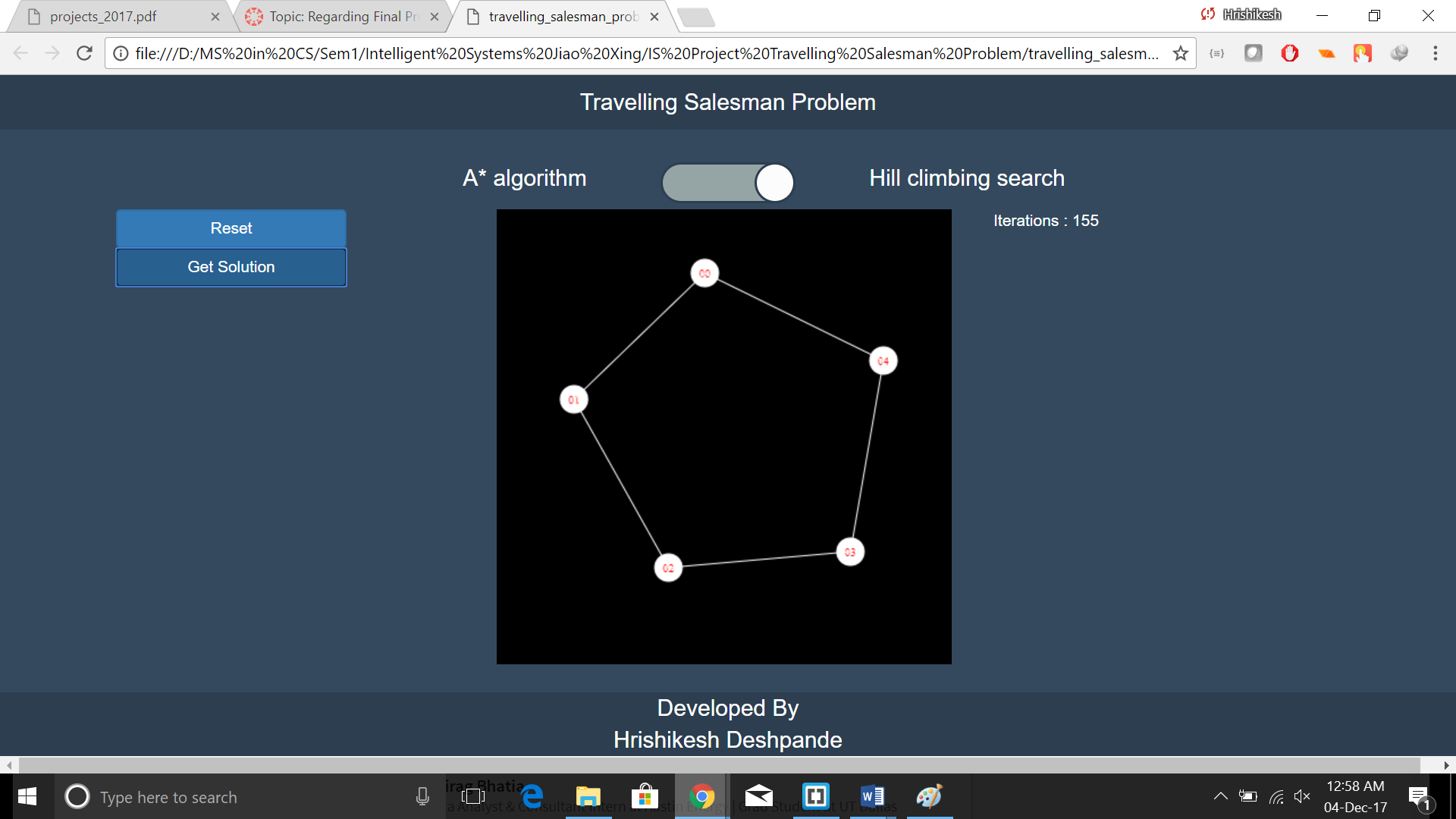


2)For Hill climbing

