

# Algorithms for the Carpool Problem

João Pedro Boavida <sup>\*†</sup>    Vikram Kamat <sup>‡</sup>    Darshana Nakum <sup>§</sup>    Ryan Nong <sup>¶</sup>  
Chai Wah Wu (mentor) <sup>||</sup>    Xinyi Zhang <sup>\*\*</sup>

August 18, 2006

## Abstract

We investigate the carpool problem, on choosing a driver from a subset of a set of people who are members of a carpool. Coppersmith et al. [4] examine the bounds on the values that characterize the fairness of carpool algorithms. In this report, we study the methods introduced in that paper and suggest computational techniques for further improving the lower bound. Specifically, we introduce an algorithm to search the tree of evolution of a given error vector. With this algorithm, we eliminate more possibilities of error vectors from consideration for a higher lower bound.

## 1 Introduction and Problem Formulation

The carpool problem can be defined as follows: A certain set of  $n$  people decide to carpool to work. Each day a certain subset of these people arrive. They need to formulate an algorithm to decide on who drives for the day.

The important thing to consider here is the choice of driver which should be as fair as possible. This means that over a long period of time, it should not be that a single driver (or a certain subset of drivers) has driven considerably more times than the rest. The algorithm should also be reasonably robust, meaning that on any given day if a person who is not supposed to drive drives or vice versa, the algorithm should be able to recover from such exceptional cases and still be able to make fair choices at a later stage.

The carpool problem was first proposed and studied by Fagin and Williams [1] and later studied by Ajtai et al. [2]. The papers, for the most part, deal with online schemes, i.e., when the schedule

---

\*School of Mathematics, University of Minnesota, Twin Cities.

†Departamento de Matemática, Instituto Superior Técnico, Lisboa, Portugal.

‡Department of Mathematics and Statistics, Arizona State University.

§Department of Mathematical Sciences, University of Nevada.

¶Computational and Applied Mathematics Department, Rice University.

||TJ Watson Research Center, IBM.

\*\*Department of Mathematics, University of Delaware.

of future arrivals of participants is not known. Fagin and Williams [1] discuss a few elementary schemes for choosing a driver each day, highlights their flaws in terms of fairness and/or robustness and then propose a scheduling algorithm which is fair (in the sense considered in the paper which also is the one we discuss now).

To define the fairness of an algorithm, we consider the ideal number of drives a participant makes. Ideally, if person  $i$  shows up  $b_k$  of the days when a total of  $k$  participants show up, then  $i$  should drive  $b_k/k$  of those days. That is, the ideal number of times that person  $i$  drives is  $\sum_k b_k/k$ . (Naor [3] gives an axiomatic characterization of the concept of “fair share”, as described in [1].) We say that an algorithm is *fair* if the highest (across all participants) deviation between the ideal and actual number of drives is bounded over time, independently of the schedule of arrivals.

We introduce the vector  $x = (x_i)$  of these errors.

If  $s$  is the set of arrivals on a given day, and  $i$  is the driver, the new error vector will be  $x' = x + \delta(s, i)$ , where  $\delta(s, i) = -e_i + \frac{1}{|s|} \sum_{j \in s} e_j$  and  $|s|$  is the number of arrivals.

In these terms, an algorithm is *fair* if there is a bound  $B$  such that  $\max_{i,t} |x_i(t)| \leq B$  independently of the schedule of arrivals.

Finding an algorithm requiring a lowest bound  $B$  has at least two benefits. Firstly, when we actually want to model fairness, the lowest values of  $B$  will guarantee the smallest deviations from the “ideal shares”. Secondly, for automated schedule generation we certainly have an interest in reducing the range of possible values of whatever variables we need to keep track of.

Clearly,  $x \in \frac{1}{L} \mathbb{Z}^n$ , where  $L = \text{lcm}(1, 2, \dots, n)$ . Moreover,  $\sum_i x_i = 0$ , so the possible error vectors lie on a hyperplane. Figure 1 shows what happens with  $n = 3$  and a bound  $B = 5$ .

