Technical solution

> solution
>
> > algorithms
> >
> > > aco.py
> > >
> > > sa.py
> > >
> > > held__karp.py
> > >
> > > common.py
> >
> > static
> >
> > > bootstrap/...
> > >
> > > css
> > >
> > > > main.scss
> > > >
> > > > start.scss

- js
  - display-output.js
  - graph-manipulation.js
- images/...
- templates
  - main.html
  - start.html
- app.py

app.py

app.py is a program that acts like a bridge between back-end and front-end

It uses flask framework to create navigation between start page and main page, so that "website_url/" returns the start page ("start.html"), and "website_url/main" returns the main page ("main.html").

```
from flask import Flask, render_template, jsonify, request
import json

#import python files with algorithms
import sys
sys.path.append("algorithms")

from algorithms.common import *
from algorithms.aco import *
from algorithms.sa import *
from algorithms.held_karp import *

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('start.html')

@app.route('/main')
def main():
    return render_template('main.html')
```

It also creates a route /calculate_outputs that is a url where requests will be sent from front-end to back-end functions (OBJECTIVE 6)

```
@app.route('/calculate_outputs', methods=['POST'])
def calculate_outputs_api():
    #get the data in the request
    data = request.get_json()
    #match the fields
    coordinates = data['coordinates']
    aco_a = data['aco_a']
    aco_b = data['aco_b']
    aco_Q = data['aco_Q']
    aco_er = data['aco_er']
    aco_ants = data['aco_ants']
    aco_iter = data['aco_iter']
    aco_shake = data['aco_shake']
    sa_a = data['sa_a']
    sa_T = data['sa_T']
    sa_iter = data['sa_iter']
    #add the distance matrix
    n, dist = calculate_distance_matrix(coordinates)
    #create a tsp input object
    tsp_input = TSP_input(n, dist, coordinates)
    #create aco parameters object
    aco_input = ACO_parameters(aco_a, aco_b, aco_Q, aco_er, aco_ants, aco_iter,
aco_shake)
    #create sa parameters object
    sa_input = SA_parameters(sa_a, sa_T, sa_iter)

    #pass the inputs to corresponding functions
    hk_output = held_karp(tsp_input)
    aco_it_found, aco_output = solve_aco(tsp_input, aco_input)
    sa_it_found, sa_output = solve_sa(tsp_input, sa_input)

    #jsonify the held-karp output to be able to return it
    hk_output = json.dumps(vars(hk_output))

    #jsonify iterations of aco output
    for i in range(len(aco_output)):
        aco_output[i] = json.dumps(aco_output[i].__dict__)
    #jsonify iterarations of sa output
    for i in range(len(sa_output)):
        sa_output[i] = json.dumps(sa_output[i].__dict__)

    #return a json with all outputs
    return jsonify({
        'hk_output': hk_output,
        'aco_it_found': aco_it_found,
        'aco_output': aco_output,
        'sa_it_found': sa_it_found,
        'sa_output': sa_output
    })
```

```
if __name__ == '__main__':
    app.run(debug=True)
```

Templates

Templates are html files, and I have 2 of those main.html and start.html

start.html

Start.html is a simple web-page, that contains 2 headings, an image and a button. In the head section I also have conected a css file and a bootstrap framework.

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>tsp</title>
    <link rel="stylesheet" href="../static/css/start.min.css">
    <script src="../static/bootstrap/js/bootstrap.min.js"></script>
  </head>
  <body>
    <p class="text-center fs-1 fw-bold hm">travelling salesman problem</p>
    <p class="text-center fs-4 fw-light">ant colony optimization <b>vs</b>
simulated annealing</p>
    <img src = "{{ url_for('static', filename='images/start_page.jpg') }}"
class="img-fluid" alt="image">
    <div class="text-center">
      <a href="{{ url_for('main') }}"><button type="button" class="btn btn-outline-
primary">start</button></a>
    </div>
  </body>
</html>
```

main.html

Main page has a bit more complex design, so I will split it into a few sections. Here's the general structure of html file. Here I connected css files and js files in header, and the body consists of 2 main parts:

- input div - where users input parameters

- graph-output-div - where visualisation and algorithms outputs are displayed

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>tsp</title>

    <link rel="stylesheet" href="../static/css/main.min.css">
    <script src="../static/js/graph-manipulation.js"></script>
    <script src="../static/js/display-output.js"></script>
    <script src="../static/bootstrap/js/bootstrap.min.js"></script>
  </head>
  <body>
    <p class="text-center fs-3 fw-bold hm">travelling salesman problem</p>
    <div class="input-graph-div div-flex">
      <div class="input">
        ...
      </div>
      <div class="graph-output-div">
        ...
      </div>
    </div>
  </body>
</html>
```

Below is the html code for the input div. The container contans all of the parameters that user can input, discussed in documented design and analysis.

```
<div class="input">
        <div class="cities div-flex">
          <label for="cities-input" class="form-label">cities</label>
          <input class="form-control form-control-sm" type="text" id="cities-input"
value="20">
          <button type="button" class="btn btn-outline-primary btn-sm" id="btn-
generate">generate</button>
          <button type="button" class="btn btn-outline-primary btn-sm btn-clear"
id="btn-clear">clear</button>
        </div>
        <div class = "speed div-flex">
          <label for="speed-input" class="form-label">speed</label>
          <input type="range" class="form-range" id="speed-input">
        </div>
        <div class = "radio">
          <div class="form-check">
            <input class="form-check-input" type="radio" name="radio" id="show-
iterations" checked>
            <label class="form-check-label" for="show-iterations">
              show iterations
            </label>
          </div>
```

```
          <div class="form-check">
            <input class="form-check-input" type="radio" name="radio" id="fast-
forward">
            <label class="form-check-label" for="fast-forward">
              fast forward
            </label>
          </div>
        </div>
        <div class="start-stop">
          <button type="button" class="btn btn-primary btn-sm btn-
start">start</button>
          <button type="button" class="btn btn-primary btn-sm btn-
stop">stop</button>
          <button type="button" class="btn btn-primary btn-sm btn-clear-
paths">clear paths</button>
        </div>
        <div class="accordion accordition-params">
          <div class="accordion-item">
            <h2 class="accordion-header">
              <button class="accordion-button collapsed" type="button" data-bs-
toggle="collapse" data-bs-target="#parameters-aco" aria-expanded="false" aria-
controls="parameters-aco">
                ant colony optimisation
              </button>
            </h2>
            <div id="parameters-aco" class="accordion-collapse collapse aco-color-
div">
              <div class="accordion-body">
                <div class="iterations-aco div-flex">
                  <label for="iterations-aco-input" class="form-
label">iterations</label>
                  <input class="form-control form-control-sm" type="text"
id="iterations-aco-input" value="20">
                </div>
                <div class="ants div-flex">
                  <label for="ants-input" class="form-label">ants</label>
                  <input class="form-control form-control-sm" type="text" id="ants-
input" value="20">
                  <label for="Q-input" class="form-label">Q</label>
                  <input class="form-control form-control-sm" type="text" id="Q-
input" value="100">
                </div>
                <div class="alpha-beta div-flex">
                  <label for="alpha-input" class="form-label">alpha</label>
                  <input class="form-control form-control-sm" type="text"
id="alpha-input" value="2">
                  <label for="beta-input" class="form-label">beta</label>
                  <input class="form-control form-control-sm" type="text" id="beta-
input" value="3">
                </div>
                <div class="e-rate div-flex">
                  <label for="e-rate-input" class="form-
label">evaporation_rate</label>
                  <input class="form-control form-control-sm" type="text" id="e-
rate-input" value="0.5">
                </div>
              </div>
```

```
            </div>
          </div>
          <div class="accordion-item">
            <h2 class="accordion-header">
              <button class="accordion-button collapsed" type="button" data-bs-
toggle="collapse" data-bs-target="#parameters-sa" aria-expanded="false" aria-
controls="parameters-sa">
                simulated annealing
              </button>
            </h2>
            <div id="parameters-sa" class="accordion-collapse collapse sa-color-
div">
              <div class="accordion-body">
                <div class="iterations-sa div-flex">
                  <label for="iterations-sa-input" class="form-
label">iterations</label>
                  <input class="form-control form-control-sm" type="text"
id="iterations-sa-input" value="1000">
                </div>
                <div class="temperarture div-flex">
                  <label for="t-input" class="form-label">T</label>
                  <input class="form-control form-control-sm" type="text" id="t-
input" value="1000">

                  <label for="ed-input" class="form-label">alpha</label>
                  <input class="form-control form-control-sm" type="text" id="ed-
input" value="0.94">
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>
</div>
```

