ai-cup

Generated by Doxygen 1.9.2

Chapter 1

USI 16th Edition AI Cup

This repo represents my submission for the 2021 AI Cup. More information about how I solved the Cup can be found in the `docs` directory.

1.1 How to run

In order to start the TSP-solver, you need to run the following command inside of the root directory of the project:
`make run`

1.2 Solvers used

In order to solve the TSPs, I used the following solvers:

- Best Nearest Neighbor (BNN)
- 2.5-opt

1.3 Results

Problem	Best Known	Student Result	Error
ch130	6110	6486	6.15%
d198	15780	16242	2.93%
eil76	538	563	4.65%
fl1577	22249	22939	3.10%
kroa100	21282	21355	0.34%
lin318	42029	43355	3.15%
pcb442	50778	52092	2.59%
pr439	107217	114668	6.95%
rat783	8806	9257	5.12%
u1060	224094	239671	6.95%

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 Problem Class Reference

`#include <Problem.hpp>`

Public Member Functions

- Problem (string filename)

  Contructor of the Problem class.
- void print_info ()

  Prints the problem info.

Public Attributes

- bool exists_optimal = false
- string name
- string filename
- int n_points
- float best_solution_len
- vector< string > lines
- vector< int > optimal_tour
- vector< vector< float > > points
- vector< vector< float > > distance_matrix

Private Member Functions

- vector< string > read_file (string filename)

  Reads the problem from a file.
- vector< vector< float > > parse_points (vector< string > lines, int n_points)

  Parses the points of the problem.
- vector< vector< float > > compute_distance_matrix (vector< vector< float > > points, int n_points)

  Compute the distance matrix.
- vector< int > check_optimal_tour ()

  Parses the optimal tour.

4.1.1 Constructor & Destructor Documentation

4.1.1.1 Problem()

```
Problem::Problem (
            string filename )  [inline]
```

Contructor of the Problem class.

Parameters

filename	The name of the file

4.1.2 Member Function Documentation

4.1.2.1 check_optimal_tour()

```
vector< int > Problem::check_optimal_tour ( )  [inline], [private]
```

Parses the optimal tour.

Returns

A vector containing the optimal tour

4.1.2.2 compute_distance_matrix()

```
vector< vector< float > > Problem::compute_distance_matrix (
            vector< vector< float > > points,
            int n_points )  [inline], [private]
```

Compute the distance matrix.

Parameters

points	The points of the problem
n_points	The total number of points

Returns

A matrix containing the distance matrix

4.1.2.3 parse_points()

```
vector< vector< float > > Problem::parse_points (
            vector< string > lines,
            int n_points )  [inline], [private]
```

Parses the points of the problem.

Parameters

lines	The lines of the file
n_points	The total number of points

Returns

A matrix containing the points

4.1.2.4 print_info()

```
void Problem::print_info ( )  [inline]
```

Prints the problem info.

4.1.2.5 read_file()

```
vector< string > Problem::read_file (
            string filename )  [inline], [private]
```

Reads the problem from a file.

Parameters

filename	The name of the file

Returns

A vector of strings containing the lines of the file

4.1.3 Member Data Documentation

4.1.3.1 best_solution_len

```
float Problem::best_solution_len
```

4.1.3.2 distance_matrix

```
vector<vector<float> > Problem::distance_matrix
```

4.1.3.3 exists_optimal

```
bool Problem::exists_optimal = false
```

4.1.3.4 filename

```
string Problem::filename
```

4.1.3.5 lines

```
vector<string> Problem::lines
```

4.1.3.6 n_points

```
int Problem::n_points
```

4.1.3.7 name

```
string Problem::name
```

4.1.3.8 optimal_tour

```
vector<int> Problem::optimal_tour
```

4.1.3.9 points

```
vector<vector<float> > Problem::points
```

The documentation for this class was generated from the following file:

- src/classes/Problem.hpp

4.2 Solver Class Reference

```
#include <Solver.hpp>
```

Public Member Functions

- Solver (Problem *problem)
  Construct a new Solver object.

Private Member Functions

- vector< float > compute_solution ()
  Solve the problem using the given algorithms.
- bool check_validity (vector< float > solution)
  Check if the solution is valid.
- float gap ()
  Compute gap.