One way to assure fairness is to, if the current state is described by the error vector  $x$ , choose the driver that makes the new error vector  $x'$  closest to the origin. In other words, the driver will be whomever has the highest score  $x_i$  (i.e., whomever has gotten the more free rides so far). This is known as the greedy algorithm.

Coppersmith et al. [4] discuss the generalized carpool problem and prove that the greedy algorithm (for the generalized problem) is the optimal strategy, in the sense that its bound  $B$  is the lowest. It is also known that the bound for the greedy algorithm satisfies  $B < \frac{n-1}{2}$ , and that  $\frac{n-1}{2}$  is the lowest bound possible for  $B$  of *any* algorithm. For the carpool problem, Coppersmith et al. [4] show that *any* algorithm satisfies  $B \geq \frac{3}{8}(n-1) - \frac{1}{4}$ . The goal of our project is to close the gap between these two bounds.

## 2 Algorithm

One way to find a bound  $B$  is to look at error vectors  $x$  with some special property (which make them easy to determine). Following Coppersmith et al. [4], we consider points with maximum second moment  $\|x\|^2 = \sum_i x_i^2$ .

Another look at Figure 1 will explain why this is a reasonable approach: If the algorithm is fair, then only finitely many error vectors can be reached, and certainly some of them will have maximum moment. These error vectors will be those closest to the boundary, and so they will give us a good lower bound.

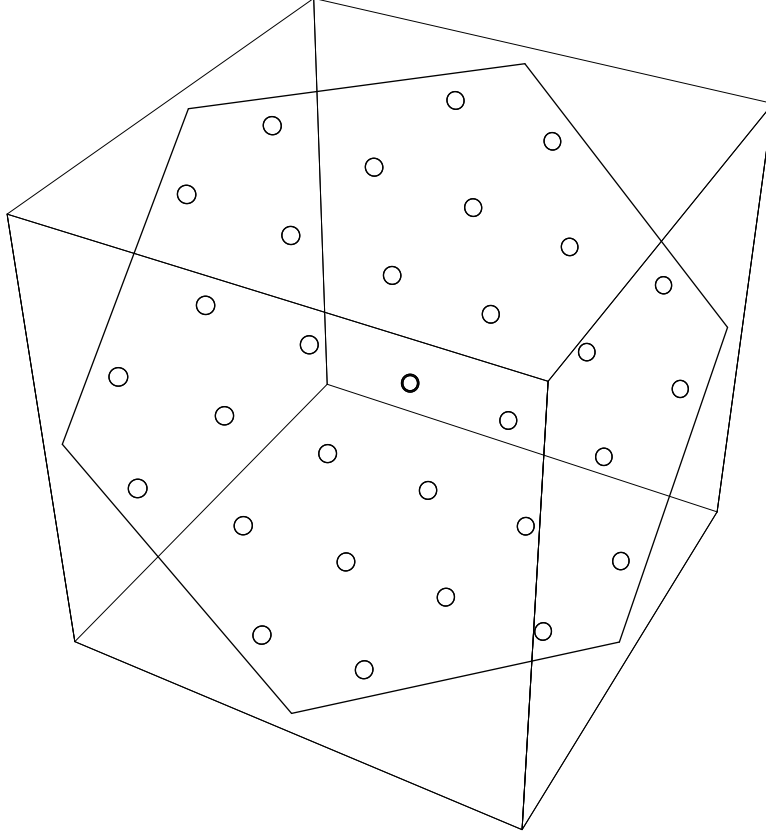


Figure 1: The state space for  $n = 3$ . A bounding box with  $B = 5$  is shown. Only points with  $x_1 - x_2 = x_2 - x_3 = 0 \pmod 3$  are reachable, and only those are shown.

Coppersmith et al [4] report certain patterns that cannot occur in a state  $x$  of maximum moment. We had some moderate success in finding more non-occurring patterns.

Suppose we are given an initial error vector  $x$ . Had it maximum moment, no algorithm, no matter what the schedule of arrivals be, would produce a new error vector  $x'$  with higher moment.