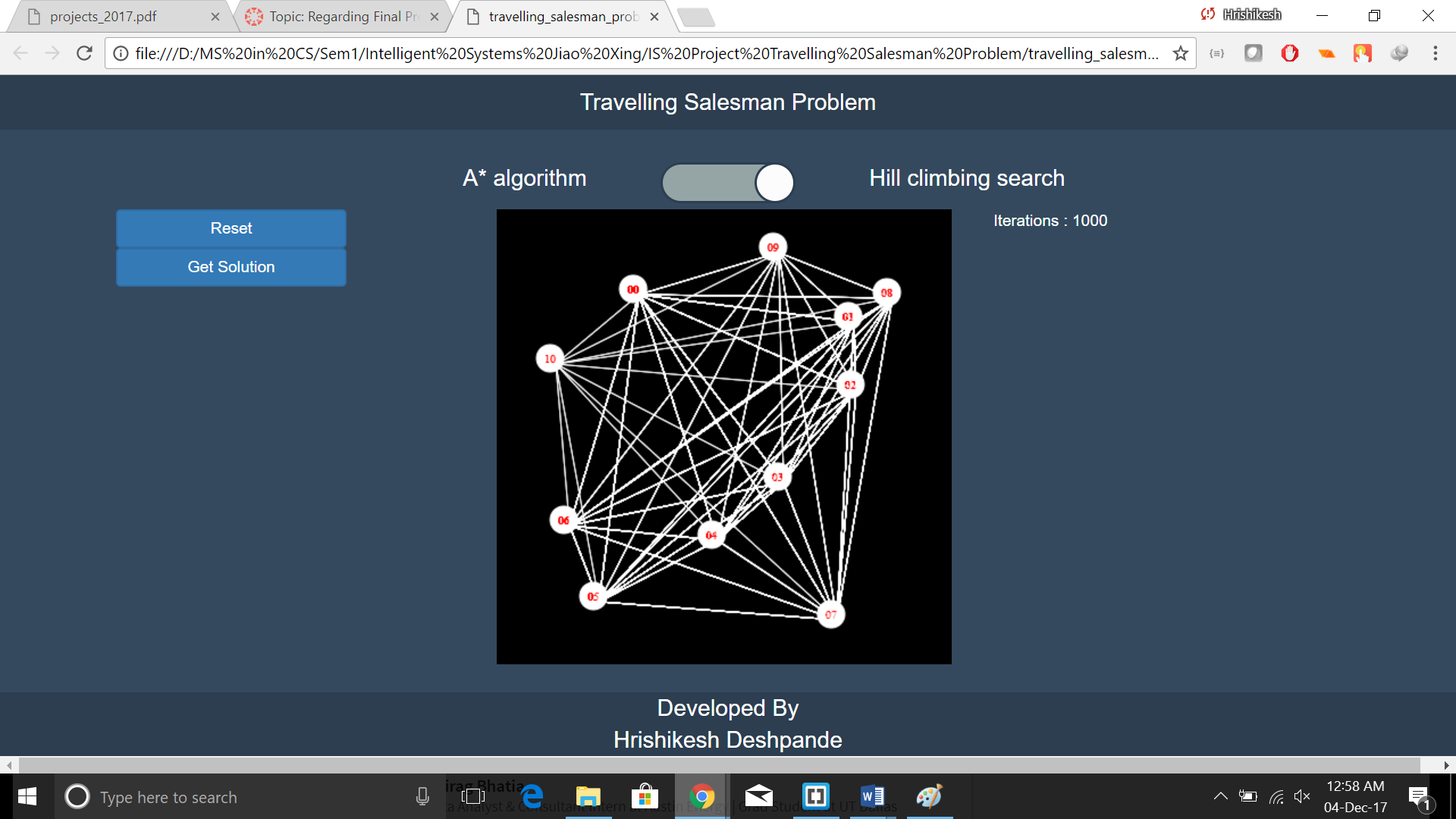
Click on the toggle switch to change to hill climbing mode

