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

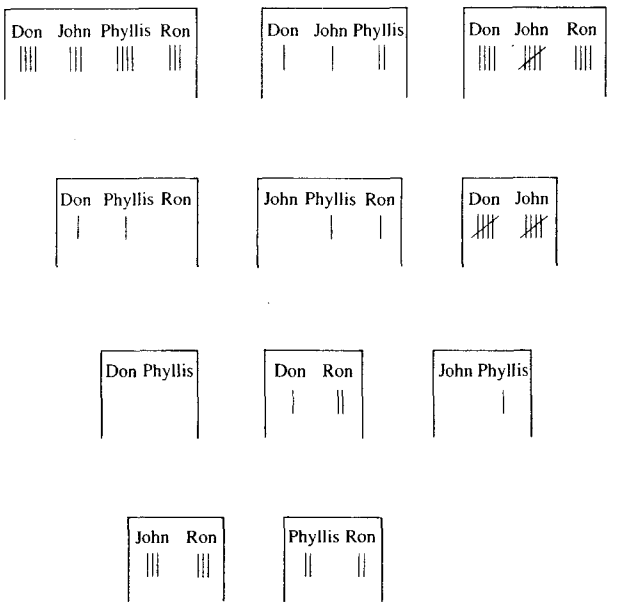

Figure 1 Books for Scheduling Algorithm 3.

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

units, and Phyllis and Ron each lost 4 units. On the next day, May 2, all four carpool members participated, and Ron was the driver. (The algorithm says that either Phyllis or Ron should be the driver on May 2, since they are tied for the lowest score, with -4 units each.) Since Ron drove, he gained 9 units, and each of the others lost 3 units. On the next day, May 3, only Don and Phyllis participated. Since Phyllis had a lower score than Don (-7 versus -3), she was the driver. She gained 6 units and Don lost 6 units. Note that by choosing U as we have (in this case, $U = 12$), every entry of the table is an integer.

An intuitive way to view this scheduling algorithm is that the "cost" of driving is taken to be U units, and this cost is divided equally among each of the participants. So, if there are k participants, then the cost to each participant is U/k. Thus, each of the participants who is not the driver "pays" U/k units to the driver.

We now show that for each row, the *checksum* (the sum of the entries) is zero. For example, on May 2, the entries are -3, 5, -7, and 5, which add to 0. This property provides a redundancy check on the arithmetic.

• *Proposition 1*
In each table generated by Scheduling Algorithm 4, the checksum of each row is zero.

Proof When k people participate, one of them (the driver) gains $U(k-1)/k$ units, the other $k-1$ participants each lose U/k units, and the values of the nonparticipants are unchanged. Thus, the *net* gain or loss is 0, and since the table is initialized to all 0's, the checksum is always 0. □

We now show that the entries in the table are bounded for each N (where N is the number of members of the carpool). We shall make use of this result later, in our proof of fairness.

The *schedule of arrivals* is a finite sequence $\langle S_1, S_2, \cdots, S_n\rangle$, where S_i is the set of participants in the carpool on day i (or as we may also say, *at time i*). Intuitively, the schedule of arrivals tells who participated in the carpool, day by day. For example, the schedule of arrivals $\langle ABC, BD, ACD\rangle$, where ABC is an abbreviation for $\{A, B, C\}$, etc., corresponds to persons A, B, and C participating in the carpool (riding in the carpool car) on the first day, persons B and D participating on the second day, and so on.

• *Theorem 2*
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 larger than M.

Proof Assume that the theorem is false; we shall derive a contradiction. Find N such that, for each M, there is a table T (which can be derived by applying Scheduling Algorithm 4 to some schedule of arrivals) with an entry larger than M.

Define the sequence $a_1, \cdots, a_N$ recursively by letting $a_1 = 0$, and $a_{i+1} = 1 + ia_i$, for $1 \le i < N$. Let M be $a_N U$. Let T be a table (that is derived by applying Scheduling Algorithm 4) with an entry larger than M. Let us call the top row (with all zeros as entries) of table T *row 0*, the next row of the table *row 1*, and so on. If the N entries of row t are $b_1 \ge b_2 \ge \cdots \ge b_N$, then define $s_t(i)$ to be b_i. Thus (with ties properly accounted for), $s_t(i)$ is the ith largest entry of row t. We think of row t as containing the scores of members of the carpool just after the carpool has driven on time t (that is, the scores after time t but before time $t+1$).