Because we cannot specify algorithms in their entirety, we conceive of this determination as a game: one player ( $\exists$ ) chooses who the arrivals are, while the other player ( $\forall$ , who could be an actual algorithm, as well as a random procedure) chooses the driver. If an error vector of higher moment is reached, player  $\exists$  wins. If an error vector is reached for a second time, then player  $\forall$  wins. Sooner or later, one of the players must win (note that there are finitely many error vectors with moment bounded by  $\|x\|^2$ ).

The question now becomes: Does any of the players have a winning strategy? If  $\exists$  does, then a judicious choice of the schedule of arrivals can force *any* algorithm to push the error vector to a moment higher than  $x$ . If instead  $\forall$  has a winning strategy, it means that there is an algorithm that can prevent the moment from increasing further.

Let's summarize this procedure. We consider the following functions:

- TestPt( $x$ ) determines whether there is a strategy to push from  $x$  to a vector of higher mo-

ment;

- $\text{TestMoment}(x, m)$  determines whether there is a strategy to push from  $x$  to a vector with moment higher than  $m$ ;
- $\text{TestSet}(x, m, s)$  determines whether there is a strategy to push from  $x$  to a vector with moment higher than  $m$ , with  $s$  being  $\exists$ 's first move;
- $\text{TestDriver}(x, m, s, i)$  determines whether there is a strategy to push from  $x$  to a vector with moment higher than  $m$ , with  $s$  being  $\exists$ 's first move, and  $i \in s$  being  $\forall$ 's counter-move.

Then the procedure can be summarized as follows:

$$\begin{aligned} \text{TestPt}(x) &\equiv \text{TestMoment}(x, \|x\|^2). \\ \text{TestMoment}(x, m) &\equiv \bigvee_{s \text{ set of arrivals}} \text{TestSet}(x, m, s). \\ \text{TestSet}(x, m, s) &\equiv \bigwedge_{i \in s} \text{TestDriver}(x, m, s, i). \\ \text{TestDriver}(x, m, s, i) &\equiv x' \leftarrow x + \delta(s, i); \\ &\quad \text{if } x' \text{ was visited before } \mathbf{return} \text{ false}; \\ &\quad \text{if } \|x\|^2 > m \mathbf{return} \text{ true}; \\ &\quad \text{if iteration depth was reached } \mathbf{return} \text{ false}; \\ &\quad \mathbf{return} \text{ TestMoment}(x', m). \end{aligned}$$

For a more detailed algorithm, see Appendix A.

As the search tree grows exponentially, the following considerations should be kept in mind while implementing the algorithm, in order to reduce the computer's running time. First, special data structures should be employed to avoid repetitions while searching through all subsets of arrivals. One specific consideration is to formulate an algorithm to keep track of all visited configurations. Then the search down the tree can be stopped once no more new configurations are encountered. In this case, then one in fact shows that the considered error vector cannot be excluded (see the next section for more on this issue). Second, subsets of arrivals may be chosen in a random or well-organized fashion to increase the likelihood of meeting the ultimate subset of arrivals early in the search.

### 3 Results and Observations

On trying to close the gap between the upper bound for the greedy algorithm  $\frac{n-1}{2} - \frac{1}{L}$  and the lower bound for any algorithm  $\frac{3(n-1)}{8} - \frac{1}{4}$ , one needs to eliminate as many cases of error vectors as possible. In order to test if an error vector can be excluded, one needs to "search" the tree of evolution of the error vector. As conveyed in the problem statement, we need to find (a tree of) subsets of arrivals  $s$  guaranteeing that, whatever the choices of drivers, the second moment will eventually get higher than in the initial configuration. The algorithm presented does exactly this.

Said algorithm has been implemented in a few different languages such as Maple, Mathematica and Matlab. One of the Matlab versions implements the detailed algorithm presented in Appendix A and returns whether or not the input error vector can be excluded at a specified search depth. If the input error vector can be excluded at the search depth, it also returns one possible evolution of the error vector. The Maple version uses the greedy algorithm to answer the same question. This is done for quick checks since if an error vector cannot be excluded by using the greedy algorithm then it cannot for any other algorithm. The Mathematica version is capable of eliminating repetitions and also has an option to pick random subsets of arrivals. It also returns a tree of evolution.

