IBM Journal of research and development

Volume 27, Number 2, March 1983

Ronald Fagin
John H. Williams

A Fair Carpool Scheduling Algorithm

Ronald Fagin
John H. Williams

A Fair Carpool Scheduling Algorithm

We present a simple carpool scheduling algorithm in which no penalty is assessed to a carpool member who does not ride on any given day. The algorithm is shown to be fair, in a certain reasonable sense. The amount of bookkeeping grows only linearly with the number of carpool members.

1. Introduction

Suppose that N people, tired of spending their time and money in gasoline lines, decide to form a carpool. We present a scheduling algorithm for determining which person should drive on any given day. We want a scheduling algorithm that will be perceived as fair by all the members so as to encourage their continued participation. We begin by presenting three algorithms (Scheduling Algorithms 1–3 below) and discussing their flaws. We then present the algorithm (Scheduling Algorithm 4) that we propose. We assume for now that on any given day at most one car is the "carpool car." This assumption is relaxed later.

Scheduling Algorithm 1 (simple rotation) The simplest scheme, and the one most often used, is simply to rotate driving, e.g., in alphabetical order. Thus, if there are N members of the carpool, then person i is responsible for driving on the ith day and every N driving days thereafter. This scheme has the obvious advantage that it is simple to describe and it is easy to determine who drives next. The difficulty with this scheme arises when one or more people do not participate in the carpool on a particular day. If the designated driver has to stay out on the day that he is supposed to drive, then he will have to swap days with someone else. After a few such occurrences, it may become difficult to determine who is to drive the next day. If a non-driver misses one or more days, should he be expected to drive in his normal rotation? If so, he may soon perceive the carpool to be more of a burden than a blessing and drop out altogether. Just as big a problem as the person who cannot drive on his scheduled day is the person who must (for personal reasons) drive on someone else's day but could otherwise participate in the carpool (for example, a person who is going to work as usual but needs to have his car in order to go to the bank to deposit the money he has saved by carpooling). We want a scheduling algorithm that will always be tolerant of exceptional conditions and that will never discourage participation. In particular, we want an algorithm that is *robust*, in the following sense: A person can drive on a day that the algorithm says someone else should drive, and it is then easy to see how to get "back in synch" later.

Scheduling Algorithm 2 (simple tokens) In order to correct the deficiencies of simple rotation, we might adopt the following procedure. Each time a person R rides with a driver $D \neq R$, then R pays D one "ride token." Of course, the tokens would not actually need to be handled; each person's current token holding could simply be recorded somewhere, and that record could be updated daily. Then the algorithm for determining who drives next would be to choose, from among the people participating that day, the person with the smallest holding of tokens.

When we formally define fairness, in Section 3, we shall see that this scheduling algorithm is not fair in our sense. In the worst case, some carpool member may be forced to drive far more than his "fair share," as we shall see. We now briefly mention a few intuitive reasons why this algorithm is

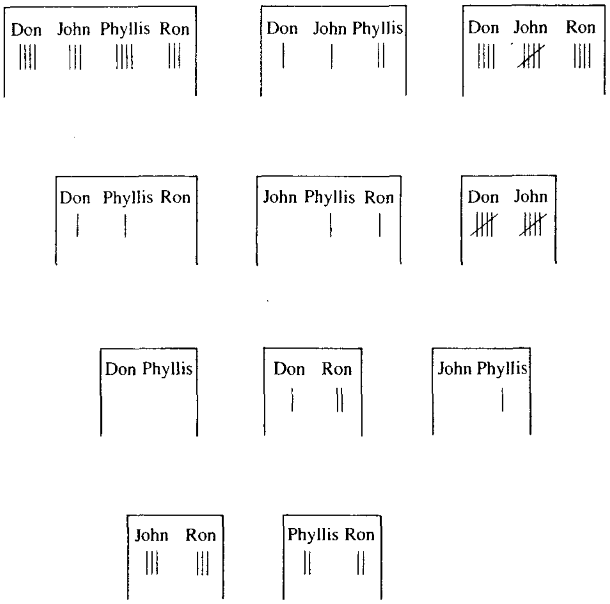

Figure 1 Books for Scheduling Algorithm 3.