Private Attributes

- int algo_name
- float duration
- int found_length
- Problem * problem
- vector< float > solution

4.2.1 Constructor & Destructor Documentation

4.2.1.1 Solver()

```
Solver::Solver (
            Problem * problem )  [inline]
```

Construct a new Solver object.

Parameters

problem	The problem to solve

4.2.2 Member Function Documentation

4.2.2.1 check_validity()

```
bool Solver::check_validity (
            vector< float > solution )  [inline], [private]
```

Check if the solution is valid.

Parameters

solution	The solution vector

Returns

true If the solution is valid

false If the solution is not valid

4.2.2.2 compute_solution()

```
vector< float > Solver::compute_solution ( )  [inline], [private]
```

Solve the problem using the given algorithms.

Returns

The solution vector of the problem

4.2.2.3 gap()

```
float Solver::gap ( )  [inline], [private]
```

Compute gap.

Compute the gap between the best known solution and the solution from the current algorithm.

Returns

The gap

4.2.3 Member Data Documentation

4.2.3.1 algo_name

```
int Solver::algo_name  [private]
```

4.2.3.2 duration

```
float Solver::duration  [private]
```

4.2.3.3 found_length

```
int Solver::found_length  [private]
```

4.2.3.4 problem

```
Problem* Solver::problem  [private]
```

4.2.3.5 solution

```
vector<float> Solver::solution  [private]
```

The documentation for this class was generated from the following file:

- src/classes/Solver.hpp

Chapter 5

File Documentation

5.1 README.md File Reference

5.2 src/classes/Problem.hpp File Reference

```
#include <vector>
#include <string>
#include <fstream>
#include <iostream>
#include "../utils/split.hpp"
#include "../utils/matrices.hpp"
#include "../utils/distance.hpp"
```

Classes

- class Problem

Macros

- #define PROBLEM_HPP

5.2.1 Macro Definition Documentation

5.2.1.1 PROBLEM_HPP

```
#define PROBLEM_HPP
```

5.3 Problem.hpp

Go to the documentation of this file.

```
1 #include <vector>
2 #include <string>
3 #include <fstream>
4 #include <iostream>
5
6 #include "../utils/split.hpp"
7 #include "../utils/matrices.hpp"
8 #include "../utils/distance.hpp"
9
10 using namespace std;
11
12 #ifndef PROBLEM_HPP
13 #define PROBLEM_HPP
14
15 class Problem {
16 public:
17   bool exists_optimal = false;
18
19   string name;
20   string filename;
21
22   int n_points;
23   float best_solution_len;
24
25   vector<string> lines;
26   vector<int> optimal_tour;
27   vector<vector<float> > points;
28   vector<vector<float> > distance_matrix;
29
30
36   Problem(string filename) {
37     this->filename = filename;
38     this->lines = read_file(filename);
39
40     this->name = split(lines[0], " : ")[1];
41     this->n_points = stoi(split(lines[3], " : ")[1]);
42     this->best_solution_len = stof(split(lines[5], " : ")[1]);
43
44     this->points = parse_points(lines, this->n_points);
45     this->distance_matrix = compute_distance_matrix(this->points, this->n_points);
46     this->optimal_tour = check_optimal_tour();
47   }
48
49
53   void print_info() {
54     cout « "\n\n##############################" « endl;
55     cout « "Problem: " « this->name « endl;
56     cout « "Number of points: " « this->n_points « endl;
57     cout « "Best solution: " « (int)this->best_solution_len « endl;
58     cout « "Optimal tour: " « boolalpha « this->exists_optimal « endl;
59   }
60
61 private:
68   vector<string> read_file(string filename) {
69     ifstream file(filename);
70     vector<string> lines;
71     string line;
72
73     while (getline(file, line)) {
74       lines.push_back(line);
75     }
76
77     return lines;
78   }
79
80
88   vector<vector<float> > parse_points(vector<string> lines, int n_points) {
89     vector<vector<float> > points(n_points, vector<float>(3));
90
91     for (int i = 7; i < n_points + 7; i++) {
92       points[i-7][0] = stof(split(lines[i], " ")[0]);
93       points[i-7][1] = stof(split(lines[i], " ")[1]);
94       points[i-7][2] = stof(split(lines[i], " ")[2]);
95     }
96
97     return points;
98   }
99
100
108    vector<vector<float> > compute_distance_matrix(vector<vector<float> > points, int n_points) {
109      vector<vector<float> > distance_matrix(n_points, vector<float>(n_points));
110
```

```
111     for (int i = 0; i < n_points; i++) {
112       for (int j = i; j < n_points; j++) {
113         distance_matrix[i][j] = distance(points[i], points[j]);
114       }
115     }
116 
117     vector<vector<float> > transposed_matrix = m_transpose(distance_matrix);
118     distance_matrix = m_sum(distance_matrix, transposed_matrix);
119 
120     return distance_matrix;
121   }
122 
123 
129   vector<int> check_optimal_tour() {
130     vector<int> tour(this->n_points);
131 
132     if (this->filename == "./problems/eil76.tsp" || this->filename == "./problems/kroA100.tsp") {
133       this->exists_optimal = true;
134 
135       ifstream file(this->filename.replace(this->filename.end()-3, this->filename.end(), "opt.tour"));
136       vector<string> lines;
137       string line;
138 
139       while (getline(file, line)) {
140         lines.push_back(line);
141       }
142 
143       for (int i = 6; i < this->n_points + 6; i++) {
144         tour[i-6] = stoi(lines[i]);
145       }
146 
147       return tour;
148     }
149 
150     return vector<int>();
151   }
152 };
153 
154 #endif // PROBLEM_HPP
```

