Whose Turn Is It to Drive Today?

Brian Lins

I have a long commute, but I am fortunate to share my commute with several colleagues. I highly recommend car pooling: Sharing driving is a great way to save money and resources, to get to know your coworkers, and to keep up with office gossip!

In any car pool, it is important to have a fair procedure for selecting the driver because all the participants benefit, but only the driver pays the cost of driving. When the car pool has many members, deciding who should drive can get complicated. Some people solve this problem by paying the driver gas money, but here we focus on nonmonetary solutions. As we will see, there is an elegant mathematical way to pick the driver.

The five members of our car pool have different schedules, and late meetings or extracurricular activities can change the subset of riders unpredictably from day to day.

For years we used a simple, but naive, approach to determine who would drive. After you rode in a car with someone else driving, you owed the driver a ride. If you rode with the same driver again before giving her a ride, you could owe two or even more rides. We always had to remember who owed rides to whom. Also, we frequently ran into nontransitive situations in which A owed B, B owed C, and C owed A. Whenever possible, we tried to pick drivers so that no one's ride debt got too high.

Let's illustrate with an example. Imagine five friends—Ashley, Brandon, Courtney, Daniel, and Emily—who use our naive car-pool system. The first week Ashley drives everyone except Emily on Monday. On Tuesday and Wednesday, Ashley gets rides from Brandon and Courtney, respectively. On Thursday, Daniel drives Emily (as shown in figure 1).

On Friday, Ashley and Daniel want to carpool together. Ashley says that Daniel should drive, since he still owes her a ride from Monday. This doesn't seem fair to Daniel. He points out that Ashley has already gotten two rides, while he just drove the day before. According to our naive method, Ashley is right and Daniel should drive.

Is this the right decision? Is it fair?

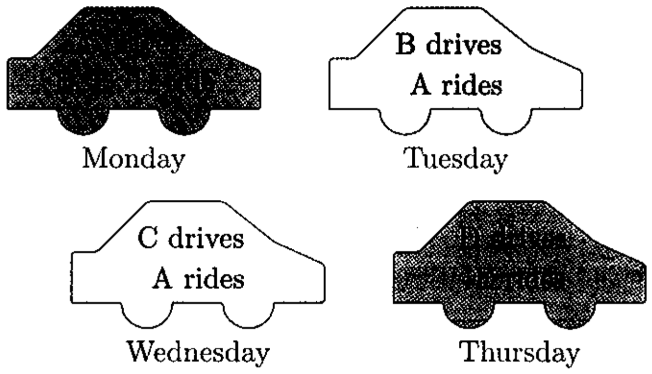

Figure 1. The car-pool schedule Monday through Thursday.

What Is Fair?

In our example, Ashley drives only once and gets three rides. Everyone else gets only one ride. Under this naive scheme, members have an incentive to drive on days when there are a lot of passengers, because it maximizes the number of rides they get in return. Clearly, this is not fair.

After a particularly bad sequence of three-way ties made my own car pool's method frustrating, I finally typed "fair car pool algorithm" into a search engine and found a better way.

In 1983, Ronald Fagin and John H. Williams published "A Fair Carpool Scheduling Algorithm." In the article, they explain what it means for a car pool–scheduling algorithm to be fair. Needless to say, our naive algorithm does not meet their criterion.

Fagin and Williams point out that an ideal solution would have each member drive half of the times that she travels in a car with two people, a third of the times she travels with three, a quarter of the times she travels with four people, and so on. Although this is often impossible, it suggests a simple formula for the ideal number of times that person i in the car pool should have driven:

$$y_i = \sum_{p=2}^{n} \frac{t(i,p)}{p},$$

where n is the number of members in the car pool and $t(i,p)$ is the number of times that person i has traveled in a car with p people.

For example, if a car-pool member travels with two people 20 times, three people nine times, and four people eight times, then she should have driven $\frac{1}{2}(20)+\frac{1}{3}(9)+\frac{1}{4}(8)=15$ times. When deciding if she has driven a fair amount, it doesn't matter how many people were in the car each time she drove. The only thing that matters is whether she has driven 15 times.

Applying this formula to the first four days of our example, we conclude that after Thursday, Ashley, Brandon, Courtney, Daniel, and Emily should have driven 5/4, 3/4, 3/4, 3/4, and 1/2 times, respectively. Notice that Ashley's value is the largest of the five.

Suppose we keep track of the ideal number of times each member should have driven using a vector $\mathbf{y}=(y_1,\ldots,y_n)$, with an entry for each member in the car pool. We can't always expect members to have driven exactly their ideal number of times, after all, as we have seen, the entries of $\mathbf{y}$ typically include fractions. By subtracting $\mathbf{y}$ from the vector $\mathbf{z}=(z_1,\ldots,z_n)$ with entries z_i equal to the total number of times member i has driven, we get a vector $\mathbf{e}=\mathbf{z}-\mathbf{y}=(e_1,\ldots,e_n)$ that we will call the *error vector*. The entries of the error vector tell us how many times each person has driven more or less than his or her ideal share. In our example, the error vector after Thursday (with people listed alphabetically) is $\mathbf{e}=(-\frac{1}{4},\frac{1}{4},\frac{1}{4},\frac{1}{4},-\frac{1}{2})$.