Below is the code for the graph-output-div, which contains the graph div, where users can add/edit/delete nodes, and a text output.

```
<div class="graph-output-div">
        <div class="graph-div" id="graph-div">
          <p></p>
        </div>
        <div class="output-div div-flex">
          <div class="iteration-output-div">
            <p class="fs-7 fw-bold" id="iteration-text">Iteration: <span
id="iteration-number">0</span></p>
          </div>
          <div class="aco-output-div">
            <p class="fs-7 fw-bold">Ant Colony Optimisation: <span id="aco-
value">0</span></p>
            <p>found: <span id="aco-found">0</span></p>
            <div class="form-check hide">
              <input class="form-check-input hide" type="checkbox" value=""
```

```
id="aco-hide">
              <label class="form-check-label hide" for="aco-hide">
                hide
              </label>
            </div>
          </div>
          <div class="sa-output-div">
            <p class="fs-7 fw-bold">Simulated Annealing: <span id="sa-
value">0</span></p>
            <p>found: <span id="sa-found">0</span></p>
            <div class="form-check hide">
              <input class="form-check-input hide" type="checkbox" value="" id="sa-
hide">
              <label class="form-check-label hide" for="sa-hide">
                hide
              </label>
            </div>
          </div>
          <div class="hk-output-div">
            <p class="fs-7 fw-bold">Held-Karp: <span id="hk-value">0</span></p>
            <div class="form-check hide">
              <input class="form-check-input hide" type="checkbox" value="" id="hk-
hide">
              <label class="form-check-label hide" for="hk-hide">
                hide
              </label>
            </div>
          </div>
        </div>
      </div>
</div>
```

Style (scss)

I have a scss file for each of the templates, and all they do - make the website look pretty

start.scss

```
$body-bg: #11001E;
$body-color: #7511a3;
$primary: #A600F3;

@import "../node_modules/bootstrap/scss/bootstrap";

.hm{
    margin-top: $spacer*1.5 !important;
    margin-bottom: $spacer*.25 !important;
```

```
    color: #A600F3 !important;
}
```

main.scss

```
$body-bg: #11001E;
$body-color: #7511a3;
$primary: #A600F3;
$aco-color: #c3448e;
$sa-color: #d79840;
$hk-color: #2858b9;
$grey: #6e6e6e;

$form-range-track-width: 50%;
$accordion-border-color: $body-bg;
$accordion-button-active-bg: $body-bg;
$accordion-button-active-color: #d680fd;
$accordion-padding-x: 0;

@import "../node_modules/bootstrap/scss/bootstrap";

.hm{
    margin-top: $spacer*1.5 !important;
    margin-bottom: $spacer*.25 !important;
}
.form-range{
    margin-left: $spacer*.8;
}
.cities{
    margin-bottom: $spacer*1;
}
.form-control{
    margin-left: $spacer*.8;
    margin-right: $spacer*1.2;
}
.input{
    margin-left: $spacer*2;
    width: 25%;
}
.radio{
    margin-top: $spacer*1;
}
.start-stop{
    margin-top: $spacer*1;
}
.btn-stop{
    margin-left: $spacer*.8;
    margin-right: $spacer*.8;
}
.accordition-params{
    margin-top: $spacer*.8;
}
```

```
.div-flex{
    display: flex !important;
    margin-top: $spacer;
}
.btn-clear{
    margin-left: $spacer;
}
.graph-output-div{
    margin-left: $spacer*2;
    margin-right: $spacer*2;
    width: 100%;
}
.graph-div{
    background-color: $body-bg;
    height: $spacer*35;
    margin-top: $spacer;
    margin-left: $spacer;
    margin-right: $spacer;
    margin-bottom: $spacer*1.5;
    box-shadow: inset 0 0 10px;
}
.output-div{
    margin-top: $spacer;
    justify-content: space-between;
    margin-left: $spacer;
    margin-right: $spacer*3;
}
.aco-output-div{
    color: $aco-color;
}
.sa-output-div{
    color: $sa-color;
}
.hk-output-div{
    color: $hk-color;
}
.hide{
    color: $grey;
}
.aco-color-div{
    color: $aco-color;
}
.sa-color-div{
    color: $sa-color;
}
.node{
    border-radius: 50%;
    width: $spacer*.7;
    height: $spacer*.7;
    position: absolute;
    background-color: $primary;
    cursor: grab;
    z-index: 3;
}
.arc{
    position:absolute;
    height: $spacer*.1;
```

```
}
.aco-path{
    background-color: $aco-color;
    z-index: 1;
}
.sa-path{
    background-color: $sa-color;
    z-index: 1;
}
.hk-path{
    background-color: $hk-color;
    z-index: 2;
}
```