5.4 src/classes/Solver.hpp File Reference

```
#include <ctime>
#include <vector>
#include "../utils/length.hpp"
#include "../utils/check_in.hpp"
#include "../solvers/two_opt.hpp"
#include "../solvers/two_dot_five_opt.hpp"
#include "../solvers/nearest_neighbors.hpp"
```

Classes

- class Solver

Macros

- #define SOLVER_HPP

5.4.1 Macro Definition Documentation

5.4.1.1 SOLVER_HPP

```
#define SOLVER_HPP
```

5.5 Solver.hpp

Go to the documentation of this file.

```
1 #include <ctime>
2 #include <vector>
3
4 #include "../utils/length.hpp"
5 #include "../utils/check_in.hpp"
6
7 #include "../solvers/two_opt.hpp"
8 #include "../solvers/two_dot_five_opt.hpp"
9 #include "../solvers/nearest_neighbors.hpp"
10
11
12 using namespace std;
13
14 #ifndef SOLVER_HPP
15 #define SOLVER_HPP
16
17 class Solver {
18 private:
19   int algo_name;
20   float duration;
21   int found_length;
22
23   Problem * problem;
24   vector<float> solution;
25
26
32   vector<float> compute_solution() {
33     clock_t start = clock();
34
35     cout « endl;
36     cout « "### Solving problem with ['best_nearest_neighbors', 'two_dot_five_opt'] ###" « endl;
37
38     vector<float> solution = best_nearest_neighbors(this->problem);
39     solution = two_dot_five_opt(solution, this->problem);
40
41     if (!check_validity(solution)) {
42       cout « "ERROR: Solution is not valid" « endl;
43       exit(1);
44     }
45
46     this->duration = (clock() - start) / (double) CLOCKS_PER_SEC;
47     this->found_length = length(solution, this->problem->distance_matrix);
48
49     cout « "Solution found in " « this->duration « " seconds" « endl;
50     cout « "Legnth of solution: " « this->found_length « endl;
51     cout « "Gap between solutions: " « fixed « setprecision(2) « this->gap() « "%" « endl;
52
53     return solution;
54   }
55
56
64   bool check_validity(vector<float> solution) {
65     int sum = 0;
66
67     for (int i = 0; i < this->problem->n_points; i++) {
68       if (check_in(solution, i)) sum += 1;
69     }
70
71     return sum == this->problem->n_points;
72   }
73
74
83   float gap() {
84     float num = this->found_length - this->problem->best_solution_len;
85     float den = this->problem->best_solution_len;
86
87     return (num / den) * 100.0;
88   }
89
90
91 public:
97   Solver(Problem * problem) {
```

```
98      this->duration = 0;
99      this->found_length = 0;
100
101      this->problem = problem;
102      this->solution = compute_solution();
103    }
104 };
105
106 #endif // SOLVER_HPP
```

5.6 src/main.cpp File Reference

```
#include <vector>
#include "./classes/Problem.hpp"
#include "./classes/Solver.hpp"
```

Functions

- int main (int argc, char **argv)

5.6.1 Function Documentation

5.6.1.1 main()

```
int main (
            int argc,
            char ** argv )
```

5.7 src/solvers/nearest_neighbors.hpp File Reference

```
#include <vector>
```

Functions

- vector< float > nearest_neighbors (Problem *problem, vector< bool > &visited, int start=0)

  Nearest neighbors.
- vector< float > best_nearest_neighbors (Problem *problem)

  Best nearest neighbors.