Intuitively, a car-pool algorithm is *fair* if the number of times each participant has driven stays close to the ideal number for that participant; that is, the error vector remains close to the zero vector. More concretely, it is fair if the entries of the error vector are bounded: There exists a constant C such that every day $|e_i| \le C$ for all i.

There are several fair algorithms for selecting drivers in a car pool. Most are not practical, but we will now describe one that is fair and easy to implement.

A Greedy Algorithm

Once you are aware of the error vector, the algorithm proposed by Fagin and Williams seems very natural. To decide whose turn it is to drive, select the participant with the lowest corresponding entry of $\mathbf{e}$. This is the person whose actual number of turns driving has fallen furthest behind his ideal fair number. Ties can be broken using a deterministic rule or a random one—any method will be fair.

This is an example of a *greedy algorithm*. A greedy algorithm is one that makes the locally optimal choice at each stage to try to obtain a globally optimal solution. In this case, we try to make the best decision about who should drive by trying to remedy the largest departure from fairness at each stage.

In our example, the greedy algorithm yields the same drivers during the first four days, but on Friday the driver would be Ashley, not Daniel.

To implement the greedy algorithm, the members of the car pool need keep track of only the error vector. Each time p people ride together, the passengers' errors decrease by $1 / p$, the driver's error increases by $(p-1) / p$, and the errors of the nonparticipants are unchanged. Because the net change in error is always zero, we think of the passengers as paying the driver.

It is not obvious that the greedy car-pool algorithm always leads to a fair solution. It does—if there are n car-pool members, then $|e_i| \leq \frac{1}{2}(n-1)$ for all i—but the proof is a little technical. See the sidebar for details.

In addition to being fair, the greedy car-pool algorithm is robust. If a car pool deviates from the algorithm, say because one of the participants has a flat tire and cannot drive, then over time the error vector returns to a fair state.

An Easy Sell

As the only mathematician in the car pool, I knew that implementing a fancy algorithm might be a tough sell. I didn't want any of my humanities colleagues to feel that I was cheating them with math.

Number of people in car	Driver gets	Each passenger pays
2	30	30
3	40	20
4	45	15
5	48	12

Table 1. To avoid talking about vectors, we use points.

More importantly, I had to convince everyone that the new algorithm would be more convenient than the old way, in addition to being more fair.

A couple of cosmetic tweaks help make the greedy algorithm more palatable to nonmathematicians. To avoid fractions when computing the error vector, multiply $\mathbf{e}$ by the least common multiple of the possible number of people in a car. Because there are five members in our pool, we multiply our error vectors by $60 = \operatorname{lcm}\{2,3,4,5\}$. By doing this, the error vectors always have integer entries.

Another obvious change is to avoid mentioning the

Greedy but Fair

Here we show that the greedy car-pool algorithm is fair. That is, if we follow the algorithm, the errors will be bounded.

Suppose we have n people in the car-pool system, $N = \{1,2,\ldots,n\}$. For any subset $I \subseteq N$ and any vector $\mathbf{x} = (x_1,\ldots,x_n) \in \mathbb{R}^n$, let $\mathbf{x}_I$ denote the sum $\sum_{x \in I} x_i$, where $\mathbf{x}_\varnothing = 0$.

Each proper subset $I \subseteq N$ corresponds to a half-space in $\mathbb{R}^n$ given by the relation $\mathbf{x}_I \leq \frac{1}{2}k(n-k)$, where k is the number of elements in I. The intersection of all these half-spaces with the hyperplane $x_N = 0$ is a convex polytope S.

The bulk of the proof is showing that if we follow the greedy algorithm, the error vector is always in S. Observe that the initial error vector $(0,\ldots,0)$ is in S.

Let $\mathbf{e} = (e_1,\ldots,e_n)$ be an error vector in S, and let $\mathbf{e}' = (e_1',\ldots,e_n')$ be the error vector on the following day, after a set of people P carpool with driver j chosen from P. We must prove that $\mathbf{e}'$ is in S.

First, although the individual error entries change after each car pool, the sum of the errors does not. Thus $\mathbf{e}_N' = 0$.

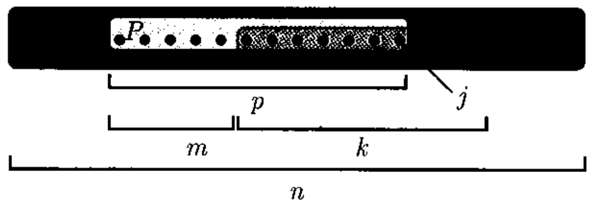

Figure 3. The sets and values in the proof.