JS

JS files manage the user interaction with the web-site, such as creating nodes, starting a visulisation, etc, and also sends requests to the flask service to get data for the visualisation

graph-manipulation.js

graph-manipulation.js allows the users to create nodes, move them, delete them, also generate some amount of nodes and clear the graph (OBJECTIVE 5.8)

```
let coordinates = []; //array to hold coordinates of all nodes created

//a function that checks whether there is a node close to position of mouse
//this is to be able to move a node when a mouse is clicked on it
function close_to(x, y){
    let n = coordinates.length;
    //iterate all nodes
    for(let i=0;i<n;i++){
        let c_x = coordinates[i][0];
        let c_y = coordinates[i][1];
        //if horizontal and vertical distance is less than 10
        if(Math.abs(c_x - x) <= 10 && Math.abs(c_y-y) <= 10){
            coordinates.splice(i, 1); //delete the node from the array (because it
will be moved in the future)
            return [c_x, c_y]; //and return the actual coordinates of the node
        }
    }
    return [-1,-1]; //if nothing is found, return [-1,-1]
}

//clear all nodes from the graph
```

```
function clear_graph(graph_div){
    while(graph_div.firstChild){ //while there is child, remove it
        graph_div.removeChild(graph_div.firstChild);
    }
    coordinates.splice(0, coordinates.length); //clear the coordinates array so
that it stays in sync
}

//get the diameter of node that is specified in css file
function get_node_width(graph_div){
    //create a new 'artificial' node and add a class 'node' to it
    const node = document.createElement('div');
    node.classList.add('node');
    graph_div.appendChild(node);
    //get the style of the node
    const nodeStyle = window.getComputedStyle(node);
    //get the width (that is also a diameter)
    const nodeWidth = parseFloat(nodeStyle.width);
    //remove the artificial node
    graph_div.removeChild(node);
    return nodeWidth;
}

//a function to check whether the position of mouse is within the graph container
function isWithinGraphDiv(graph_div, x, y){
    //get the diameter of nodes
    nodeWidth = get_node_width(graph_div);
    //get the position of the graph container
    const containerRect = graph_div.getBoundingClientRect();
    //to be within container, coordinate has to be at least a node diameter from
its bounds
    //it is so that when a new node created, it doesn't go beyond the bounds
    const isWithinContainer = (
        x >= containerRect.left + nodeWidth &&
        x <= containerRect.right - nodeWidth &&
        y >= containerRect.top + nodeWidth &&
        y <= containerRect.bottom - nodeWidth
    );
    return isWithinContainer;
}

//function to add a node to a graph
function add_node(graph_div, x, y){
    //if position isn't within the graph container, node can't be added, so return
false
    if(!isWithinGraphDiv(graph_div, x, y)) return false;

    //create a node - div with class node
    const node = document.createElement('div');
    node.classList.add('node');
    //add node coordinates to its id, this is for accessing it in the future
    //(for example if the node is being moved)
    node.id = x + "-" + y;
    //add the node to graph container for it to appear on the website
    graph_div.appendChild(node);
    //get the node's style -> it's width/diameter
    const nodeStyle = window.getComputedStyle(node);
```

```
    const nodeWidth = parseFloat(nodeStyle.width);
    //calculate top and left coordinates of node by substracting half diameter from centre
    node.style.left = x - nodeWidth/2 +'px';
    node.style.top = y - nodeWidth/2 +'px';
    //add coordinates of created node to the coordinates array to keep track of the nodes
    coordinates.push([parseFloat(x), parseFloat(y)]);
    //return true because the node was created
    return true;
}

//function to generate random nodes
function generate_cities(div, n){
    clear_graph(div);//clear all nodes first
    nodeWidth = get_node_width(div);//get diameter of the node
    //get and calculate the bounds of graph container
    //nodes should be at least diameter from the actual bounds so that nodes don't touch the bound
    const containerRect = div.getBoundingClientRect();
    const left = containerRect.left + nodeWidth;
    const right = containerRect.right - nodeWidth;
    const top = containerRect.top + nodeWidth;
    const bottom = containerRect.bottom - nodeWidth;

    generated = 0;
    while (generated<n){ //lop while didn't generate n nodes
        //create a random coordinate within the container bounds
        let x = Math.floor(Math.random() * (right - left + 1)) + left;
        let y = Math.floor(Math.random() * (top - bottom + 1)) + bottom;
        //if a node can be added (add_node returns true), that increase the count of generated
        if(add_node(div, x, y)) generated++;
    }
}

//event listener for any of users mouse moves/clicks, etc
document.addEventListener('DOMContentLoaded', function(){
    //graph_div is a graph container, that we access by id
    const graph_div = document.getElementById("graph-div");
    //initialise values that we use for graph manipulation
    let selected_node = null;
    let created_node = 0;

    //if users moves mouse down (clicks), it can mean three things:
    //user wants to move a node (if mouse position is close to an existing node)
    //user wants to delete a node (if first applies and it's a right click/ctrl is pressed)
    //if neither of 2 above, user wants to create a new node, but we only add it to display when mouse is moved up
    graph_div.addEventListener('mousedown', function(event){
        //get the coordinates of user's mouse
        const x = event.clientX;
        const y = event.clientY;
        //proceed if and only if mouse position is within the container bounds that allow nodes inside
        if(isWithinGraphDiv(graph_div, x, y)){
```

```
            //graph_locked is true if visualisation is on (se display-output.js for reference)
            if(graph_locked){ //if visualisation is on, user can't do any changes to the display
                //user needs to clear the paths created in the visualisation to be able to edit the display
                alert("clear the paths to edit the graph");
                return;
            }
            //event.button == 2 is if user right clicked
            //on mac, there is no right click, so we have to add an option with a ctrl key
            const right_click = (event.button == 2) || (event.ctrlKey);

            let close_to_coordinate = close_to(x, y);
            //if there is a node that the coordinate is close to, user either wants to move it or delete it
            if (close_to_coordinate[0]!=-1){
                //create an id of the node the mouse is close to
                let id = close_to_coordinate[0]+"-"+close_to_coordinate[1];
                //here we assume that user wants to move the node, and set the selected node to the close node
                selected_node = document.getElementById(id);
                selected_node.style.cursor = 'grabbing';//change the style of the node to grabbing
                //however, if the user right clicks, they want to delete the node, and not move it
                if(right_click){
                    event.preventDefault();//this is just so that context menu doesn't show up as default when right clicking
                    //delete the node from the container
                    graph_div.removeChild(selected_node);
                    //because we used close_to function before, we don't need to delete the node from coordinates array
                    //    as it was already removed from there
                    selected_node = null; //set selected_node back to null as we've deleted the node that was selected
                }
            }
            else if (!right_click) created_node = 1; //if there is no close node and it's not right click, a node is to be created
            //but instead of creating it now, we save a state of creating it, and add to visual display when mouse is realeased
        }
    })
    //if mouse is moved, it only has an effect on graph when a node is selected (so user moves that selected node)
    graph_div.addEventListener('mousemove', function(event){
        //get the coordinates of the mouse
        const x = event.clientX;
        const y = event.clientY;
        //check whether position is within the bounds (to prevent nodes being moved outside the container) and if node is selected
        if(isWithinGraphDiv(graph_div, x, y) && selected_node){
            //change the node position to the position of the mouse - radius
            nodeWidth = get_node_width(graph_div);//get node width/diameter
            selected_node.style.left = x-nodeWidth/2+'px';
```

```
            selected_node.style.top = y-nodeWidth/2+'px';
            //update the id
            selected_node.id = x+"-"+y;
            //note that we don't edit anything in coordinates array here
            //this is because we deleted the node from the array when it was
selected and we will add it back in when mouse is released
        }
    })
    //if mouse is released it can mean 2 things:
    //user moved the selected node to its final position (if selected_node is not
null)
    //user created a node and it is to be added to display when mouse is realesed
(if created_node is true)
    graph_div.addEventListener('mouseup', function(event){
        if (selected_node){//if node is selected, this node was being moved and now
has to be added to coordinates array
            //get the coordinates of the node from its id (id is in the form 'x-y')
            //note that we don't use the position of the mouse here, as it could be
moved outside of bounds and then released
            const x = selected_node.id.split("-")[0];
            const y = selected_node.id.split("-")[1];
            coordinates.push([parseFloat(x), parseFloat(y)]); //add coordinates to
coordinates array to keep it in sync
            selected_node.style.cursor = 'grab'; //change cursor back to grab
            selected_node = null;//node is no longer selected, so reset
selected_node to null
        }
        else if (created_node){//if node was created, it has to be added to the
display
            created_node = 0; //reset created_node back to 0
            //get the coordinates of the mouse
            const x = event.clientX;
            const y = event.clientY;
            if(isWithinGraphDiv(graph_div, x, y)){ //if coordinates are within the
bounds, add the node
                add_node(graph_div, x, y);
            }
        }
    });
    //this is to prevent contexmenu from showing up when user right clicks
    graph_div.addEventListener("contextmenu", function (event) {
        event.preventDefault();
    });
    //if clear button is clicked, clear the graph from nodes
    document.getElementById("btn-clear").addEventListener("click", function(){
        //graph_locked is true if visualisation is on (se display-output.js for
reference)
        if(graph_locked){ //if visualisation is on, user can't do any changes to
the display
            //user needs to clear the paths created in the visualisation to be able
to edit the display
            alert("clear the paths to edit the graph");
            return;
        }
        clear_graph(graph_div); //clear the graph if it's not locked
    });
    //if generate button is clicked, generate a graph
```

```
    document.getElementById("btn-generate").addEventListener("click", function () {
        //graph_locked is true if visualisation is on (se display-output.js for
reference)
        if(graph_locked){ //if visualisation is on, user can't do any changes to
the display
            //user needs to clear the paths created in the visualisation to be able
to edit the display
            alert("clear the paths to edit the graph");
            return;
        }
        //get the value of number of cities input
        const numberOfCities = parseInt(document.getElementById("cities-
input").value);
        //if numberOfCities is a positive integer, generate cities
        if (!isNaN(numberOfCities) && Number.isInteger(numberOfCities) &&
numberOfCities > 0) {
            generate_cities(graph_div, numberOfCities);
        } else { //else output an alert
            alert("Please enter a valid positive integer for the number of
cities.");
        }
    });

})
```