5.7.1 Function Documentation

5.7.1.1 best_nearest_neighbors()

```
vector< float > best_nearest_neighbors (
            Problem * problem )
```

Best nearest neighbors.

The best nearest neighbors algorithm.

Parameters

problem	The problem to solve

Returns

The solution

5.7.1.2 nearest_neighbors()

```
vector< float > nearest_neighbors (
            Problem * problem,
            vector< bool > & visited,
            int start = 0 )
```

Nearest neighbors.

The nearest neighbors algorithm.

Parameters

problem	The problem to solve
visited	The visited nodes
start	The starting node

Returns

The solution

5.8 nearest_neighbors.hpp

Go to the documentation of this file.

```
1 #include <vector>
2
3 using namespace std;
4
5
16 vector<float> nearest_neighbors(Problem * problem, vector<bool> & visited, int start=0) {
17   vector<float> result;
18
19   result.push_back(start);
20   visited[start] = true;
21
22   for (int i = 0; i < problem->n_points-1; i++) {
23     float min_dist = INFINITY;
24     float min_dist_idx = 0;
25
26     for (int j = 0; j < problem->n_points; j++) {
27       if (i == j) continue;
28
29       float dist = problem->distance_matrix[result[result.size()-1]][j];
30
31       if (dist < min_dist && !visited[j]) {
32         min_dist = dist;
33         min_dist_idx = j;
34       }
```

```
35     }
36
37     visited[min_dist_idx] = true;
38     result.push_back(min_dist_idx);
39   }
40
41   return result;
42 }
43
44
53 vector<float> best_nearest_neighbors(Problem * problem) {
54   vector<float> result;
55
56   float min_length = INFINITY;
57
58   for (int i = 0; i < problem->n_points; i++) {
59     vector<bool> visited(problem->n_points, false);
60
61     vector<float> temp = nearest_neighbors(problem, visited, i);
62     int curr_len = length(temp, problem->distance_matrix);
63
64     if (curr_len < min_length) {
65       result = temp;
66       min_length = curr_len;
67     }
68   }
69
70   return result;
71 }
```

5.9 src/solvers/two_dot_five_opt.hpp File Reference

```
#include <tuple>
#include <vector>
#include <algorithm>
```

Functions

- tuple< vector< float >, int > two_dot_five_opt_step (vector< float > solution, Problem *problem)

  Two-dot-five-opt algorithm step.
- vector< float > two_dot_five_opt (vector< float > solution, Problem *problem)

  Two-dot-five-opt algorithm.

5.9.1 Function Documentation

5.9.1.1 two_dot_five_opt()

```
vector< float > two_dot_five_opt (
            vector< float > solution,
            Problem * problem )
```

Two-dot-five-opt algorithm.

Perform the two-dot-five-opt algorithm on a solution.

Parameters

solution	Solution to be optimized
problem	Problem to be solved

Returns

The optimized solution

5.9.1.2 two_dot_five_opt_step()

```
tuple< vector< float >, int > two_dot_five_opt_step (
            vector< float > solution,
            Problem * problem )
```

Two-dot-five-opt algorithm step.

Perform a single step of the two-dot-five-opt algorithm.

