Google Hash Code 2018: An Approach

Alessio Quercia

Abstract—This paper is intended to show a possible approach to the Google Hash Code 2018 Online Qualification round problem, that is an optimization problem to be solved within 4 hours. This constraint has been ignored, given that this approach has been developed after the competition for academic purposes. The main goal was to solve the problem in an efficient way, using an heuristic (or metaheuristic) approach. To achieve this goal, different greedy approaches have been developed and compared by looking at their results. Then, the best performing one has been used as a base to build a GRASP (Greedy Randomized Adaptive Search Procedure) heuristic approach. Indeed, randomness has been added to the greedy in the choice criterion, allowing the algorithm to make choices that are not always the best ones.

1 Introduction

Google Hash Code is a team programming competition requiring participants to solve an engineering problem within four hours. Each year Google launches a new Hash Code, proposing a new problem, which is often an optimization problem to solve in an efficient way.

The time constraint has been ignored, given that this approach has been developed a year after the competition took place.

The paper is organized in three main parts: the first one contains a detailed description of the problem, the second one introduces the two main approaches (a greedy and its GRASP version) used during this project, together with other minor approaches and the choice criteria adopted to efficiently solve the task. In the last part, composed by three sections, the experiments and their results are shown and discussed.

2 Problem description

Google Hash Code 2018 Online Qualification Round task was about Self-driving rides. The main goal was to look for an efficient way to get some commuters to their destinations with a fleet of self-driving vehicles in a simulated city (represented by a grid).

2.1 Input

The input is read from a file, containing the following integer numbers:

- R, the number of rows of the grid;
- C, the number of columns of the grid;
- F, the number of vehicles in the fleet;
- N, the number of pre-booked rides;
- B, the bonus for starting the ride on time;
- T, the number of steps in the simulation.

The subsequent N lines describe the pre-booked rides, containing the following integer numbers:

- a, the row of the starting point;
- b, the column of the starting point;
- x, the row of the destination point;
- y, the column of the destination point;
- s, the earliest start, that is the earliest step in which the ride can start;
- f, the latest finish, that is the latest step by which the ride must finish to get points for it.

2.2 Scoring

Each ride completed before its latest finish earns the number of points equal to the distance between the start intersection and the finish intersection, where the distance **d** from a point (a, b) to a point (x, y) is computed as 1-norm distance:

$$d = |a - x| + |b - y| \tag{1}$$

Additionally, each ride which started exactly in its earliest allowed start step gets an additional timeliness bonus of **B**. The total score is the sum of all points earned by all rides completed by all vehicles.

3 Method

In the following subsections the needed structures, the main greedy and heuristic approaches and the choice criteria tested during the experiments are described.

3.1 Structures

Some structures have been useful to keep all the information needed during the simulations:

- DIST[v][r], the matrix containing the distances from each vehicle v to each ride r;
- REW[v][r], the matrix containing the total reward that each vehicle v could obtain by making each ride r;
- BEST[v][K], the matrix containing for each vehicle, the first K best 3-uples according to the adopted choice criterion, containing: a value, the id of the ride having that value, and the time to complete that ride.

Alessio Quercia, Heuristic Algorithms, A/A 2017-2018, Università degli Studi di Milano, via Celoria 18, Milano, Italy
E-mail: alessio.quercia@studenti.unimi.it

3.2 Greedy approaches

Two greedy approaches have been tested during this project. Each approach shares a common core structure:

1) Load the samples;
2) For each sample
    a) Instantiate and initialize the required structures;
    b) Run the simulation;
    c) Store the output;
    d) Compute the sample score;
    e) Compute the sample execution time;
3) Compute the total score;
4) Compute the total execution time;

The main difference between the two tested greedy approaches lies on the implementation of the method to run the simulation. Indeed, in the first approach the simulation was dealt as a game loop over the total time T, whereas in the second approach each vehicle is computed separately.