display-output.js

display-output.js fetches user inputs and send a request to flask service when user presses the "start" button, also creates a visualisation of the output

```
//initialise the variables
//last_xx_path is for when path is hiden by user it can be displayed again
var last_aco_path = null;
var last_sa_path = null;
var last_hk_path = null;

var visualisation_on = false; //visualisation is on when iterations are shown
var iteration = -1; //number of iteration that is displayed
var graph_locked = false; //graph is locked when any paths are shown on the display

//function to get algorithms output
async function get_algorithms_output() {
    try{
        //get values of all inputs
        var iterationsACOInput = document.getElementById('iterations-aco-
input').value;
        var antsInput = document.getElementById('ants-input').value;
        var QInput = document.getElementById('Q-input').value;
        var alphaInput = document.getElementById('alpha-input').value;
        var betaInput = document.getElementById('beta-input').value;
        var eRateInput = document.getElementById('e-rate-input').value;
```

```
        var iterationsSAInput = document.getElementById('iterations-sa-
input').value;
        var tInput = document.getElementById('t-input').value;
        var edInput = document.getElementById('ed-input').value;

        //check if any are empty
        if (iterationsACOInput === '' || antsInput === '' || QInput === '' ||
            alphaInput === '' || betaInput === '' || eRateInput === '' ||
iterationsSAInput === '' || tInput === '' || edInput === '') {
            alert('Please fill in all the input fields.');
            reject('Incomplete input fields');
            return;
        }

        //check types of inputs
        function isValidFloat(value) {
            return !isNaN(parseFloat(value));
        }
        function isValidInteger(value) {
            return !isNaN(parseInt(value)) && Number.isInteger(parseFloat(value));
        }

        iterationsACOInput = parseInt(iterationsACOInput)
        iterationsACOInput = (isValidInteger(iterationsACOInput) &&
iterationsACOInput>0) ? iterationsACOInput : (alert('ACO iterations must be a
positive integer'), reject('Wrong format'));

        antsInput = parseInt(antsInput)
        antsInput = (isValidInteger(antsInput) && antsInput>0) ? antsInput :
(alert('Number of ants must be a positive integer'), reject('Wrong format'));

        QInput = parseInt(QInput)
        QInput = (isValidInteger(QInput) && QInput>0) ? QInput : (alert('Q must be
a positive integer'), reject('Wrong format'));

        alphaInput = parseInt(alphaInput)
        alphaInput = (isValidInteger(alphaInput) && alphaInput>0) ? alphaInput :
(alert('Alpha must be a positive integer'), reject('Wrong format'));

        betaInput = parseInt(betaInput)
        betaInput = (isValidInteger(betaInput) && betaInput>0) ? betaInput :
(alert('Beta must be a positive integer'), reject('Wrong format'));

        eRateInput = parseFloat(eRateInput)
        eRateInput = (isValidFloat(eRateInput) && eRateInput>0 && eRateInput<1) ?
eRateInput : (alert('Evaporation rate must be a number between 0 and 1'),
reject('Wrong format'));

        iterationsSAInput = parseInt(iterationsSAInput)
        iterationsSAInput = (isValidInteger(iterationsSAInput) &&
iterationsSAInput>0) ? iterationsSAInput : (alert('SA iterations must be a positive
integer'), reject('Wrong format'));

        tInput = parseInt(tInput)
        tInput = (isValidInteger(tInput) && tInput>0) ? tInput : (alert('T must be
a positive integer'), reject('Wrong format'));
```

```
        edInput = parseFloat(edInput)
        edInput = (isValidFloat(edInput) && edInput>0 && edInput<1) ? edInput :
(alert('Alpha (SA decrease rate) must be a number between 0 and 1'), reject('Wrong
format'));
        console.log(coordinates);
        //create request data
        var requestData = {
            coordinates: coordinates,
            aco_a: alphaInput,
            aco_b: betaInput,
            aco_Q: QInput,
            aco_er: eRateInput,
            aco_ants: antsInput,
            aco_iter: iterationsACOInput,
            aco_shake: 0,
            sa_a: edInput,
            sa_T: tInput,
            sa_iter: iterationsSAInput
        };
        //get the response
        //await is to not continue executing code until we get the response
        const response = await fetch('/calculate_outputs', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify(requestData),
        });
        //get response to json format
        const data = await response.json();
        //console.log(data);
        return data;
    }catch (error) { //if there was an error while executing, output and throw
        console.error('Error:', error);
        throw error;
    }
}

//function that returns values of custom parameters regarding to visualisation
function get_custom_parameters(){
    var speed_input = document.getElementById('speed-input').value; //between 0
anqd 100
    var fast_forward = document.getElementById('fast-forward').checked; //false or
true
    var hide_aco = document.getElementById('aco-hide').checked;
    var hide_sa = document.getElementById('sa-hide').checked;
    var hide_hk = document.getElementById('hk-hide').checked;
    //return a dictionary with all input values
    return {
        "speed-input": speed_input,
        "fast-forward": fast_forward,
        "hide-aco": hide_aco,
        "hide-sa": hide_sa,
        "hide-hk": hide_hk
    }
}
```

```
//function that creates an arc in visualisation
function draw_arc(div, x1, y1, x2, y2, class_name){
    //create a line (div) with class arc and another class_name
    const line = document.createElement('div');
    line.classList.add(class_name);
    line.classList.add("arc");

    //d is a parameter - number of pixels line is shifted to the right
    //it's o for hk, 2 for aco, and -2 for sa (so that lines don't overlap and all can be seen)
    var d = 0;
    if(class_name=="aco-path") d = 2;
    if(class_name=="sa-path") d = -2;

    //calculate width(length) and angle to the horizontal using coordinates of start and end
    const width = Math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2);
    const angle = Math.atan2(y2 - y1, x2 - x1) * (180 / Math.PI);

    //shifting d pixels to the right
    x1 = x1 - d * Math.sin(angle * (Math.PI / 180));
    y1 = y1 + d * Math.cos(angle * (Math.PI / 180));

    //add all these values to arc's style
    line.style.left = `${x1}px`;
    line.style.top = `${y1}px`;
    line.style.width = `${width}px`;
    line.style.transformOrigin = '0 0';
    line.style.transform = `rotate(${angle}deg)`;
    //and add the arc to graph container so that it's shown on the display
    div.appendChild(line);
}

//function that displays a path
function display_iteration_path(path, coordinates, class_name){
    //hide the path for this particular output type first
    hide_path(class_name);
    //iterate through nodes in the path
    for(let i=0; i<path.length-1; i++){
        //get coordinates of 2 neighbouring nodes
        var x1 = coordinates[path[i]][0]; var y1 = coordinates[path[i]][1];
        var x2 = coordinates[path[i+1]][0]; var y2 = coordinates[path[i+1]][1];
        //draw an arc in graph container
        graph_div = document.getElementById("graph-div");
        draw_arc(graph_div, x1, y1, x2, y2, class_name);
    }
}

//function that hides a path of a particular class
function hide_path(class_name){
    //get all elements with the class
    var elements = document.getElementsByClassName(class_name);
    for (var i = elements.length-1; i>=0; i--) {
        elements[i].remove();//remove them one by one
    }
}
```

```
//function that updates a value in a certain text box
function updateValue(id, value) {
    const valueElement = document.getElementById(id);
    valueElement.textContent = value;
}

//function that shows/hides paths outputs depending on the state of "hide" check