Parameters

solution	Solution to be optimized
problem	Problem to be solved

Returns

Tuple containing the new solution and the total gain

5.10 two_dot_five_opt.hpp

Go to the documentation of this file.

```
1 #include <tuple>
2 #include <vector>
3 #include <algorithm>
4
5 using namespace std;
6
7
17 tuple<vector<float>, int> two_dot_five_opt_step(vector<float> solution, Problem * problem) {
18   int gain = 0;
19
20   int best_i = 1;
21   int best_j = 2;
22
23   int method = -1;
24   bool swapped = false;
25
26   vector<float> new_solution = solution;
27
28   for (int i = 1; i < solution.size()-2; i++) {
29     for (int j = i+1; j < solution.size()-1; j++) {
30       // 2-opt gain
31       int old_two_opt_len = problem->distance_matrix[new_solution[i]][new_solution[i-1]]
32                           + problem->distance_matrix[new_solution[j]][new_solution[j+1]];
33       int new_two_opt_len = problem->distance_matrix[new_solution[j]][new_solution[i-1]]
34                           + problem->distance_matrix[new_solution[i]][new_solution[j+1]];
```

```
35
36        int two_opt_gain = - old_two_opt_len + new_two_opt_len;
37
38        // Node shift 1 gain
39        int old_node_shift_1_len = problem->distance_matrix[new_solution[i]][new_solution[i-1]]
40                                 + problem->distance_matrix[new_solution[i]][new_solution[i+1]]
41                                 + problem->distance_matrix[new_solution[j]][new_solution[j+1]];
42        int new_node_shift_1_len = problem->distance_matrix[new_solution[i-1]][new_solution[i+1]]
43                                 + problem->distance_matrix[new_solution[i]][new_solution[j]]
44                                 + problem->distance_matrix[new_solution[i]][new_solution[j+1]];
45
46        int node_shift_1_gain = - old_node_shift_1_len + new_node_shift_1_len;
47
48        // Node shift 2 gain
49        int old_node_shift_2_len = problem->distance_matrix[new_solution[i]][new_solution[i-1]]
50                                 + problem->distance_matrix[new_solution[j]][new_solution[j-1]]
51                                 + problem->distance_matrix[new_solution[j]][new_solution[j+1]];
52        int new_node_shift_2_len = problem->distance_matrix[new_solution[j]][new_solution[i-1]]
53                                 + problem->distance_matrix[new_solution[i]][new_solution[j]]
54                                 + problem->distance_matrix[new_solution[j-1]][new_solution[j+1]];
55
56        int node_shift_2_gain = - old_node_shift_2_len + new_node_shift_2_len;
57
58        if (two_opt_gain < gain || node_shift_1_gain < gain || node_shift_2_gain < gain) {
59          best_i = i;
60          best_j = j;
61          swapped = true;
62
63          if (two_opt_gain < node_shift_1_gain && two_opt_gain < node_shift_2_gain) {
64            method = 1;
65            gain = two_opt_gain;
66          } else if (node_shift_1_gain < two_opt_gain && node_shift_1_gain < node_shift_2_gain) {
67            method = 2;
68            gain = node_shift_1_gain;
69          } else if (node_shift_2_gain < two_opt_gain && node_shift_2_gain < node_shift_1_gain) {
70            method = 3;
71            gain = node_shift_2_gain;
72          }
73        }
74      }
75    }
76
77    if (method == 1) {
78      // Two-opt swap
79      reverse(new_solution.begin() + best_i, new_solution.begin() + best_j + 1);
80    } else if (method == 2) {
81      // Node shift 1 swap
82      vector<float> array_one = {new_solution.begin(), new_solution.begin() + best_i};
83      vector<float> array_two = {new_solution.begin() + best_i + 1, new_solution.begin() + best_j + 1};
84      vector<float> array_three = {new_solution.begin() + best_j + 1, new_solution.end()};
85
86      vector<float> sol = array_one;
87      sol.insert(sol.end(), array_two.begin(), array_two.end());
88      sol.push_back(new_solution[best_i]);
89      sol.insert(sol.end(), array_three.begin(), array_three.end());
90
91      new_solution = sol;
92    } else if (method == 3) {
93      // Node shift 2 swap
94      vector<float> array_one = {new_solution.begin(), new_solution.begin() + best_i};
95      vector<float> array_two = {new_solution.begin() + best_i, new_solution.begin() + best_j};
96      vector<float> array_three = {new_solution.begin() + best_j + 1, new_solution.end()};
97
98      vector<float> sol = array_one;
99      sol.push_back(new_solution[best_j]);
100      sol.insert(sol.end(), array_two.begin(), array_two.end());
101      sol.insert(sol.end(), array_three.begin(), array_three.end());
102
103      new_solution = sol;
104    } else {
105      return make_tuple(solution, false);
106    }
107
108    return make_tuple(new_solution, swapped);
109  }
110
111
121 vector<float> two_dot_five_opt(vector<float> solution, Problem * problem) {
122    bool swapping = true;
123    vector<float> new_solution = solution;
124
125    while (swapping) {
126      auto res = two_dot_five_opt_step(new_solution, problem);
127      new_solution = get<0>(res);
128      swapping = get<1>(res);
129    }
130
```

```
131     return new_solution;
132 }
```

5.11 src/solvers/two_opt.hpp File Reference

```
#include <tuple>
#include <vector>
#include <algorithm>
```

Functions

- tuple< vector< float >, int > two_opt_step (vector< float > solution, Problem *problem)

  Two-opt algorithm step.
- vector< float > two_opt (vector< float > solution, Problem *problem)

  Two-opt algorithm.

