TRAVELLING SALESMAN PERSON

Unidade curricular: Desenho de Algoritmos

Docente: Pedro Diniz

Descintes:

Up202108882 – André Teixeira

Up202208158 – Luciano Ferreira

MAIN FILES

- connection.cpp
- connection.h
- FileReaderToy.cpp
- FileReaderToy.h
- Graph.h
- main.cpp
- Menu.cpp
- Menu.h
- place.cpp
- place.h
- README.md
- Tarefas.h
- worldGraph.cpp
- worldGraph.h

WorldGraph.h -- Main graph we use, na extention of graph.h

Place.h -- Vertex's content (used as a node)

Connection.h -- Edge's content (used as na edge)

Menu.h -- Graphical interface

FileReader.h -- File parsing

Tarefas.h -- Main place where we store our solving algorithms

BACKTRACKING ALGORITHM

```
std::vector<Place> tspBacktrack(WorldGraph& graph) {

    /**
    *@return returs the minimal path using a backtrack algorithm
    *@param graph the graph we will be using to find its mimimum path
    Complexity O(V·V·V!)*/

    std::vector<Place> min_path;
    double min_weight = INT_MAX;

    for (auto& a :Vertex<Place> *& : graph.getVertexSet()) {
        a->setVisited( v: false);
    }

    std::vector<Place> path{ graph.get_place( id: 0) };
    backtrack( &: graph,  place: graph.get_place( id: 0),  &: path,  weight: 0,  &: min_weight,  count: 1,  &: min_path);

    return min_path;
}
```

For a small Graph: ex:Stadium.csv
-- 10 nodes – 55 edges

It takes 13 seconds to run.

Fora a bigger Graph: ex: ExtraFullyConnectedGraphs
--24 nodes – 300 edges

It takes (we gave up).

After seeing this, we actually saw that this way of solving the problem is unfeasabe on gigger graphs do to its high computer demand.

TRIANGULA APROXIMATION

```cpp
std::vector<int> tsp_triangular_aprox(WorldGraph& graph)
{
    vector<int> triangular_aprox;
    std::unordered_map<int,int> hamiltonian_path;

    WorldGraph MST_graph = PrimMST(graph);
    MST_graph.set_all_unvisited();
    Vertex<Place> * first_place = MST_graph.findVertex(MST_graph.get_place(0));
    std::vector<int> pre_order_walk = preorderWalk(MST_graph, first_place);
    for(auto place : MST_graph.getVertexSet())
    {
        hamiltonian_path[place->getInfo().get_id()] = 0;
    }
    for(auto place : pre_order_walk){
        if(hamiltonian_path[place] == 0){
            hamiltonian_path[place] = 1;
            triangular_aprox.push_back(place);
        }
    }
    triangular_aprox.push_back(0);
    return triangular_aprox;
}
```

For a small Graph: ex:Stadium.csv
-- 10 nodes – 55 edges

It takes 0.6 seconds to run.

Fora a bigger Graph: ex: ExtraFullyConnectedGraphs
--24 nodes – 300 edges

It takes 0.6 seconds to run – 1134306 (m.u.)

As we saw these method is way more efficient in bigger graphs and easy to implement. However due to its speed the sollution can still be a some what far from the precise solution.

SIMULATED ANNEALING HEURISTIC

The Simulated Annealing Heuristic
Even though this heuristic uses to much of the random, it allows us to avoid local minima so we can get closer to the actual solution.
Its performance and efficiency depend on how well the parameters are set.

```
std::vector<int> tsp_simulated_annealing(WorldGraph& graph, double initial_temperature, double cooling_rate, int num_iterations)
{
    graph.set_all_unvisited();
    std::vector<int> current_solution = tsp_triangular_aprox(graph);
    double current_distance = calculate_total_distance(graph, current_solution);
    std::vector<int> best_solution = current_solution;
    double best_distance = current_distance;
    double current_temperature = initial_temperature;
    for(auto i = 0; i < num_iterations; ++i){
        std::vector<int> new_solution = current_solution;
        int index1 = std::rand() % (new_solution.size() - 2) + 1;
        int index2 = std::rand() % (new_solution.size() - 2) + 1;
        std::swap(new_solution[index1], new_solution[index2]);
        double new_distance = calculate_total_distance(graph, new_solution);
        double delta_distance = new_distance - current_distance;
        if (delta_distance < 0 || std::exp(-delta_distance / current_temperature) > static_cast<double>(std::rand()) / RAND_MAX){
            current_solution = new_solution;
            current_distance = new_distance;
        }
        if (new_distance < best_distance){
            best_solution = new_solution;
            best_distance = new_distance;
        }
        current_temperature *= cooling_rate;
    }
    return best_solution;
}
```

Fora a Graph like : ex: ExtraFullyConnectedGraphs
--24 nodes – 300 edges

It takes 13 seconds to run –1118039 (m.u.)

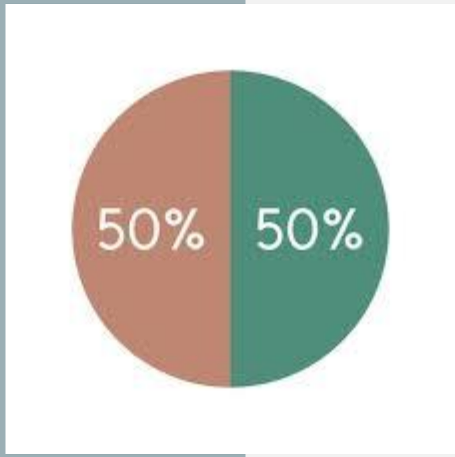
As we can see it actually gets us a better solution than the triangular aproximation, altough it takes some more time.

DIFICULDADES PRINCIPAIS

The optimization and the efficiency of our algorithms.

Participation

André



Luciano