3.2.1 Game loop greedy

This approach is similar to a game loop over the total time T, where, before starting the loop, the best ride is assigned to each vehicle, and then, at each step t:

1) If a vehicle completed a ride, update the structures in the positions concerning it and on the available rides;
2) If a vehicle completed a ride and if it is possible, assign it a new (best) ride;
3) Make a 1-step move;

This approach has been replaced by a new and better performing one, that is another greedy algorithm, which considers each vehicle separately.

3.2.2 Separate vehicles greedy

In this approach, each vehicle is considered separately, until all vehicles are considered:

1) For each vehicle:
    a) The current step is initialized to 0;
    b) Update the structures in the positions concerning it and on the available rides;
    c) Until it is possible to assign a new (best) ride:
        i) Assign it and update the current step;
        ii) Make a n-steps move (to complete the assigned ride);
        iii) Update the structures in the positions concerning it and on the available rides;

The inner loop cycles until it is possible to assign a new ride to the considered vehicle, that is until there is still time to make the ride assigned from the BEST matrix according to the choice criterion. If there are no more feasible and available rides, the inner loop ends and the algorithm considers the next vehicle (until all the vehicle are considered).

This approach resulted to have a better performance over the provided samples, with respect to the Game Loop Greedy.

3.3 Choice criteria

Different choice criteria have been tested to fill the BEST matrix, containing the best K rides for each vehicle, as explained above.

Each choice criterion, in order to fill the BEST matrix with the best K rides checks each available ride: if the considered ride is feasible, that is its required time summed up to the vehicle current step is less than the total time T and the ride can be finished on time (before the latest finish f), the ride can be done, otherwise, the ride is not considered. In this way, each choice criterion makes sure that the rides filled in the BEST matrix are feasible rides, that is they can be done in time. If there is no feasible ride for the considered vehicle, its row in the BEST matrix is filled with rides having index -1, allowing the assign method to understand that it is not possible to assign another ride.

The best performing choice criteria are described in the following subsections.

3.3.1 Maximizing the reward

The first naive idea was to always assign to a vehicle the available and feasible ride with the highest reward, hoping to achieve the maximum possible reward for each sample. In this case, the BEST matrix was filled for each vehicle with the K rides having the highest rewards in descending order, in such a way to have the ride with the best possible reward in the first position.

This choice criterion resulted to be excessively greedy, indeed, by always assigning the ride having the highest reward, the required times to complete the rides were ignored. Indeed, the choice criterion prioritizes longer rides (given that the reward is the ride distance summed up to the bonus, in case the ride started on time) and therefore less rides are effectively done at the end.

3.3.2 Minimizing the required time

Given that focusing on the reward resulted to be too much of a greedy choice, the second idea was to focus on the rides that could be finished on time with the lowest required time. The required time was computed as the sum of the distance from the vehicle to the ride starting point, the (possible) time to wait, and the distance from the ride starting point to the ride destination point. In this case, the BEST matrix was filled for each vehicle with the K rides having the lowest required time in ascending order, in such a way to have the rides with the best possible required time in the first position.

This choice criterion allowed to complete more shorter rides with respect to the criterion maximizing the reward. However, the results were similar and still not good enough.

3.3.3 Maximizing a utility function

Assigning rides to vehicles according to an auxiliary function resulted to be more efficient. Instead of simply maximizing the reward or minimizing the required time, the maximization of an auxiliary utility function computed as their difference (minus the amount of time the vehicle arrived late) led to better results. In fact, the utility function u associated to each couple (vehicle, ride) was computed as follows:

$$u = reward - required\_time - late\_time \quad (2)$$

where *reward* is the achievable reward by the vehicle by doing that ride, *required_time*, the total time needed to complete that ride and *late_time* the amount of steps the vehicle is late with respect to the ride earliest start (0, if it is on time or early; greater than 0, otherwise).

In this case, the BEST matrix was filled for each vehicle with the K rides having the highest utility value in descending order, in such a way to have rides with the best possible utility value in the first position.

The idea behind this criterion was to assign the ride having the best ratio quality/price, where the quality is the reward and the price is the required time to complete the ride. Before applying this criterion, other mixed criteria have been tested. Some of these mixed criteria are briefly described in the next subsection.