Date	Don	John	Phyllis	Ron
	0	0	0	0
May 1	0	8	−4	−4
May 2	−3	5	−7	5
May 3	−9	5	−1	5

Figure 2 Books for Scheduling Algorithm 4.

not fair. (1) It is certainly quite advantageous to drive on days when many people are participating (since the driver gets one ride token from each of the other participants). If a carpool member were unlucky enough to be the designated driver on several "bad" (sparsely attended) days, then he might decide that the algorithm is not fair, and might even be driven to drop out of the carpool. (2) On a "good" day (a day in which there are many participants), if two carpool participants A and B were tied for the lowest score, then both A and B would want very much to drive, and some tie-breaking scheme would have to be devised. (3) Finally, this algorithm is not robust in the sense we have defined: If A were a carpool member, if it were not A's turn to drive according to the algorithm (that is, A did not have the lowest score among the participants on that day), and if A insisted on driving his car on that day for personal reasons, then the other carpool members would be quite unhappy if this were a "good" day.

Scheduling Algorithm 3 (subsets) The next scheduling algorithm to be described does turn out to be fair in our sense; the problem, as we shall see, is the amount of bookkeeping required. This algorithm records, for each of the $2^N - (N + 1)$ nontrivial subsets of carpool members (subsets of two or more), the number of times that each member of the subset has driven that particular group of people. For example, if there are four people named Don, John, Phyllis, and Ron in the carpool, then the books at a given point might look like Fig. 1 (where, for example, a tally is entered under Phyllis in the Don-Phyllis-Ron table on a day in which only Don, Phyllis, and Ron participate in the carpool and Phyllis drives). If the table is as in Fig. 1, then on the next day in which the only participants are Don, Phyllis, and Ron, the driver should be the person (in this case, Ron) with the least number of tallies in the Don-Phyllis-Ron table. With this method, it is clear that a person is not penalized for nonparticipation on any day. It is intuitively clear that this algorithm is fair, since it is essentially simple rotation applied separately to each of the $2^N - (N + 1)$ nontrivial subsets. Further, it is clear that this algorithm is robust in our sense. Unfortunately, the bookkeeping for this algorithm becomes a nightmare (if the number N of people is, say, four or more) because the size of the book grows exponentially with the size of the carpool. Further, this scheduling algorithm neglects certain trade-offs. For example, Phyllis and John appear together in four of the tables in Fig. 1, but Scheduling Algorithm 3 makes no attempt to trade off rides in the tables in which Phyllis and John appear together. In fact, in Fig. 1, Phyllis has driven more times than John in each of the four tables in which they both appear.

2. The proposed scheduling algorithm

We now give our proposed scheduling algorithm.

Scheduling Algorithm 4 (fair carpool scheduling algorithm) We begin by defining U to be a value that, intuitively, represents the total cost of a trip. It is convenient to take U to be the least common multiple of $1, 2, \cdots, m$, where m is the largest number of people who ever ride together at a time in the carpool. In the running example we shall give, we assume that this number m is taken equal to the total number N of members of the carpool, which in turn is assumed to be 4. Thus, U is taken to be the least common multiple of 1, 2, 3, and 4; that is, U is 12. As drawn in Fig. 2, the books consist of a single table, with one column for the date and one column for each carpool participant. Each day that the carpool drives, a new row is entered into the table. The table is initialized with a row of all 0's (the first row of the table in Fig. 2). If, on a given day, there are k participants in the carpool and A is the driver, then the A entry is increased by $U(k - 1)/k$ units (that is, the entry for that day in the A column is $U(k - 1)/k$ more than the A entry in the previous row), and the entries of the riders who do not drive are each decreased by U/k. For example, in Fig. 2, the first day of the carpool was May 1, and John was the driver. On that day, Phyllis and Ron rode in John's car. Thus, John gained 8

$$s_k(1) + \cdots + s_k(i) > s_{k-1}(1) + \cdots + s_{k-1}(i). \tag{7}$$

We now show that (7) implies that

$$s_{k-1}(i + 1) + U > s_k(i). \tag{8}$$

Now (7) says that the sum of the i biggest scores strictly increases between rows $k - 1$ and k. How can this happen? Let A be the driver of the carpool at time k. Thus, A has the lowest score in row $k - 1$ among those who participate in the carpool on day k. It is not hard to see that for the sum of the i biggest scores to strictly increase between rows $k - 1$ and k, it is necessary that

1. A's score in row $k - 1$ is $s_{k-1}(j)$ for some $j > i$; that is, A's score is one of the lowest $N - i$ scores in row $k - 1$, and
2. A's score in row k is $s_k(m)$ for some $m \leq i$; that is, A's score is one of the biggest i scores in row k.