5.11.1 Function Documentation

5.11.1.1 two_opt()

```
vector< float > two_opt (
            vector< float > solution,
            Problem * problem )
```

Two-opt algorithm.

Perform the two-opt algorithm on a solution.

Parameters

solution	Solution to be optimized
problem	Problem to be solved

Returns

The optimized solution

5.11.1.2 two_opt_step()

```
tuple< vector< float >, int > two_opt_step (
            vector< float > solution,
            Problem * problem )
```

Two-opt algorithm step.

Perform a single step of the two-opt algorithm.

Parameters

solution	Solution to be optimized
problem	Problem to be solved

Returns

Tuple containing the new solution and the total gain

5.12 two_opt.hpp

Go to the documentation of this file.

```
1 #include <tuple>
2 #include <vector>
3 #include <algorithm>
4
5 using namespace std;
6
7
17 tuple<vector<float>, int> two_opt_step(vector<float> solution, Problem * problem) {
18   int gain = 0;
19
20   int best_i = 1;
21   int best_j = 2;
22
23   bool swapped = false;
24
25   vector<float> new_solution = solution;
26
27   for (int i = 1; i < solution.size()-2; i++) {
28     for (int j = i+1; j < solution.size()-1; j++) {
29       int old_link_len = problem->distance_matrix[new_solution[i]][new_solution[i-1]]
30                          + problem->distance_matrix[new_solution[j]][new_solution[j+1]];
31       int new_link_len = problem->distance_matrix[new_solution[j]][new_solution[i-1]]
32                          + problem->distance_matrix[new_solution[i]][new_solution[j+1]];
33
34       int gain_value = - old_link_len + new_link_len;
35
36       if (gain_value < gain) {
37         best_i = i;
38         best_j = j;
39         swapped = true;
40         gain = gain_value;
41       }
42     }
43   }
44
45   reverse(new_solution.begin() + best_i, new_solution.begin() + best_j + 1);
46
47   return make_tuple(new_solution, swapped);
48 }
49
50
60 vector<float> two_opt(vector<float> solution, Problem * problem) {
61   bool swapping = true;
62   vector<float> new_solution = solution;
63
64   while (swapping) {
65     auto res = two_opt_step(new_solution, problem);
66     new_solution = get<0>(res);
67     swapping = get<1>(res);
68   }
69
70   return new_solution;
71 }
```

5.13 src/utils/check_in.hpp File Reference

```
#include <vector>
```

Functions

- bool check_in (vector< float > result, float j)

  Check if the given number is in the given vector.

5.13.1 Function Documentation

5.13.1.1 check_in()

```
bool check_in (
            vector< float > result,
            float j )
```

Check if the given number is in the given vector.

Parameters

result	The vector to check.
j	The number to check.

Returns

true The number is in the vector.

false The number is not in the vector.

5.14 check_in.hpp

Go to the documentation of this file.

```
1 #include <vector>
2
3 using namespace std;
4
5
14 bool check_in(vector<float> result, float j) {
15   for (auto i : result) {
16     if (i == j) return true;
17   }
18
19   return false;
20 }
```

5.15 src/utils/distance.hpp File Reference

```
#include <cmath>
#include <vector>
```

Functions

- int distance (vector< float > p1, vector< float > p2)

  Compute the Euclidean distance between two points.

5.15.1 Function Documentation

5.15.1.1 distance()

```
int distance (
            vector< float > p1,
            vector< float > p2 )
```

Compute the Euclidean distance between two points.