During the course of the workshop, using these implementations, we were able to exclude a number of error vectors. The following table contains these cases in terms of lists of gaps (Note: Some of these lists of gaps have already been reported in Coppersmith et al. [4]). See Appendix B for more details of the schedules of arrivals.

$\left(\frac{0}{12}\right)$	$\left(\frac{1}{12}\right)$	$\left(\frac{2}{12}\right)$	$\left(\frac{3}{12}\right)$
$\left(\frac{4}{12}\right)$	$\left(\frac{5}{12}\right)$	$\left(\frac{6}{12}, \frac{6}{12}\right)$	$\left(\frac{6}{12}, \frac{7}{12}\right)$
$\left(\frac{6}{12}, \frac{8}{12}\right)$	$\left(\frac{6}{12}, \frac{9}{12}\right)$	$\left(\frac{6}{12}, \frac{10}{12}\right)$	$\left(\frac{6}{12}, \frac{11}{12}\right)$
$\left(\frac{7}{12}, \frac{6}{12}\right)$	$\left(\frac{7}{12}, \frac{7}{12}\right)$	$\left(\frac{7}{12}, \frac{8}{12}\right)$	$\left(\frac{7}{12}, \frac{9}{12}\right)$
$\left(\frac{8}{12}, \frac{6}{12}\right)$	$\left(\frac{8}{12}, \frac{7}{12}\right)$	$\left(\frac{9}{12}, \frac{6}{12}\right)$	$\left(\frac{9}{12}, \frac{7}{12}\right)$
$\left(\frac{10}{12}, \frac{6}{12}\right)$	$\left(\frac{11}{12}, \frac{6}{12}\right)$	$\left(\frac{6}{12}, \frac{12}{12}, \frac{6}{12}\right)$	$\left(\frac{6}{12}, \frac{12}{12}, \frac{7}{12}\right)$
$\left(\frac{6}{12}, \frac{12}{12}, \frac{8}{12}\right)$	$\left(\frac{6}{12}, \frac{12}{12}, \frac{9}{12}\right)$	$\left(\frac{7}{12}, \frac{10}{12}, \frac{7}{12}\right)$	$\left(\frac{7}{12}, \frac{10}{12}, \frac{8}{12}\right)$
$\left(\frac{7}{12}, \frac{11}{12}, \frac{7}{12}\right)$	$\left(\frac{7}{12}, \frac{11}{12}, \frac{8}{12}\right)$	$\left(\frac{7}{12}, \frac{12}{12}, \frac{6}{12}\right)$	$\left(\frac{12}{12}, \frac{12}{12}, \frac{12}{12}\right)$
$\left(\frac{12}{12}, \frac{12}{12}, \frac{12}{12}\right)$	$\left(\frac{12}{12}, \frac{12}{12}, \frac{12}{12}\right)$	$\left(\frac{12}{12}, \frac{12}{12}, \frac{12}{12}\right)$	$\left(\frac{8}{12}, \frac{8}{12}, \frac{8}{12}\right)$
$\left(\frac{8}{12}, \frac{12}{12}, \frac{9}{12}\right)$	$\left(\frac{8}{12}, \frac{9}{12}, \frac{8}{12}\right)$	$\left(\frac{9}{12}, \frac{8}{12}, \frac{8}{12}\right)$	$\left(\frac{12}{12}, \frac{12}{12}, \frac{12}{12}\right)$

The goal here would be to be able to exclude all sequences of gaps averaging less than a certain constant  $\alpha$ . This way, any (ordered) error vector  $(x_i)$  would have gaps  $x_{i+1} - x_i \geq \alpha$ , and so  $x_n - x_1 \geq \alpha(n - 1)$ , and  $\max_i |x_i| \geq \frac{\alpha}{2}(n - 1)$  (this is a rough approximation). As Coppersmith et al. [4] show that the choice  $\alpha \geq \frac{3}{4}$  is possible, we tried to exclude gaps that would allow us to raise that value.