function update_hide_paths(){
    custom_parameters = get_custom_parameters();

    if(custom_parameters["hide-aco"] == false && last_aco_path != null)
display_iteration_path(last_aco_path, coordinates, "aco-path");
    if(custom_parameters["hide-aco"] == true) hide_path("aco-path");

    if(custom_parameters["hide-sa"] == false && last_sa_path != null)
display_iteration_path(last_sa_path, coordinates, "sa-path");
    if(custom_parameters["hide-sa"] == true) hide_path("sa-path");

    if(custom_parameters["hide-hk"] == false && last_hk_path != null)
display_iteration_path(last_hk_path, coordinates, "hk-path");
    if(custom_parameters["hide-hk"] == true) hide_path("hk-path");
}

//function that clears all text outputs
function clear_outputs(){
    updateValue("iteration-number", 0);
    updateValue("aco-value", 0);
    updateValue("aco-found", 0);
    updateValue("sa-value", 0);
    updateValue("sa-found", 0);
    updateValue("hk-value", 0);
}

//function that displays output from algorithms
function display_output(output, custom_parameters){
    //create variables for each output
    var hk_output = JSON.parse(output["hk_output"]);
    var aco_output = output["aco_output"];
    var sa_output = output["sa_output"];
    var aco_it_found = output["aco_it_found"];
    var sa_it_found = output["sa_it_found"];

    //calculate the number of iterations that wee need to show
    var max_iteration = Math.max(aco_output.length, sa_output.length);
    //set aco and sa best to -1 (best cost of path)
    var aco_best = -1;
    var aco_found = -1;
    var sa_best = -1;
    var sa_found = -1;

    //we do not iterate through held-karp, so we can display it's output straight
away
    last_hk_path = hk_output["path"]; //set last held-karp path to the held-karp
path (there is only one path)
    updateValue("hk-value", Math.round(parseFloat(hk_output["cost"]))); //update
value of the cost
    if(!custom_parameters["hide-hk"]){ //if hide isn't checked, display the path
```

```
        display_iteration_path(hk_output["path"], coordinates, "hk-path");
    }

    function display_iterations(){
        //if visualisation isn't on, return
        if(!visualisation_on) return;
        //get custom parameters
        custom_parameters = get_custom_parameters();

        var speed = custom_parameters["speed-input"]; //1 to 100
        var fast_forward = custom_parameters["fast-forward"]; //true if display
with maximum speed
        var i = iteration;
        iteration ++; //increase iteration

        if(fast_forward) updateValue("iteration-number", max_iteration); //if fast
forward, show the last iteration number
        else updateValue("iteration-number", i+1); //update the number of iteration

        if(fast_forward) i=aco_it_found;//if fast forward, show the best aco
iteration
        if(i<aco_output.length) { //if current iteration is less or equal to the
maximum aco iteration
            var aco_iteration = JSON.parse(aco_output[i]); //get a dictionary of
aco iteration
            if (aco_best == -1 || parseFloat(aco_iteration.cost) <aco_best){ //if
current iteration is better than the best
                aco_best = parseFloat(aco_iteration.cost); //set best cost to
current cost
                aco_found = i+1;
                last_aco_path = aco_iteration.best_route; //update last aco path /
best aco path
                if(custom_parameters["hide-aco"]==false)
display_iteration_path(aco_iteration.best_route, coordinates, "aco-path");
                updateValue("aco-value", Math.round(aco_best));
                updateValue("aco-found", i+1);
            }
        }
        if(fast_forward) i=sa_it_found;//if fast forward, show the best sa
iteration
        if(i<sa_output.length) {//if current iteration is less or equal to the
maximum sa iteration
            var sa_iteration = JSON.parse(sa_output[i]);//get a dictionary of sa
iteration
            if (sa_best == -1 || parseFloat(sa_iteration.cost) <sa_best){ //if
current iteration is better than the best
                sa_best = parseFloat(sa_iteration.cost); //set best cost to current
cost
                sa_found = i+1;
                last_sa_path = sa_iteration.path; //update last sa path / best sa
path
                if(custom_parameters["hide-sa"]==false)
display_iteration_path(sa_iteration.path, coordinates, "sa-path");
                updateValue("sa-value", Math.round(sa_best));
                updateValue("sa-found", i+1);
            }
        }
```

```
        if(iteration >= max_iteration || fast_forward){ //if iteration is the last
possible iteration, we need to stop
            visualisation_on = false; //reset visualisation_on to false
            iteration = -1; //reset iteration to -1
            return;
        }
        //execute next iteration with a time delay, that is calculated using the
speed input
        var delay = (100 - speed) * 10;
        setTimeout(function () {
            display_iterations();
        }, delay);

    }
    //display the first iteration
    display_iterations();

}



//event listener for any of users mouse moves/clicks, etc
document.addEventListener('DOMContentLoaded', function() {
    //when the start button is clicked, visualisation starts (unless it's already
on)
    document.querySelector('.btn-start').addEventListener('click', async function()
{
        if(visualisation_on){ //if visualisation is on, alert and return
            alert("visualisation is already on");
            return;
        }
        //if visualisation is not on already, get the output of algorithms
        //await is here so that no further instruction are executed until we get a
response
        var output = await get_algorithms_output();
        var custom_parameters = get_custom_parameters(); //get custom parameters
        console.log(output);
        console.log(custom_parameters);

        visualisation_on = true; //set visualisation_on to true
        graph_locked = true; //lock he graph (so that user can't edit the nodes)
        if (iteration == -1) iteration = 0;
        //if iteration was -1, it means that previous visualisation was finished,
so we set iteration to 0
        //if iteration is not -1, it means that the previous visualisation was
paused, so we just continue without resetting
        display_output(output, custom_parameters); //this function displays the
output of algorithms
    });
    //when stop button is clicked, it pauses the visualsitation at current
iteration
    document.querySelector('.btn-stop').addEventListener('click', function() {
        if(!visualisation_on){ //if visualisation is not on, there is nothing to
pause
            alert("visualisation is not on");
            return;
        }//otherwise, reset visualisation_on to false
```

```
        visualisation_on = false;
        iteration -= 1; //sudstract 1 from iteration number (so that when
visualisation is resumed, no iterations are skiped)
    });
    //if clear paths button is clicked, clear visualisation from paths and reset
everything
    document.querySelector('.btn-clear-paths').addEventListener('click', function()
{
        visualisation_on = false; //turn visualisation off
        iteration = -1; //reset iteration number
        last_aco_path = null; //reset last_aco_paths
        last_sa_path = null;
        last_hk_path = null;
        graph_locked = false; //unlock the graph
        hide_path('arc'); //hide all paths
        clear_outputs(); //clear all text outputs
    });
    //if any of hide checks are checked/unchecked, update all paths and hide/show
them accordingly
    document.getElementById("aco-hide").addEventListener('change', function(){
        update_hide_paths();
    })
    document.getElementById("sa-hide").addEventListener('change', function(){
        update_hide_paths();
    })
    document.getElementById("hk-hide").addEventListener('change', function(){
        update_hide_paths();
    })
});
```

