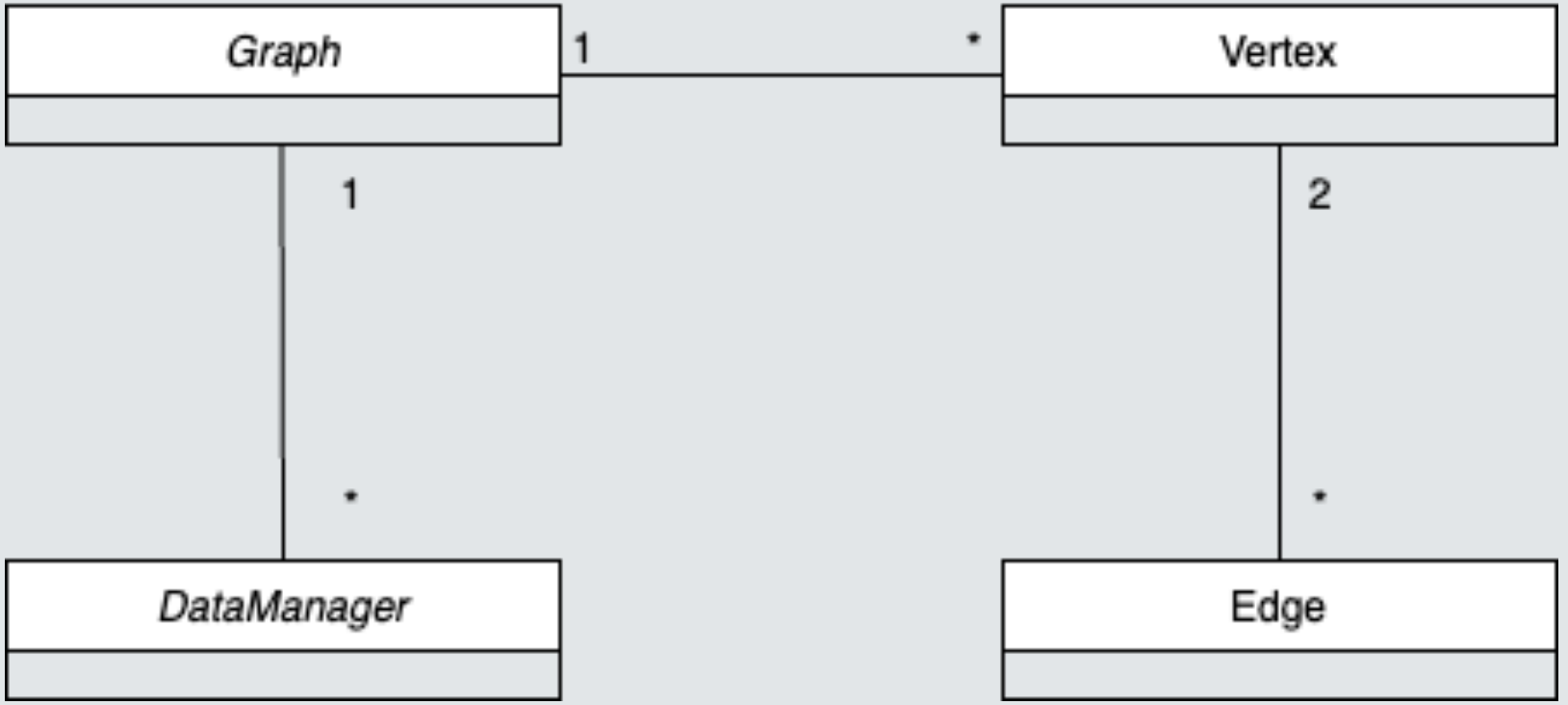
Ocean Shipping and Urban Deliveries

Desenho de Algoritmos

Gonçalo Marques up202206205
Miguel Duarte up202206102
Tómas Teixeira up202208041

Index

Class Diagram



Reading the Dataset

origem	destino	distancia

origem	destino	distancia	label origem	label destino

id	longitude	latitude

Backtracking

A brute force approach to get the optimal solution.

```
double DataManager::tspBacktracking() {
    g->resetAll();
    std::vector<Vertex*> path, bestPath;
    Vertex* origin = g->getVertex( id: 0);
    path.push_back(origin);
    origin->setVisited( flag: true);
    double res = FLT_MAX;

    tspBacktrackingAux( count: 1, currentVertex: origin, cost: 0, &: res, &: path, &: bestPath);

    return res;
}
```

Triangular approximation

A less expensive approach than backtracking but not feasible for big graphs.

```
double DataManager::tspApproximation() const {
    g->resetAll();

    std::vector<Vertex*> mst = g->prim();

    return getTourDistance( tour: mst);
}
```

Simulated annealing with 2-opt approximation

- By using simulated annealing, we were able to improve the accuracy of the nearest neighbor heuristic, despite having a longer running time.

```cpp
std::vector<Vertex*> DataManager::optimizeTour(std::vector<Vertex*>& tour) const{
    std::vector<Vertex*> bestTour = tour;
    double currentDistance = getTourDistance(tour);
    double bestDist = currentDistance;
    double currentTemp = 5000;
    double stopTemp = 1;

    while (currentTemp > stopTemp) {
        int index1 = rand() % tour.size();
        int index2 = rand() % tour.size();

        if (index1 == index2 || index1 == 0 || index2 == 0) continue;
        if (index1 > index2) std::swap( &: index1,  &: index2);

        std::reverse( first: tour.begin() + index1,  last: tour.begin() + index2 + 1);
        double newDist = getTourDistance(tour);

        if (newDist < currentDistance || ((double)rand() / RAND_MAX) <= exp((currentDistance - newDist) / currentTemp)) {
            currentDistance = newDist;
            if (newDist < bestDist) {
                bestTour = tour;
                bestDist = newDist;
            }
        } else {
            std::reverse( first: tour.begin() + index1,  last: tour.begin() + index2 + 1);
        }

        currentTemp *= 0.995;
    }

    return bestTour;
}
```

Nearest neighbor

A greedy approach that opts for the shortest non-visited path.

```cpp
std::vector<Vertex*> DataManager::getNearestNeighbourTour(bool connected, const unsigned int& start) const {
    g->resetAll();
    Vertex* current = g->getVertex( id: start);
    std::vector<Vertex*> tour;
    tour.push_back(current);

    current->setVisited( flag: true);

    for(size_t i = 1; i < g->getVertices().size(); i++) {
        current = findNearestNeighbour( v: current, connected);
        if(current == nullptr) {
            return {};
        }
        current->setVisited( flag: true);
        tour.push_back(current);
    }

    return tour;
}
```

Highlights

- The main highlight is the simulated annealing algorithm. We opted for that approach because of its performance and because we found it interesting.

Individual contribution

- Everyone worked equally, in every task. As the project was not very extensive, we opted to work in every task, so that we could take the most out of the product.