3.3.4 Mixed criteria

After having tested the maximization of the reward and the minimization of the required time criteria, the immediate resulting ideas was trying to mix them in such a way to fill the BEST matrix for each vehicle with:

- the best K rides having the minimum required time, among the rides having the highest rewards, that is, maximizing the reward first, and, in case of equality, minimizing the required time;
- the best K rides having maximum reward, among the rides having the lowest required time, that is, minimizing the required time first, and, in case of equality, maximizing the reward;
- the best K rides having the minimum waiting time, among the rides having the highest rewards, among the rides having the lowest required time, that is, minimizing the waiting time first, maximizing the reward next and minimizing the required time as last criterion;
- the best K rides having the minimum waiting time, among the rides having the lowest required time, among the rides having the highest reward, that is, minimizing the waiting time first, minimizing the required time next and maximizing the reward as last criterion;
- the best K rides having the lowest required time, among the rides having the highest utility value, that is, maximizing the utility function (2) first, and, in case of equality, minimizing the required time.

This last mixed criterion was the best performing one. Indeed, it is just an improvement of the criterion which simply maximizes the utility function, described in the previous subsection.

3.4 GRASP approach

GRASP (Greedy Randomized Adaptive Search Procedure) is an iterative randomized sampling technique consisting of two phases: a construction phase and a local search phase. In the first phase, an initial solution is built using an adaptive randomized greedy function, while in the second phase a local search procedure is applied to the constructed solution in hope of finding and improvement. At the end of the process, the overall best solution is returned as a result.

At each construction iteration, an element has to be added to the solution, therefore a choice has to be made. Indeed, all the elements are ordered in a candidate list (which can be restricted to a list containing the first K best elements) according to a greedy function, which measures the benefit of each element. GRASP is adaptive because the benefits associated to the elements are updated at each iteration to reflect the changes brought by the selection of the previous element. The random component is in the choice itself of the element. Indeed, the selected element is not always the best in the candidate list, as a greedy algorithm would do, but it is selected randomly from the restricted candidate list containing the K best elements according to the greedy function.

The solution built during the construction phase is not guaranteed to be local optimal. For this reason it is suggested to apply a local search to attempt to improve the solution [1].

The GRASP version adopted for the best greedy algorithm (the separate vehicle greedy) described above has no local search phase, because it would have required much more execution time per solution (for a probably small improvement) and because it was hard to define a neighborhood for a given solution. Its general scheme is described in the following list:

1) Load the samples
2) For TIMES times:
    a) For each sample:
        i) Reset the sample to its original form
        ii) Run the randomized greedy algorithm
    b) If the solution is better than the best solution, update the best solution

The randomized version of the greedy algorithm adds randomization in the selection of the element, as described for the GRASP construction phase. To prioritize the selection of the first best element, the random selection is not made in a uniform way. Indeed, the index *i* of the element to select in the restricted candidate list is computed with the following formula:

$$i = r^q \times K \tag{3}$$

where *r* is a random real number between 0 and 1 is selected uniformly, *q* is a constant integer number and *K* is the restricted candidate list size. In this way, the greater is the value of *q*, the more the algorithm prioritizes the selection of the very best element (that is the first one in the restricted candidate list).

By tuning the algorithm parameters (*TIMES*, *K* and *q*), it was in general possible to achieve slightly better results with respect to the greedy.

Another GRASP version has then been tested by also adding randomness in the choice of the choice criterion itself. Indeed, in this version three different criteria have been considered, including the best performing one. Even if the choice of the best criterion was prioritized, the overall results achieved using this approach were not good enough to compete with the greedy version.

4 Simulations and experiments

Many experiments have been made to test which was the best performing greedy algorithm and which was the best criterion to select the rides to assign to the vehicles. Few experiments were needed to understand which GRASP version was better performing, while more tests have been required to achieve an improvement using the GRASP version with respect to the greedy version.

The system architecture and the implementation details used for the experiments are described in the following subsections, while the results are discussed in the next section.

4.1 System architecture

I ran the simulations on my laptop, characterized by an Intel Core i7-6700HQ CPU, a Nvidia Geforce GTX 950M GPU and 8 GB RAM.