Since table T contains an entry larger than M, we know that $s_t(j) > M$, for some t and j. Hence, $s_t(1) > M$, since $s_t(1) \ge s_t(j)$. Let t_1 be the least t such that $s_t(1) > M$. We now show that there are $t_2, \cdots, t_N$, where $t_1 > t_2 > \cdots > t_N$, such that for each i ($1 \le i \le N$),

$$s_{t_i}(i) > M - a_i U. \tag{1}$$

We already know that (1) holds when $i = 1$, since $a_1 = 0$. Assume inductively that we have found $t_1 > t_2 > \cdots > t_i$ such that $s_{t_p} > M - a_p U$ for $1 \le p \le i$; in particular (when $p = i$) we see that (1) holds. We must find $t_{i+1} < t_i$ such that

$$s_{t_{i+1}}(i+1) > M - a_{i+1} U. \tag{2}$$

Now $s_t(j) \ge s_t(i)$ when $1 \le j \le i$. Hence,

$$s_t(1) + \cdots + s_t(i) \ge i s_t(i). \tag{3}$$

By (1) and (3), it follows that when $t = t_i$, we have

$$s_t(1) + \cdots + s_t(i) > iM - ia_i U. \tag{4}$$

Let k be the least value of t such that (4) holds. Note for later use that $k > 0$, since $s_0(j) = 0$ for each j. We now show that

$$s_k(i) > M - ia_i U. \tag{5}$$

If $i = 1$, then $k = t_1$, by definition of t_1 (since $a_1 = 0$). So, if $i = 1$, then (5) holds. We now show that (5) holds if $i > 1$. We know that $k \le t_i$, since, as we showed, (4) holds when $t = t_i$. Since $k \le t_i < t_1$, it follows by minimality of t_1 that $s_k(j) \le M$, for $1 \le j \le N$. In particular,

$$s_k(j) \le M, \text{ for } 1 \le j \le i-1. \tag{6}$$

By (4), with $t = k$, and by (6), it follows that (5) holds, which was to be shown.

We know that k is the least value of t such that (4) holds, and that, as noted, $k > 0$. Therefore,

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

We now show that (7) implies that

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

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,

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

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

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

Further,

$$s_k(m) \geq s_k(i), \quad (11)$$

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,

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

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

$$s_t(N) > 0. \quad (13)$$

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

and A_7, and the driver is A_5; and so on. Then there is a third round; on the first day of the third round, the participants are A_1 and A_5, and the driver is A_1; and so on. This continues for a total of $\lfloor \log_2 N \rfloor$ rounds, where $\lfloor x \rfloor$ is the greatest integer not exceeding x. It is straightforward to see that after the final round, A_1's score is $r/2$ (where $N = 2^r$). Thus, $f(2^r) \geq r/2$. Hence, f is unbounded.

We close the proof by noting another way of showing unboundedness. As before, let $A_1, \cdots, A_N$ be the carpool members. Assume that on the first day, everyone participates, and the driver is A_N. Assume that on the second day, the participants are $A_1, \cdots, A_{N-1}$, and the driver is A_{N-1}, and so on. Thus, on the ith day ($1 \leq i \leq N-1$), the participants are $A_1, \cdots, A_{N-i+1}$, and the driver is A_{N-i+1}. It is clear that A_1's score after $N-1$ days is $-U/N - U/(N-1) - U/(N-2) - \cdots - U/2$, which gets arbitrarily large in absolute value as N increases (and which, in particular, is asymptotic to $-U \log N$). □

It follows from the proof of Proposition 4 that $f(N) \geq (1/2) \lfloor \log_2 N \rfloor$. D. Coppersmith (private communication) has improved this logarithmic lower bound to a linear lower bound by using the following argument. Let N be the number of members of the carpool. As in the proof of Theorem 2, let the scores just after time t be $s_t(1) \geq s_t(2) \geq \cdots \geq s_t(N)$. Define the "figure of merit" just after time t to be $(N-1)s_t(1) + (N-2)s_t(2) + \cdots + (0)s_t(N)$. We now define the schedule of arrivals. On each day, the set of participants consists of two members with the same score. If there are no two members with the same score, then the carpool stops running. If i and j ride together, and i is the driver, then i's score increases by $U/2$ and j's score decreases by $U/2$. The net effect on the figure of merit of increasing one value $s_t(i)$ by $U/2$ and decreasing $s_t(i+1)$ by $U/2$ is to increase the figure of merit by $U/2$. Further, it is easy to see that the net effect of reshuffling the scores to keep the $s_t(i)$'s nondecreasing can only increase the figure of merit further. Keep the carpool running until either no two participants have the same score, or until the figure of merit has gone beyond N^3U, whichever comes first. In the first case (where the carpool is run until no two participants have the same score), we know that since all carpool members started with a score of 0, and since scores change by $U/2$ at a time, the scores will be at least $U/2$ apart. That is, no two scores will be closer together in value than $U/2$. It is not hard to verify that this fact, along with the fact that the sum of the scores is 0, implies that the largest score is at least $(N-1)U/4$. In the second case (where the carpool is run until the figure of merit has gone beyond N^3U), it is clear that the largest score is greater than NU. So in either case, the largest score is at least $(N-1)U/4$, which is linear in N, as promised. Note that the linear lower bound is attained even when no more than two carpool members ever ride together. Coppersmith also shows (by a more detailed analysis) a lower bound of $(N-1)U/3$, which is attained even with no more than three carpool members ever riding together.