Algorithms

common.py

file common.py has some class types, functions, and constants that all of the algorithms use

```
import math

inf = 10000000000
max_n = 15 #for held karp

class TSP_input():
    def __init__(self, n, dist, coordinates):
        self.n = n
        self.dist = dist
        self.coordinates = coordinates
```

```python
class ACO_parameters(): #OBJECTIVE 2.2
    def __init__(self, alpha, beta, Q, evaporation_rate, n_ants, iterations,
shake):
        self.alpha = alpha
        self.beta = beta
        self.Q = Q
        self.evaporation_rate = evaporation_rate
        self.n_ants = n_ants
        self.iterations = iterations
        self.shake = shake

class ACO_output():
    def __init__(self, n, n_ants, ant_route, best_route, pheromone, cost):
        self.n = n
        self.n_ants = n_ants
        self.ant_route = ant_route
        self.best_route = best_route
        self.pheromone = pheromone
        self.cost = cost


class SA_parameters(): #OBJECTIVE 3.2
    def __init__(self, alpha, T, iterations):
        self.alpha = alpha
        self.T = T
        self.iterations = iterations

class SA_output():
    def __init__(self, T, path, cost, probability):
        self.T = T
        self.path = path
        self.cost = cost
        self.probability = probability

class heldkarp_output():
    def __init__(self, cost, path):
        self.path = path
        self.cost = cost


def calculate_distance_matrix(coordinates):
    dist = []
    n = len(coordinates)

    for [x1, y1] in coordinates:
        dist_row = []
        for [x2, y2] in coordinates:
            dist_row.append(math.sqrt(pow(x1-x2, 2)+pow(y1-y2, 2)))
        dist.append(dist_row)
    return n, dist
```