The following are a few observations which we have made but did not implement or prove in our algorithm. First, it is possible to show if a given error vector cannot be excluded. As we consider only fair algorithms, the error is bounded. Thus, the corresponding number of possible configurations of an error vector is finite. If one exhausts all the possible configurations but cannot find a schedule which results in higher second moment, then the given error vector cannot be excluded. Second, at a given stage, the greedy algorithm is the most optimal in that it always returns the lowest moment amongst those of the set of all possible algorithms. Third, we only consider the cases with up to and including 4 arrivals. In this scenario, in order to close the gap between the upper bound and the lower bound mentioned earlier in the problem statement, we should be able to exclude the following lists of gaps  $\left(\frac{7}{12}, \frac{11}{12}, \frac{9}{12}\right)$ ,  $\left(\frac{8}{12}, \frac{10}{12}, \frac{9}{12}\right)$ ,  $\left(\frac{9}{12}, \frac{9}{12}, \frac{9}{12}\right)$ . As to this point of writing this report, our routines have not been able to determine if these cases are excludable.

## 4 Future Work

There are a few directions for future exploration. One can first keep trying to see if the mentioned lists of gaps  $(\frac{7}{12}, \frac{11}{12}, \frac{9}{12})$ ,  $(\frac{8}{12}, \frac{10}{12}, \frac{9}{12})$ ,  $(\frac{9}{12}, \frac{9}{12}, \frac{9}{12})$  can in fact be excluded. If it is the case, then the lower bound for the carpool problem using any possible algorithm can be increased; thus, the gap between the two bounds is smaller. If it is not the case, then one should consider more possible subsets of arrivals. Unless one proves the conjecture analytically, considerations of more possible subsets of arrivals is indispensable in trying to increase the lower bound. During our course of work, we only considered the cases of up to and including 4 arrivals. As the number of arrivals increases, the search tree grows larger and wider. Therefore, coding considerations should be taken into account in order to speed up the computer's running time.

**Acknowledgement:** We would like to thank the IMA and the organizers for this wonderful opportunity and the hospitality. We would like to thank our mentor, Dr. Wu, for presenting to us a very interesting and challenging problem and guiding us through the problem solving process.

## References

- [1] R.Fagin and J.H.Williams, *A fair carpool scheduling algorithm*, IBM Journal of Research and development **27**(2) (1983), 133–139.
- [2] M.Ajtai, J.Aspnes, M.Naor, Y.Rabani, L.J.Schulman and O.Waarts, *Fairness in Scheduling*, Journal of Algorithms **29**(2) (1998), 306–357.
- [3] Moni Naor, *On fairness in the carpool problem*, Journal of Algorithms **55**(1) (2005), 93–98.
- [4] Don Coppersmith, Tomasz J.Nowicki, Guiseppe A.Paleologo, Charles Tresser, Chai Wah Wu, *The Optimality of the On-line greedy algorithm in carpool and chairman assignment problems*, IBM Research Report, (2005)

## A Algorithm

### A.1 Algorithms

The following algorithm determines whether or not a given error vector can be excluded from consideration for a higher lower bound in the carpool problem.

**Definitions:**  $d$  - search depth;  $E = (E_i)$  - error vector;  $flag$  - excludability of  $E$ ;  $i$  - designated driver;  $k$  - number of arrivals;  $k_{max}$  - maximum number of arrivals;  $k_{min}$  - minimum number of arrivals;  $l$  - search level;  $M$  - second moment of a given  $E$ ;  $M_n$  - updated second moment of  $E$ ;  $M_o$  - initial second moment of  $E$ ;  $n$  - number of participants;  $S$  - set of subsets  $s$  of  $k$  arrivals, where  $k \in \mathbb{Z}^+$  and  $k \leq n$