Coppersmith's argument, taken together with the proof of Theorem 2, shows that $(N-1)/3 \leq f(N) \leq a_N$, where $a_1 = 0$ and $a_{i+1} = 1 + ia_i$ ($1 \leq i \leq N$). There is an exponential gap between these lower and upper bounds. It is an interesting combinatorial problem to tighten these bounds [1].

We close this section by noting another interesting combinatorial problem. Let us say that a vector $(a_1, \cdots, a_N)$, where $a_1 \geq \cdots \geq a_N$, is *an attainable vector of scores* if there is a schedule of arrivals such that, starting with a score of 0 for every member of the carpool, and always applying Scheduling Algorithm 4, there is a time t where the vector $(s_t(1), \cdots, s_t(N))$ of scores is equal to $(a_1, \cdots, a_N)$. We conjecture that if $(a_1, \cdots, a_N)$ is an attainable vector of scores, then so is the negation $(-a_N, \cdots, -a_1)$. If the conjecture is true, then the M' of Corollary 3 and the M of Theorem 2 can, of course, be taken to be the same.

3. Fairness

In this section, we discuss a concept of fairness and show that our scheduling algorithm (Scheduling Algorithm 4) is fair. However, we shall see that Scheduling Algorithm 2 (simple tokens) is not fair. We shall also see that Scheduling Algorithm 1 (simple rotation) is fair (when it can be applied), and that Scheduling Algorithm 3 (subsets) is fair (but it requires too much bookkeeping).

To help us understand fairness, let us first consider Scheduling Algorithm 3 (subsets). Scheduling Algorithm 3 is fair in the sense that among the times that person A rides precisely with, say, B and C, the driver is person A approximately $1/3$ of the time (with the obvious generalization that A is the driver approximately $1/k$ of the time that he rides with a fixed subset of $k-1$ others.) Less restrictively, we might consider a scheduling algorithm fair if each person is the driver approximately $1/k$ of the time that he rides with $k-1$ others (not necessarily a fixed subset of $k-1$ others.) Thus, if the carpool consists precisely of A, B, C, and D, then A might be expected to drive approximately $1/3$ of the time that he rides with precisely two among B, C, and D. In other words, let c_X be the number of times (through time t) that X is precisely the set of those participating in the carpool on that day. Then during those days that A rides with precisely two among B, C, and D, the number of times that we might want A to drive is approximately

$$\frac{1}{3}(c_{ABC} + c_{ABD} + c_{ACD}).$$

Even less restrictively, assume that through time t, person A has participated in the carpool on b_2 days when exactly 2

persons participated in the carpool, on b_3 days when exactly 3 persons participated in the carpool, and so on. Let us define A's *ideal number of drives* to be the number

$$\frac{1}{2}b_2 + \frac{1}{3}b_3 + \cdots + \frac{1}{N}b_N. \tag{14}$$

Our notion of fairness is that A should be the driver of the carpool car approximately this number of times.

We are now ready to give our formal definition of fairness. We say that a carpool scheduling algorithm is *fair* if for each N (where N is the number of members of the carpool), there is a number P such that whatever the schedule of arrivals, it is the case that at each time t and for each carpool member A, the number of times that A has actually driven differs from his ideal number of drives in absolute value by no more than P.

We shall show that our scheduling algorithm is fair. We first prove a simple proposition.

• *Proposition 5*
Let x be the number of times that A has actually driven through time t, and let y be A's ideal number of drives through time t. Then the number $(x - y)U$ is A's entry in row t of the table in Scheduling Algorithm 4.