4.2 Implementation details

The algorithms have been implemented in C, mostly using the Standard Library, following the pseudo-codes described above.

5 Results

In this section, the best result for the best performing greedy algorithm and its GRASP version are discussed.

The best greedy algorithm resulted to be the one computing each vehicle separately and assigning the rides according to the mixed criterion which maximizes the utility (2) first and minimizes the required time next, as already described above. Its results for each sample are shown in TABLE 1.

The GRASP version of the best greedy algorithm has been tested by fixing the *TIMES* parameter to 100 and changing the other parameters (K and q) in order to achieve better results than the greedy version. The best results obtained by using this version have been obtained by setting K to 2 and q to 1000, as shown in TABLE 2.

Comparing the results in the two tables, it is possible to see that the GRASP version achieved a slightly better result, by paying about 100 times the execution time of the greedy algorithm. Moreover, Fig. 1 shows that ten over ten different executions (with different seeds) of the GRASP version led to better results with respect to the greedy algorithm, implying that also its mean result is better. This suggests that the GRASP expected result is probably better than the result produced by the greedy version, by setting $TIMES = 100$, $K = 2$ and $q = 1000$.

Sample	Result	Time (s)
a_example	10	0.004
b_should_be_easy	173977	0.018
c_no_hurry	15809027	3.713
d_metropolis	11209219	3.941
e_high_bonus	21068945	3.989
Total	48261178	11.669

TABLE 1: Separate vehicle greedy results.

Sample	Result	Time (s)
a_example	10	0.003
b_should_be_easy	173977	0.015
c_no_hurry	15809650	4.057
d_metropolis	11254423	4.549
e_high_bonus	21099945	4.237
Total	48338005	12.868 1202.337

TABLE 2: GRASP best achieved results by setting $TIMES = 100$, $K = 2$ and $q = 1000$. The total time in the first line is the execution time of the single iteration of the algorithm in which the best result has been achieved, while the total time in the second line indicates the overall execution time of the algorithm.

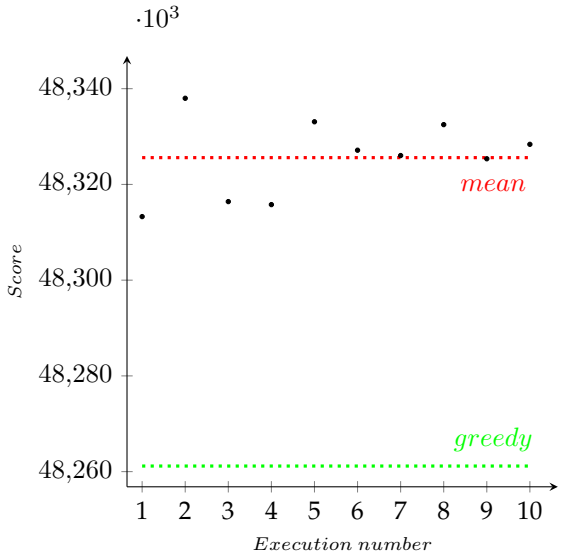

Fig. 1: The points represents the results obtained by running the GRASP version for 10 times, by setting $TIMES = 100$, $K = 2$ and $q = 1000$. The green dotted line indicates the score achieved by the greedy algorithm, whereas the red one depicts the mean score achieved by its GRASP version in 10 executions with different random seeds.

6 Conclusions

It was possible to achieve some good results on the problem instances both by using a greedy algorithm and its GRASP version. This last one, resulted to achieve slightly better overall results, but it required about 100 times the execution time required by the greedy algorithm.

Given that the greedy algorithm already produces good results, the solutions provided by its GRASP version are not considerably improving. Despite that, the GRASP version resulted to be a guarantee over the results, considering that in the worst case the result would have been the same as the the greedy one.

To conclude, it is probably better to use a GRASP approach on a greedy algorithm that is producing results that are still not good enough to achieve better improvements from the greedy to the GRASP version.

References

[1] T. A. Feo and M. G. Resende, "Greedy randomized adaptive search procedures," *Journal of global optimization*, vol. 6, no. 2, pp. 109–133, 1995.