aco.py

Ant Colony Optimisation

```
from random import randint, random
from algorithms.common import *

class Graph():
    def __init__(self, n, dist): #n - number of cities, dist - distance matrix

        if n==0: # check if any cities are given
            return "No cities are given"
        if n!=len(dist) or n!=len(dist[0]): #check if the size of matrix is correct
            return "The size of the distance matrix isn't correct"


        self.n = n
        max_d = 0 #maximum distance
        self.pheromone = []
        self.heuristic = []
        self.dist = dist
        self.update = []
        for i in range(n):
            self.pheromone.append([])
            self.update.append([])
            for j in range(n):
                if i == j:   dist[i][j] = inf
                if dist[i][j]>max_d:     max_d = dist[i][j]; #find max distance
                self.pheromone[i].append(1) #at first, pheromones are equal to 1
                if i == j:  self.pheromone[i][j] = 0

                self.update[i].append(0)

        for i in range(n):
            self.heuristic.append([])
            for j in range(n):
                self.heuristic[i].append(max_d/dist[i][j]) #heuristic value of edge
is max distance / length od edge
                # heuristic value - attractiveness of edge, the bigger is distance,
less attractive the edge is
                if i==j:     self.heuristic[i][j] = 0
                if dist[i][j] == 0:  self.heuristic[i][j] = 0; #to avoid inf
heuristic


    def choose_first_city(self):
        return randint(0, self.n-1)

    def choose_next_city(self, route, used, parameters):
        l = len(route)
        last_city = route[-1]
        probabilities = []
        sum_probabilities = 0
```

```
        if l == self.n:
            return -1; #if there are no available vertices, path is finished, so
return -1

        for i in range(self.n):

            #if we already used an edge, its probability is 0
            if i in used:
                probability = 0
            else:
                #probability of edge is pheromone^alpha * heuristic*beta
                probability = pow(self.pheromone[last_city][i], parameters.alpha) *
pow(self.heuristic[last_city][i], parameters.beta)

            probabilities.append(probability)
            sum_probabilities += probability

        if sum_probabilities==0: #if all probabilities are 0, just choose any node
that hasn't been used yet
            for i in range(self.n):
                if not i in used:
                    return i

        for i in range(self.n):
            probabilities[i] = probabilities[i] * 1.0 / sum_probabilities #divide
all probabilities by their total
            #so that the sum of all probabilities is 1


        random_n = random() #generate random number in range [0,1)

        #find in which range the random_n lies and return the vertex number
        for i in range(self.n):
            if i in used:   continue #don't consider vertices that are already used
            random_n -= probabilities[i];
            if random_n<=0: return i

        return -1 # if nothing found, return -1


    def path_length(self, path): #simple function for finding the path length
        length = 0
        for i in range (1, len(path)):
            length += self.dist[path[i-1]][path[i]] #sum up the distances between
consecutive nodes
        return length

    def update_pheromone_levels(self, parameters): #update pheromone levels (after
each iteration)
        for i in range(self.n):
            for j in range(self.n):
                self.pheromone[i][j] = (1-
parameters.evaporation_rate)*self.pheromone[i][j] #decrease by percentage dictated
by evaporation rate
                self.pheromone[i][j] += self.update[i][j] #add the update (delta
tau sum)
```

```
    def clear_update(self): #set update to 0
        for i in range(self.n):
            for j in range(self.n):
                self.update[i][j] = 0


    def shake_pheromones(self): #optional function that is executed when the best
route is improved
        #sets value of pheromone for all edges to the mean of all pheromone values
        total_pheromone = 0

        for i in range(self.n):
            for j in range(self.n):
                total_pheromone += self.pheromone[i][j]

        mean = total_pheromone/(self.n*(self.n-1))
        for i in range(self.n):
            for j in range(self.n):
                self.pheromone[i][j] = mean

class Ant():

    def __init__(self, ant_number):
        self.k = ant_number
        self.path_length = 0
        self.path = []

    def construct_path(self, graph, parameters):
        next_city = graph.choose_first_city() #choose first city
        self.path.clear()
        used = []
        while next_city!=-1: #while it's possible to choose another node
            self.path.append(next_city) #add current node to the path
            used.append(next_city)
            next_city = graph.choose_next_city(self.path, used, parameters) #choose
the next node randomly using the probabilstic function

        self.path.append(self.path[0]) #path has to be a cycle, so add the first
city to the end
        self.path_length = graph.path_length(self.path)


class ACO():
    def __init__(self, tsp_input, parameters):
        self.n_ants = parameters.n_ants
        self.ants = []
        for ant in range(self.n_ants):
            self.ants.append(Ant(ant)) #create artificial ants :) my favourite part

        self.graph = Graph(tsp_input.n, tsp_input.dist)
        self.parameters = parameters

    def construct_paths(self): #construct path for each ant
        for i in range(len(self.ants)):
            self.ants[i].construct_path(self.graph, self.parameters)
```

```
    def compare_edges(self): #calculate update (sum of delta tau) for each edge
        self.graph.clear_update()
        for k in range(len(self.ants)):
            ant = self.ants[k]
            for i in range(1, len(ant.path)):
                self.graph.update[ant.path[i-1]][ant.path[i]] += self.parameters.Q
/ ant.path_length;

    def iteration(self):
        self.construct_paths() #construct path for each ant
        self.compare_edges() #create an update
        self.graph.update_pheromone_levels(self.parameters) #update pheromone
levels

        #the rest of this function is creating the output
        #OBJECTIVE 2.3 - return data after each iteration
        best_length = inf
        best_route = []
        ant_route = []

        for k in range(len(self.ants)):
            ant = self.ants[k]
            if ant.path_length < best_length:
                best_length = ant.path_length
                best_route = ant.path
            ant_route.append(ant.path)

        output = ACO_output(self.graph.n, self.parameters.n_ants, [], best_route,
[], best_length)
        return output

    def shake(self): #shake function
        self.graph.shake_pheromones()


def solve_aco(input, parameters):
    aco = ACO(input, parameters)

    best = inf
    best_found = 0
    output = []

    for iteration in range(parameters.iterations):
        iteration_output = aco.iteration()
        output.append(iteration_output) #OBJECTIVE 2.3

        if iteration_output.cost < best: #if the length is less than the best
length stored#
            #update the best solution
            best = iteration_output.cost
            best_found = iteration


    return best_found, output
```