Proof We prove the proposition by induction on t. It is obviously true for $t = 0$, since every entry in row 0 is 0, and in this case $x = y = 0$. Assume inductively that the statement of the proposition is true for $t = m$; we shall show that it holds for $t = m + 1$. Let x_t be the number of times that A has actually driven through time t, let y_t be A's ideal number of drives through time t, and let A_t be A's entry in row t of the table. By inductive assumption, the number $(x_m - y_m)U$ equals A_m (that is, A's entry in row m of the table). We must show that the number $(x_{m+1} - y_{m+1})U$ equals A_{m+1} (that is, A's entry in row $m + 1$ of the table). Assume that there are k participants in the carpool at (on the day corresponding to) time $t + 1$. There are two cases, depending on whether A is the driver at time $m + 1$.

Case 1 A is the driver at time $m + 1$. Then $x_{m+1} = x_m + 1$, and $y_{m+1} = y_m + (1/k)$. Hence,

$$(x_{m+1} - y_{m+1})U = (x_m - y_m)U + U(k - 1)/k. \tag{15}$$

But by assumption,

$$(x_m - y_m)U = A_m. \tag{16}$$

Since A is the driver at time $m + 1$, it follows from Scheduling Algorithm 4 that

$$A_{m+1} = A_m + U(k - 1)/k. \tag{17}$$

It follows from (15), (16), and (17) that $(x_{m+1} - y_{m+1})U = A_{m+1}$, which was to be shown.

Case 2 A is not the driver at time $m + 1$. Then $x_{m+1} = x_m$, and $y_{m+1} = y_m + (1/k)$. Hence, $(x_{m+1} - y_{m+1})U = (x_m - y_m)U - U/k$. As in Case 1, it follows easily that $(x_{m+1} - y_{m+1})U$ is A's entry in row $m + 1$. □

The next theorem discusses the fairness or unfairness of the scheduling algorithms we have discussed. We are most interested in the result that Scheduling Algorithm 4 is fair.

• *Theorem 6*
Scheduling Algorithm 1 (when it applies), Scheduling Algorithm 3, and Scheduling Algorithm 4 are fair, but Scheduling Algorithm 2 is not fair.

Proof Recall that a carpool scheduling algorithm is *fair* if for each N (where N is the number of members of the carpool), there is a number P such that whatever the schedule of arrivals, it is the case that at each time t and for each carpool member A, the number of times that A has actually driven differs from his ideal number of drives in absolute value by no more than P.

Scheduling Algorithm 1 (simple rotation) is fair (when it applies) Of course, Scheduling Algorithm 1 is very limited, since it is not even defined unless every carpool member participates in the carpool on every day. If so, then it is easy to see that the desired number P above can be taken to be 1.

Scheduling Algorithm 2 (simple tokens) is not fair Assume that there are 6 carpool members A, B, C, A', B', and C', and that the schedule of arrivals is $\langle AA', ABC, AB, AC, A'B'C', A'B', A'C', ABC, AB, AC, A'B'C', A'B', A'C', \cdots, ABC, AB, AC, A'B'C', A'B', A'C' \rangle$, where the sequence $ABC, AB, AC, A'B'C', A'B', A'C'$ repeats over and over a total of m times after the initial AA' (and so the number of days is $6m + 1$.) We shall show that there is no number P as defined above that works for every m. On the first day, when AA' is the set of carpool participants, either A or A' is the driver. Assume without loss of generality that A' is the driver; otherwise, everything we now say holds when we replace A, B, C by (respectively) A', B', C'. We leave to the reader the simple verification that under Scheduling Algorithm 2 it follows that on each of the m days the set of carpool participants is precisely ABC (respectively, AB or AC), the driver is always A (respectively, B or C.) Now there are exactly $2m + 1$ days that A participates in the carpool when precisely 2 people participate (namely, A and one of A', B, or C), and there are exactly m days that A participates in the carpool when precisely 3 people participate (namely A, B, and C). Thus, A's ideal number of drives is $(1/2)(2m + 1) + (1/3)m$, which equals $(4/3)m + (1/2)$. The number of times that A actually drives is $m + 1$. The difference between ideal and actual is $(m/3) - (1/2)$, which is not bounded by any fixed number P (as m gets large.) This was to be shown.

Scheduling Algorithm 3 (subsets) is fair It is easy to see that P can be taken to be equal to the number of subsets that contain a given member A, that is, P can be taken to be 2^{N-1}, where N is the number of carpool members.