**ALGORITHM 1. Determine the Excludability of an Error Vector of a Set of  $n$  Participants**

**Inputs:**  $E, d, k_{min}, k_{max}$

**Output:**  $flag$

**Algorithm:**

1. Compute  $M_o$  (See **ALGORITHM 1a**)
2. Set  $l = 1$
3. While  $s \in S$
4.   Increment  $l$
5.   Set  $flag = 1$
6.   If  $k_{min} \leq k \leq k_{max}$
7.     For  $i = 1, 2, \dots, k$
8.       Update  $E$  (See **ALGORITHM 1b**)
9.       Compute  $M_n$
10.       If  $M_n \leq M_o$
11.         Set  $flag = 0$
12.         If  $l \leq d$
13.         Go to 3.
14.         Else
15.         Break out of For
16.         EndIf
17.       EndIf
18.       If  $flag = 0$
19.         Break out of For
20.       EndIf
21.   EndFor
22.   Else
23.     Set  $flag = 0$
24.   EndIf
25.   If  $flag = 1$
26.     Break out of While
27.   EndIf
28. End While

**ALGORITHM 1a. Compute the Second Moment of an Error Vector**

**Input:**  $E$

**Output:**  $M$

**Algorithm:**

1. Compute  $M = \sum_i E_i^2$

**ALGORITHM 1b. Update an Error Vector**

**Inputs:**  $E, i, k, k_{min}, k_{max}$

**Output:**  $E$

**Algorithm:**

1. Compute  $m = lcm(k_{min}, k_{max})$
2. Compute  $inc = (k - 1) * m/k$
3. Compute  $dec = m/k$
4. Subtract  $inc$  from driver  $i$
5. Add  $dec$  from  $k - 1$  passengers

## A.2 Example

In the following example, we would like to check whether the error vector  $E = (0, \frac{3}{6}, \frac{7}{6})$  can be excluded from consideration for a higher lower bound. In this example,  $n = 3$ ,  $M_o = 58$ ,  $k_{min} = 2$ ,  $k_{max} = 3$ ,  $S = \{\{\}, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$ . For simplicity, let us write  $E$  as  $E = (0, 3, 7)$ . One can check that  $\forall s \in \{\{\}, \{1\}, \{2\}, \{3\}\}$ ,  $M$  would not change. Otherwise, the evolution is as follows:

$l$	$s$	$i$	$E$	$M_n$	$> M_o$
1			(0, 3, 7)		
2	{1, 2}	1	(-3, 6, 7)	94 > 58	✓
		2	(3, 0, 7)	58	×
	{1, 3}	1	(-3, 3, 10)	118 > 58	✓
		3	(3, 3, 4)	34 < 58	×
	{2, 3}	2	(0, 0, 10)	100 > 58	✓
		3	(0, 6, 4)	52 < 58	×
	{1, 2, 3}	1	(-4, 5, 9)	122 > 58	✓
		2	(2, -1, 9)	86 > 58	✓
		3	(2, 5, 3)	38 < 58	×

At this level ( $l = 2$ ), one can pick any  $s$  and continue the search procedure for updated  $E$  with  $M_n \leq M_o$ . For instance, we pick  $s = \{1, 2, 3\}$  and continue with updated  $E = (2, 5, 3)$  as follows:

$l$	$s$	$i$	$E$	$M_n$	$> M_o$
2			(2, 5, 3)	38 < 58	×
3	{1, 3}	1	(-1, 5, 6)	62 > 58	✓
		3	(5, 5, 0)	50 < 58	×
...					
3			(5, 5, 0)	50 < 58	×
4	{1, 2}	1	(2, 8, 0)	68 > 58	✓
		2	(8, 2, 0)	68 > 58	✓

Then one can conclude that the error vector  $E = (0, \frac{3}{6}, \frac{7}{6})$  can be excluded from consideration for a higher lower bound due to the following sequence of evolution:  $(0, 3, 7) \rightarrow (2, 5, 3) \rightarrow (5, 5, 0) \rightarrow (2, 8, 0)$ , which results in an increase in second moment.

## B Schedules of Arrivals

In this appendix we show  $\exists$ 's winning strategy to exclude several sequences of gaps. Each node shows an error vector. For each state,  $\exists$  chooses the arrivals. Each choice of driver leads to a new error state. All the leaves have higher moment than the original state, proving that  $\exists$  does have a winning strategy.