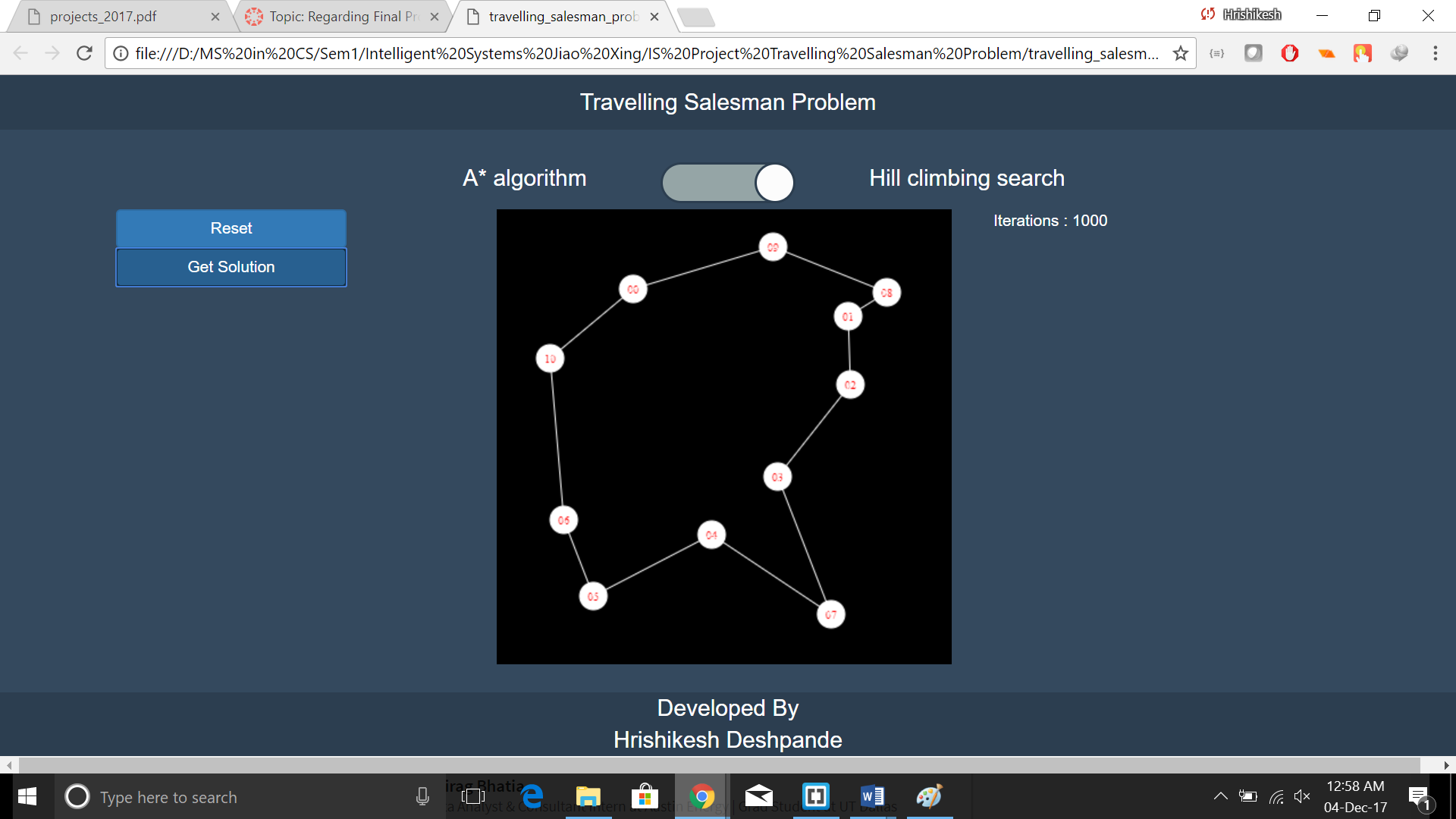


Click on get solution



Another sample





**How to run the program**

1) Unzip the file in any desired folder

2) If the default browser in your system is chrome and file association for html files is the browser , then just double-click the travelling\_salesmen\_problem.html file which will open it in your browser.

3)If html files are not associated with browser, then right click on travelling\_salesmen\_problem.html file and hover over open with and select google chrome from the dropdown menu.

4)If you don’t have chrome, you can use internet explorer as well but support for mozilla firefox. is not guaranteed as some of the features used for the user interface are not supported in some versions of mozilla firefox.

5) In Internet explorer, it may prompt you that ActiveX controles have been blocked in which case, please unblock them as they do not allow to draw canvas.

6) If no supported browser is available in your system, please install google chrome from <https://www.google.com/chrome/browser/desktop/index.html>. As the installation file is too big, a link has been provided here.

7) No batch file is needed as the html file itself acts as the executable for the program.

8)Once the program starts running, the user can click anywhere on the canvas to generate a city at the clicked location. On clicking on the canvas again, another city will be generated and automatically connected to all the cities already present on the canvas.

9) The toggle switch just above the canvas can be used to select the search strategy to be used for solving the problem.

10)Once user is done generating the cities, press on the get solution button to solve the problem and get the solution on canvas itself.

11)In case of hill climbing approach, the program will display the iterations it is performing near the right hand top corner of canvas.

12)The changed route is also animated on the canvas. This is generally visible in case of large number of cities. In case of small number of cities, the search is pretty fast and solution is reached before anything can be animated.

**Comparison of optimal solution found with A\* and hill climbing**

TSP is a problem in which the final state is not known. Hence, in many scenarios, hill climbing may not reach the shortest tour. It may be close to the shortest one. The number of allowed iterations can be increased yet it does not guarantee that it will reach the shortest tour. There is always a chance that it may get stuck in local maxima. On the contrary, A\* algorithm always reaches the shortest tour. Since we visit each city one by one, unlike hill climbing where we consecutively try shorter and shorter routes, and the search is guided by the path cost and the heuristic. Also, in A\* there is distinct criteria that the goal state has been reached but there are no such criteria in hill climbing. We can keep track of the shortest tour in the past but, yet we cannot predict that there might not be a shorter route unless all possibilities are exhausted which in case of large number of cities is impossible.

Conclusion – For small number of cities, both reach shortest tour but for large number of cities the hill climbing may not reach the optimal solution i.e. the shortest tour but A\* always reaches the shortest tour.

**Possible future improvements and extensions –**

1) A history of tours for hill climbing can be maintained and some algorithm that may guess that hill climbing has reached the goal state with some high probability can be implemented.

2) Visualization for A\* state space search can be implemented.

3)The number of iterations can be input by user.

4)More sophisticated version of tree and priority queue may be used/implemented to make the search faster and efficient.

**References –**

1)Wikipedia – for information about minimum spanning tree

2)Youtube – for information about minimum spanning tree and graphs

3)Stackoverflow – for bootstrap, AngularJS, JavaScript canvas

4)Codepen – for bootstrap, AngularJS, JavaScript canvas

5)Bootstrap documentation – for bootstrap

6)AngularJS documentation-for AngularJS