Scheduling Algorithm 4 is fair As in the statement of Proposition 5, let x be the number of times that A has actually driven through time t, and let y be A's ideal number of drives through time t. By Proposition 5, the number $(x - y)U$ is A's entry in row t of the table. By Corollary 3, we can find a positive number M' (which depends only on N, the number of members of the carpool) such that no entry in the table is larger in absolute value than M'. Thus, $|(x - y)\ U| \leq M'$. Hence, $|(x - y)| \leq M'/U$. So, we can take P to be M'/U. □

4. Further observations

As well as being both fair and manageable, our carpool scheduling algorithm (Scheduling Algorithm 4) has some additional attractive features. First, although it will always determine whose turn it is to drive on a particular day, it is robust in the presence of deliberate imbalance. Thus a person could drive (or not drive) for several days in a row if he needed to, regardless of whether the scheduling algorithm says he should or should not drive, and the imbalance would eventually be eliminated. (It is beyond the scope of this paper to make this last sentence precise. One meaning is that after the driving table is artificially made imbalanced, the entries will remain bounded from then on, as in Theorem 2, provided the scheduling algorithm is faithfully adhered to from then on.)

Second, the "ride units" of the method can become a commodity that can be bought and sold. This can also allow for "carpool members" who never drive at all. Thus, if A does not have a car but wishes to participate in the carpool, if B is a carpool participant (with a car), and if A and B can agree on a fair market value for a ride unit, then B can sell ride units to A, and A need never drive. (In effect, B is "driving for" A.) In fact, the group for which this scheme was developed had such a participant. His name being Don and twelve being the least common multiple of the possible subset sizes of the carpool, the "ride unit" became affectionately known as the *Duodecadon.*

Finally, although we derived this scheduling algorithm on the assumption that there would be only one official carpool car on any one day, that assumption turned out to be superfluous! In fact, there can be as many carpool cars as there are people driving. Each driver of a car containing k participants gets credited with $U(k - 1)/k$ units, and each rider who does not drive in such a car gets debited U/k units in some master (perhaps company-wide!) record. Note that a person driving alone gets 0 units, or no change to the record. In this generalized scheduling algorithm, an arbitrary group of people, who had never previously carpooled with one another, could decide to ride to work together, and it would make perfect sense for them to ask, "Whose turn is it to drive today?"

Acknowledgments

The first author would like to acknowledge his former Berkeley roommates, Larry Carter and John Gill, who successfully implemented with him Scheduling Algorithm 3 (to determine whose turn it was to cook). Further, the authors thank Phyllis Reisner, Don Stanat, and Jim Sutton, who at various times participated with the authors in the carpool in which Scheduling Algorithm 4 was developed. We are grateful to Carl Hauser for calculating $f(5)$ and to Don Coppersmith for proving the linear lower bound on f in Section 2.

Note

1. After this paper went to press, Coppersmith lowered the upper bound to $(N - 1)/2$. Thus we now know that $(N - 1)/3 \leq f(N) \leq (N - 1)/2$.

Received July 6, 1982; revised September 13, 1982

Ronald Fagin *IBM Research Division, 5600 Cottle Road, San Jose, California 95193.* Dr. Fagin is the manager of the foundations of computer science group in the Computer Science Department in San Jose. He joined IBM in 1973 at the Thomas J. Watson Research Center, Yorktown Heights, New York. While there, he did research on storage management analysis. In 1975, he transferred to San Jose, where most of his research has centered on the theory of relational data bases. He has received two IBM Outstanding Innovation Awards. The first, received in 1981, was for fundamental contributions to relational data base theory. The second, also received in 1981, was for his joint research on extendible hashing, a fast access method for dynamic files. He received his B.A. in mathematics from Dartmouth College, Hanover, New Hampshire, in 1967, and his Ph.D. in mathematics from the University of California at Berkeley in 1973. Dr. Fagin is a member of the Association for Computing Machinery and its special interest groups on the Management of Data and on Automata and Computability Theory.

John Hayden Williams *IBM Research Division, 5600 Cottle Road, San Jose, California 95193.* Dr. Williams is a Research staff member in San Jose, where he is working with IBM Fellow John Backus on the development of Functional Programming Languages, an alternative to conventional programming languages. He joined IBM in 1978; prior to that, he was an Associate Professor of Computer Science at Cornell University, Ithaca, New York. Dr. Williams received his B.S. and M.S. in mathematics and his Ph.D. in computer science in 1969 from the University of Wisconsin at Madison.