Now the driver's score increases by less than U when he drives. Therefore, A's score just before he drove [that is, $s_{k-1}(j)$] differs from his score just after he drove [that is, $s_k(m)$] by less than U. Hence,

$$s_{k-1}(j) + U > s_k(m). \tag{9}$$

Now $s_{k-1}(i + 1) \geq s_{k-1}(j)$, since $j > i$, and so (by adding U to both sides), we get

$$s_{k-1}(i + 1) + U \geq s_{k-1}(j) + U. \tag{10}$$

Further,

$$s_k(m) \geq s_k(i), \tag{11}$$

since $m \leq i$. Clearly, (8) follows immediately from (9), (10), and (11). Now (5) and (8) together imply that

$$s_{k-1}(i + 1) > M - (ia_i + 1)U,$$

that is,

$$s_{k-1}(i + 1) > M - a_{i+1}U. \tag{12}$$

Define t_{i+1} to be $k - 1$. Then (12) tells us that (2) holds. Further, $t_{i+1} < t_i$, since we already showed that $k \leq t_i$. This completes the induction. Hence, (1) holds for each i $(1 \leq i \leq N)$. Let $t = t_N$. We see from (1), when $i = N$, that $s_t(N) > M - a_N U$. But $M = a_N U$, and so

$$s_t(N) > 0. \tag{13}$$

Since $s_t(i) \geq s_t(N)$ for $1 \leq i \leq N$, it follows from (13) that $s_t(i) > 0$ for each i $(1 \leq i \leq N)$. Thus, every entry of row t is strictly positive, and so the checksum of row t is strictly positive. But this contradicts Proposition 1, which says that the checksum of every row is 0. This contradiction completes the proof. □

- *Corollary 3*

Let N, the number of members of the carpool, be fixed. Then there is a number M' such that for each schedule of arrivals, the table derived by applying Scheduling Algorithm 4 contains no entry whose absolute value is larger than M'.

Proof Let M be as in Theorem 2, and let T be a table derived by applying Scheduling Algorithm 4 to some schedule of arrivals. By Theorem 2, we know that no positive entry in the table can be larger than M. How large in absolute value can the smallest entry (the negative entry with the biggest absolute value) in the table be? Let r be a row of the table. Now no entry of the table can be larger than M, and there can be at most $N - 1$ positive entries in row r (because, by Proposition 1, the checksum of row r is 0). Hence, the sum of the positive entries in row r is at most $(N - 1)M$. Since the checksum of row r is 0, the absolute value of the sum of the negative entries in row r is equal to the sum of the positive entries in row r, and so is also at most $(N - 1)M$. Therefore, the absolute value of the smallest ("most negative") member of row r is at most $(N - 1)M$. Thus, we can take M' to be $(N - 1)M$. □

It follows from our proof of Theorem 2 that an upper bound M on the size of the biggest entry that can ever appear in the table is $a_N U$, where N is the number of carpool members and where $a_1 = 0$ and $a_{i+1} = 1 + ia_i$ $(1 \leq i \leq N)$. This bound is not the best possible. For example, if $N = 2$, then our upper bound is U, whereas it is very easy to see that in this case the actual upper bound is only $U/2$. If $N = 3$, then our upper bound is $2U$, whereas a careful examination of the possibilities shows that the actual upper bound is $(5/6)U$. Let us define the function f by letting $f(N)U$ be the actual upper bound if there are N carpool members. Thus, $f(2) = 1/2$ and $f(3) = 5/6$. We note that $f(4) = 7/6$ and $f(5) = 8/5$. We have not found $f(N)$ exactly for $N \geq 6$.

- *Proposition 4*

The function f is monotone and unbounded.

Note By *monotone*, we mean that if $N_1 \leq N_2$, then $f(N_1) \leq f(N_2)$. By *unbounded*, we mean $f(N)$ gets arbitrarily large as N gets large.

Proof Any score that can be obtained in a carpool with N_1 members can be obtained in a carpool with $N_2 \geq N_1$ members: we can simply assume that $N_2 - N_1$ members of the larger carpool never participate. Monotonicity follows immediately.

We now show unboundedness. Let $N = 2^r$, and assume that the carpool members are $A_1, \cdots, A_N$. Assume that on the first day, the participants are A_1 and A_2, and the driver is A_1; on the second day, the participants are A_3 and A_4, and the driver is A_3; and so on for a total of $N/2$ days. Then there is a second round that begins on the $((N/2) + 1)$th day. On the first day of the second round, the participants are A_1 and A_3, and the driver is A_1; on the next day, the participants are A_5

☐

• • •

☐

• • •