sa.py

```
from math import exp
from algorithms.common import *
import numpy as np
import random
import copy


class Graph():
    def __init__(self, n, dist, coordinates): #n - number of cities, dist -
distance matrix

        if n==0: # check if any cities are given
            return "No cities are given"
        if n!=len(dist) or n!=len(dist[0]): #check if the size of matrix is correct
            return "The size of the distance matrix isn't correct"


        self.n = n
        for i in range(n):
            for j in range(n):
                if i == j:   dist[i][j] = inf

        self.dist = dist
        self.coordinates = coordinates


class State():
    def __init__(self, graph, **kwargs):
        type = kwargs['type']

        if type == "random": #if a random state has to be generated, use the
generate_random_state() function
            self.generate_random_state(graph)

        elif type == "neighbour": #if a neighbouring state has to be generated,
retrieve the current state attrbute and modify it to create a different state
            current_state = kwargs["current_state"]
            self.path = current_state.path
            self.generate_neighbour_state(graph)

        else:
            return

        self.calculate_cost(graph) # finaly, calculate the cost of the state


    def calculate_cost(self, graph):
        self.cost = 0 #initialise cost as 0
        for i in range(1, len(self.path)):
            self.cost += graph.dist[self.path[i]][self.path[i-1]] #add the weight
of each edge in the path
```

```
    def generate_random_state(self, graph):
        n = graph.n
        array = np.array(list(range(n))) #generate an array [0, 1, 2, ..., n]
        path = np.random.permutation(array) #shuffle the array to create a random path
        path = np.concatenate((path, [path[0]])) #append the first node to the end to create a cycle

        self.path = path

    def generate_neighbour_state(self, graph):

        self.path = self.path[:-1] #cut the last node as it's a repeat of the first node

        x = random.randint(1, 4) #generate a random number to choose one of 3 modifications
        if x==1:
            self.swap_two_nodes()
        elif x==2:
            self.reverse_segment()
        elif x==3:
            self.insert_random_node()
        else:
            self.insert_random_segment()

        self.path = np.concatenate((self.path, [self.path[0]])) #append the fist node to the end to create a cycle



    def swap_two_nodes(self):
        index = random.randint(0, len(self.path)-2) #choose a random index
        self.path[index], self.path[index+1] = self.path[index+1], self.path[index] #swap 2 neighbouring nodes

    def reverse_segment(self):
        #choose 2 distinct random numbers from 0 to n-1 (we're using sample function so that they are distinct)
        index1, index2 = random.sample(range(len(self.path)), 2)
        start, end = min(index1, index2), max(index1, index2) #assign start and end of the segment

        self.path[start:end+1] = self.path[start:end+1][::-1] #reverse the segment

    def insert_random_node(self):

        index_to_move = random.randint(0, len(self.path)-1) # choose a random index for the item to be moved
        # choose a random destination index different from the source index
        index_destination = random.choice([i for i in range(len(self.path)) if i != index_to_move])

        # move the item to the new position
        item_to_move = self.path[index_to_move]
        self.path = np.delete(self.path, index_to_move)
```

```
        self.path = np.insert(self.path, index_destination, item_to_move)


    def insert_random_segment(self):
        #choose 2 random indecies
        index1, index2 = random.sample(range(self.path.size), 2)
        start, end = min(index1, index2), max(index1, index2)

        segment = self.path[start:end+1] # extract the random segment
        self.path = np.delete(self.path, slice(start, end+1)) # delete the segment
from the original position

        index_destination = random.randint(0, self.path.size)# choose a random
destination index different from the source index
        self.path = np.insert(self.path, index_destination, segment)# insert the
segment at the new position


class SA():
    #function to initialise the problem
    def __init__(self, graph_params, sa_params):
        self.graph = Graph(graph_params.n, graph_params.dist,
graph_params.coordinates) #create a graph object
        #store sa parameters in SA object
        self.T = sa_params.T
        self.alpha = sa_params.alpha
        self.state = State(self.graph, type="random") #create a random state

    def iteration(self):

        #generate a neighbouring state
        current_state = self.state
        new_state = State(self.graph, type="neighbour", current_state =
copy.deepcopy(current_state))

        accept = False
        probability = 1

        #if cost of new state is less, accept it
        if new_state.cost < current_state.cost:
            accept = True

        #if not, accept it with probability P = exp(-dC/T)
        else:
            delta_cost = new_state.cost - current_state.cost
            probability = exp(-delta_cost/self.T)

            if probability >= random.random():
                accept = True

        if accept:
            self.state = copy.deepcopy(new_state)

        # Temperature decay
        self.T = self.alpha*self.T

        #OBJECTIVE 3.3 - return output after each iteration
```

```
        output = SA_output(self.T, current_state.path.tolist(), current_state.cost,
probability)

        return output


def solve_sa(tsp_input, sa_params):
    sa = SA(tsp_input, sa_params) #initialise the problem
    output = []
    best = inf
    best_found = 0

    for iteration in range(sa_params.iterations):

        iteration_output = sa.iteration()
        output.append(iteration_output) #OBJECTIVE 3.3

        if iteration_output.cost < best: #if the length is less than the best
length stored#
            #update the best solution
            best = iteration_output.cost
            best_found = iteration

    return best_found, output
```

held_karp.py

```
from algorithms.common import *


def held_karp(input):
    n = input.n
    dist = input.dist

    if n>max_n:
        return heldkarp_output(0, [])

    dp = [[inf] * max_n for _ in range(1 << n)]  # Set S is represented as a binary
number of length n where 1 at position x corresponds to including city x
    path = [[-1] * max_n for _ in range(1 << n)]  # To store the path information


    for mask in range(1, 1 << n): #set all elements of dp array to infinity
        for v in range(1, n):
            dp[mask][v] = inf

    dp[1][0] = 0  # distance from 1 to 1 is 0

    for mask in range(1, 1 << n):  # mask represents a set S
        for v in range(1, n):
            for u in range(n):
                #1<<v is left shift of 1, v bits (the same as 2 to the power of v)
```

```
                #mask & (1<<v) is 0 if mask has 0 at position v -> so S doesn't
contain v
                #mask & (1<<v) is 1 if mask has 1 at position v -> so S contains v
                if u == v or not (mask & (1 << v)) or not (mask & (1 << u)):
                    continue  # S has to contain v and u
                if dist[u][v] != -1:
                    # ^ is xor, so mask ^ (1<<v) is S\v
                    if dp[mask][v] > dp[mask ^ (1 << v)][u] + dist[u][v]:
                        dp[mask][v] = dp[mask ^ (1 << v)][u] + dist[u][v]
                        path[mask][v] = u

    mask = (1 << n) - 1  # represents set S = {1,2,3,...,n}
    min_dist = inf
    end_vertex = -1 #this is needed to than reconstruct the optimal path

    for v in range(1, n):
        #find the minimum distance among all dp values
        if dist[v][0] != -1 and min_dist > dp[mask][v] + dist[v][0]:
            min_dist = dp[mask][v] + dist[v][0]
            end_vertex = v

    # reconstruct the path
    path_list = []
    while mask > 0 and end_vertex!=-1:
        path_list.append(end_vertex)
        u = path[mask][end_vertex]
        mask ^= (1 << end_vertex)
        end_vertex = u

    path_list.append(path_list[0])
    return heldkarp_output(min_dist, path_list) #OBJECTIVE 4.2 - return best route
and its cost
```