Parameters

p1	The first point
p2	The second point

Returns

The Euclidean distance

5.16 distance.hpp

Go to the documentation of this file.

```
1 #include <cmath>
2 #include <vector>
3
4 using namespace std;
5
6
14 int distance(vector<float> p1, vector<float> p2) {
15   int x1 = p1[1];
16   int y1 = p1[2];
17   int x2 = p2[1];
18   int y2 = p2[2];
19
20   return round(sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2)));
21 }
```

5.17 src/utils/length.hpp File Reference

```
#include <vector>
```

Functions

- int length (vector< float > solution, vector< vector< float > > distance_matrix)

  Length of tour.

5.17.1 Function Documentation

5.17.1.1 length()

```
int length (
            vector< float > solution,
            vector< vector< float > > distance_matrix )
```

Length of tour.

This function computes the length of a given tour.

Parameters

solution	The tour
distance_matrix	The distance matrix

Returns

The length of the tour

5.18 length.hpp

Go to the documentation of this file.

```
1 #include <vector>
2
3 using namespace std;
4
5
15 int length(vector<float> solution, vector<vector<float> > distance_matrix) {
16   int length = 0;
17   int starting_node = solution[0];
18   int from_node = starting_node;
19
20   for (int i = 1; i < solution.size(); i++) {
21     int to_node = solution[i];
22     length += distance_matrix[from_node][to_node];
23     from_node = to_node;
24   }
25
26   length += distance_matrix[starting_node][from_node];
27
28   return length;
29 }
```

5.19 src/utils/matrices.hpp File Reference

```
#include <vector>
```

Functions

- vector< vector< float > > m_transpose (vector< vector< float > > b)

  Computes the transpose of a matrix.
- vector< vector< float > > m_sum (vector< vector< float > > a, vector< vector< float > > b)

  Compute the sum of two matrices.

5.19.1 Function Documentation

5.19.1.1 m_sum()

```
vector< vector< float > > m_sum (
            vector< vector< float > > a,
            vector< vector< float > > b )
```

Compute the sum of two matrices.

Parameters

a	The first matrix
b	The second matrix

Returns

The resulting matrix

5.19.1.2 m_transpose()

```
vector< vector< float > > m_transpose (
            vector< vector< float > > b )
```

Computes the transpose of a matrix.

Parameters

b	The matrix to transpose

Returns

The transposed matrix

5.20 matrices.hpp

Go to the documentation of this file.

```
1 #include <vector>
2
3 using namespace std;
4
5
12 vector<vector<float> > m_transpose(vector<vector<float> > b) {
13     vector<vector<float> > trans_vec(b[0].size(), vector<float>(b[0].size()));
14
15     for (int i = 0; i < b.size(); i++) {
16       for (int j = 0; j < b[i].size(); j++) {
17         if (trans_vec[j].size() != b.size()) {
18           trans_vec[j].resize(b.size());
19         }
20
21         trans_vec[j][i] = b[i][j];
22       }
23     }
24
25     return trans_vec;
26 }
27
28
36 vector<vector<float> > m_sum(vector<vector<float> > a, vector<vector<float> > b) {
37   vector<vector<float> > result(a[0].size(), vector<float>(a[0].size()));
38
39   for (int i = 0; i < a.size(); i++) {
40     for (int j = 0; j < a[i].size(); j++) {
41       result[i][j] = a[i][j] + b[i][j];
42     }
43   }
44
45   return result;
46 }
```

5.21 src/utils/split.hpp File Reference

```
#include <vector>
#include <string>
#include <sstream>
```

Functions

- vector< string > split (string s, string delimiter)

  Splits a string into a vector of strings given a delimiter.

5.21.1 Function Documentation

5.21.1.1 split()

```
vector< string > split (
            string s,
            string delimiter )
```

Splits a string into a vector of strings given a delimiter.

Parameters

s	The string to split`
delimiter	The delimiter

Returns

vector<string>

5.22 split.hpp

Go to the documentation of this file.

```
1 #include <vector>
2 #include <string>
3 #include <sstream>
4
5 using namespace std;
6
7
15 vector<string> split(string s, string delimiter) {
16   size_t pos_end;
17   size_t pos_start = 0;
18   size_t delim_len = delimiter.length();
19
20   string token;
21   vector<string> res;
22
23   while ((pos_end = s.find (delimiter, pos_start)) != string::npos) {
24     token = s.substr (pos_start, pos_end - pos_start);
25     pos_start = pos_end + delim_len;
26     res.push_back (token);
27   }
28
29   res.push_back (s.substr (pos_start));
30   return res;
31 }
```

Index