Next, let I be a nonempty proper subset of N with k elements. (Figure 3 illustrates the relationships between all the sets and values in the proof.) Let $\Delta_I = \mathbf{e}_I' - \mathbf{e}_I$. If $\Delta_I \leq 0$, then $\mathbf{e}_I' \leq \mathbf{e}_I \leq \frac{1}{2}k(n-k)$, as desired. So we may assume that $\Delta_I > 0$, which can happen only if the driver, j, is in I.

Because $\mathbf{e}$ is in S,

$$\mathbf{e}_{I-\{j\}} \leq \tfrac{1}{2}(k-1)(n-k+1)$$

and

$$\mathbf{e}_{P \cup I} \leq \tfrac{1}{2}(k+m)(n-k-m),$$

where m is the number of elements in $P - I$.

Because the greedy algorithm selected j to be the driver, $e_i \geq e_j$ for all i in P. Therefore,

$$\mathbf{e}_{I \cup P} = \mathbf{e}_{I-\{j\}} + e_j + \mathbf{e}_{P-I} \geq \mathbf{e}_{I-\{j\}} + (m+1)e_j.$$

We can use all these inequalities to obtain a tighter

	Ashley	Brandon	Courtney	Daniel	Emily
Monday	45	−15	−15	−15	
Tuesday	−30	30			
Wednesday	−30		30		
Thursday				30	−30
Total	−15	15	15	15	−30

Table 2. The technique applied.

word "vectors"! In my car pool, we just talk about points. Everyone has a certain number of points, and when we carpool, the passengers pay the driver points according to table 1. Of course, these are simply the fractional transaction costs multiplied by 60.

Here is another way to think about these numbers. The cost of a trip is always 60 points, which is divided equally among all the people in the car, including the driver. If p members drive together on a given day, then each passenger pays $60 / p$ and the driver nets $60(p-1) / p$. Table 2 shows this technique applied to our example. It is easy to see that Ashley should drive Daniel on Friday.

Unlike in our naive algorithm, all the participants benefit equally when there are more people in the car. If a fourth person joins a car with three other people, then the cost for each passenger decreases by five, and the points earned by the driver increase by five. This reduces the incentive to avoid driving when the car isn't full.

A disadvantage of the greedy car-pool algorithm is that it requires some bookkeeping. Fagin and Williams describe using a notebook to keep track of the transfer of points in their car pool. Instead of a paper notebook, my car pool uses a shared online spreadsheet. Everyone in the car pool has access to the file and can easily update the spreadsheet from a computer or phone.

An additional benefit is that we can look up how often we carpool and easily estimate how much money we save as a result. I can't imagine going back to car pooling without the greedy car-pool algorithm. ■

bound on $\mathbf{e}_I$:

$$\begin{aligned}\mathbf{e}_I &= \mathbf{e}_{I-\{j\}} + e_j\\ &= \frac{m}{m+1}\mathbf{e}_{I-\{j\}} + \frac{1}{m+1}(\mathbf{e}_{I-\{j\}} + (m+1)e_j)\\ &\le \frac{m}{m+1}\mathbf{e}_{I-\{j\}} + \frac{1}{m+1}\mathbf{e}_{P\cup I}\\ &\le \tfrac{1}{2}k(n-k) - \tfrac{1}{2}m.\end{aligned}$$

The largest possible value for Δ_I occurs when $I \cap P$ contains only the driver, in which case $\Delta_I = \frac{m}{m+1}$. Thus, we conclude that

$$\begin{aligned}\mathbf{e}_I' &= \mathbf{e}_I + \Delta_I\\ &\le \tfrac{1}{2}k(n-k) - \tfrac{1}{2}m + \frac{m}{m+1}\\ &\le \tfrac{1}{2}k(n-k).\end{aligned}$$

So $\mathbf{e}'$ is in S.

Finally, we will show that the entries are bounded. Let e_i be the error for one of the individuals. Because $\mathbf{e}$ is in S, $e_i = \mathbf{e}_{\{i\}} \le \frac{1}{2}(n-1)$. But also $e_i = -\mathbf{e}_{N-\{i\}} \ge -\frac{1}{2}(n-1)$. So $|e_i| \le \frac{1}{2}(n-1)$.

Further Reading

Fagin and Williams's article "A Fair Carpool Scheduling Algorithm" (*IBM Journal of Research and Development* **27** no. 2 [March 1983] 133–139) is very readable.

Our proof was inspired by one in "The Optimality of the On-Line Greedy Algorithm in Carpool and Chairman Assignment Problems" by Don Coppersmith, Tomasz J. Nowicki, Giuseppe A. Paleologo, Charles Tresser, and Chai Wah Wu (*ACM Transactions on Algorithms* **7** no. 3 [July 2011], Article no. 37). This paper explains how greedy algorithms can be used in various applications similar to car pools, and it shows how greedy algorithms are optimal in these situations.

Brian Lins is an associate professor of mathematics at Hampden-Sydney College. He lives near Richmond, Virginia, with his family.
Email: blins@hsc.edu

http://dx.doi.org/10.4169/mathhorizons.23.2.16