Project artificial intelligence 2018-2019

Olivier Van den Eede[1], Dries Kennes[1]

[1]KU Leuven - De Nayer

1 Problem description

The Cambio car sharing project lets you make a reservation for a car at a specific time in a specific zone. The goal of this optimization problem is to find the most optimal location for each car, in order to assign as many reservations as possible in one city.

The city has been divided into zones, every car has a vehicle type and each reservation consists of:

- A required zone;
- A start and end date and time;
- A type of vehicle.

This means a request can only be assigned to the required zone, or a neighboring zone. Only cars of the right type can be used for a reservation. And of-course a reservation can only be assigned if a car is free for the entire duration of the reservation.

The goal is to assign each car to a zone, and assign each reservation to a car. No limit has been specified for the amount of cars assigned to one zone.

The cost of this problem has to be as low as possible, and can be calculated as shown in formula 1 where P_1 and P_2 are penalty factors.

$$cost = P_1 * \#Notassigned + P_2 * \#Assignedtoneighbor \tag{1}$$

2 Our approach

2.1 Data structures and representation

2.1.1 Request, Zone and Car

Requests and zones are parsed into objects of their respective class containing all fields necessary to store the input data in basic types like integers, strings and lists. Cars do not have a custom object, they are represented with a simple string on the request object.

After parsing the input data, all request and zone objects are stored in 2 lists, and in a dictionary for fast lookup by id.

2.2 Solution

A solution is represented by a Solution object. This object contains a reference to the lists with parsed input data, and 2 dictionary's. These dictionary's represent the relationship between a car and zone, and between a request and a car. The cost function can be calculated based on the car_zone and req_car dictionary's.

2.3 Auxiliary data structures

To speed up the calculation process, we created some extra data structures with pre-calculated data.

The first structure is the overlap matrix. This 2D-matrix represents the overlapping of 2 requests, this allows for a fast feasibility check because the slow overlap checking loop is pre-calculated.

2.4 Algorithm

Our local search algorithm is composed of 3 major parts.

2.4.1 Greedy assign

The greedy assign function loops over the currently unassigned requests, and try's to find a car already assigned to the right zone or a neighboring zone.

If a suitable car is found, the request will be assigned. If no suitable car is available, the function will take a new car and assign it to the required zone and request.

This function is used to create our initial solution, and guaranties a feasible solution. After each of our local moves, this function in called to try and fill all gaps and assign as many requests as possible.

2.4.2 Local moves

Our algorithm is based around 5 small local changes:

- Move a request from the 'optimal' zone to one of its neighbors
- Move a request assigned to a neighbor zone to the 'optimal' zone.
- Swap the car assigned to a request with another car in the same zone.
- Unassign 1 car from all its requests.
- Unassign 1 request.

2.5 Meta heuristic

To control all small local changes, we implemented Simulated Annealing. This allows our algorithm to find a different solution in the neighborhood of the working solution. This new solution will be accepted if the cost is lower, and accepted or rejected with a random probability based on the simulated annealing factors.

When the stop-condition for our algorithm is reached, a new random initial solution will be generated and start again.

3 Results

4 Future work