

## 1 Online min-weight matching- The parking game

Premise:

- People input a location they want to go and the app assigns a parking location.
- Visitors' "costs" are the distances from where they park to where they want to go.
- The goal is to assign spots so that people walk as little as possible

We can visualize this in the below image:

Our problem is that we can't move people once they are already parked. In 1.1, purple arrives first

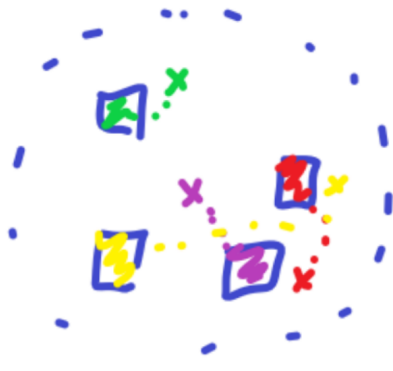


Figure 1.1: The squares are parking spots while x's are locations users want to go to.

and is assigned the closest spot. Then red arrives and wants to go to a location near purple's spot. So they are assigned to the next closest spot. This is continued for yellow and green. Although it minimizes walking distance given the available spots, if we knew where all visitors were going ahead of time, we could use the Hungarian Algorithm <sup>1</sup> to solve it (It is also incentive compatible).

Although we are trying to minimize costs, large costs doesn't necessarily mean we could do better. Here, they have to cross the entire circle, but there was no better (or other) option.

In the "online" version of this problem, We want to minimize our costs compared to the optimal solution. In particular, we want  $(ourcost/(OPTcost))$  to be small. How bad can the *Greedy algorithm* be? In particular, what is its worst case competitive ratio?

Imagine the parking spots are all in a straight line:

First, red arrives at 0. Since  $|1 - 0| < |1.001 - 0|$  they're assigned to the spot at 1. The successive users (orange, yellow, green,...) would like to go wherever the last user was assigned. Since the spot to the right is incrementally closer than the spot at -1.001, the users are assigned to the spot to the right of the location they wish to go. Once there are no other spots, the last user (pink) is assigned to the spot at -1.001.

<sup>1</sup>The Hungarian method, is a combinatorial optimization algorithm that solves the assignment problem in polynomial time .



Figure 1.2: The squares are parking spots while x's are locations users want to go to.

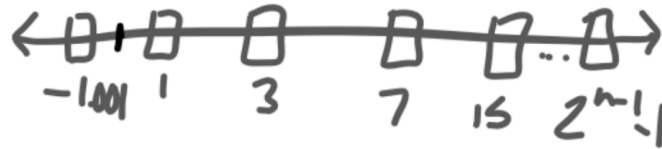


Figure 1.3: The squares are parking spots.

Greedy's cost :  $2 * (2^{2-1} - 1) + 1 = 2^2 - 1$

Opt cost: 1 (if the first person had been assigned to the spot at -1.001 and everyone else parked at the location they want to go to).

Competitive ratio is  $2^2 - 1$ .

Note: The greedy algorithm is incentive compatible.

Referring to figure 1.5, we can see the cost:  $1 + 2(n - 1) = 2n - 1$

Opt's cost: 1

For every deterministic algorithm, I can specify an instance where it has  $CR \geq 2n - 1$

How does Opt change with the addition of each visitor? Let  $Opt_i$  be the optimal solution for the first  $i$  visitors. How "different" are  $Opt_i$  and  $Opt_{i+1}$ , where  $Opt_{i+1}$  is chosen to be the solution that "agrees with"  $Opt_i$  the most? Imagine drawing the lines for the matching of  $Opt_i$  and  $Opt_{i+1}$ , as seen in figure 1.6.

We just showed that in the chain of optimal solutions easily 1 new parking spot is used. The algorithm PERMUTATION is to assign every new driver to this spot.

The PERMUTATION algorithm has a  $CR = 2n - 1$  on every instance.

When  $i = 1$ , we just match the visitor to the closest spot, so  $Cost_i \leq Opt_i$

We'll assume we satisfy  $Cost_i \leq (2i - 1)Opt_i$  for the first  $i$  drivers, and show we still satisfy the  $i + 1st$  driver.

$$\begin{aligned} Cost_{i+1} &= Cost_i + d(r_{i+1} s^*) \\ &\leq Cost_i + Opt_i + Opt_{i+1} \\ &\leq (2i - 1)Opt_i + Opt_i + Opt_{i+1} \end{aligned}$$

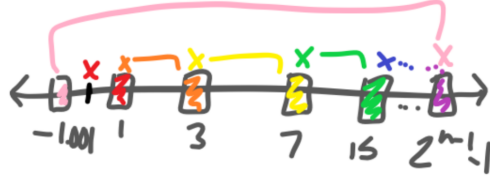


Figure 1.4: The squares are parking spots and x's are locations users want to go to.

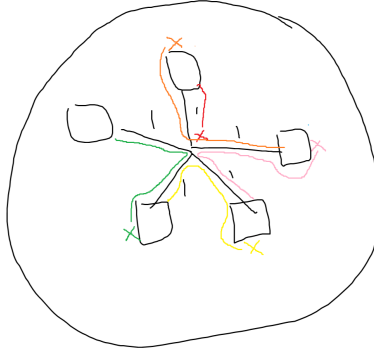


Figure 1.5

$$\begin{aligned} &\leq (2i - 1)Opt_{i+1} + Opt_{i+1} + Opt_{i+1} \\ &\leq (2i + 1)Opt_{i+1} = (2(i + 1) - 1)Opt_{i+1} \end{aligned}$$

It lost incentive compatibility! This algorithm was good, but it doesn't align with incentives.

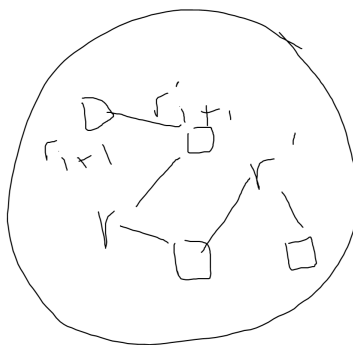


Figure 1.6