Changing Lanes on a Highway

Thomas Petig
Qamcom Research and Technology AB, Sweden

Elad M. Schiller
Chalmers University of Technology, Sweden

Jukka Suomela
Aalto University, Finland

Abstract

We study a combinatorial optimization problem that is motivated by the scenario of autonomous cars driving on a multi-lane highway: some cars need to change lanes before the next intersection, and if there is congestion, cars need to slow down to make space for those who are changing lanes. There are two natural objective functions to minimize: (1) how long does it take for all traffic to clear the road, and (2) the total number of maneuvers. In this work, we present an approximation algorithm for solving these problems in the two-lane case and a hardness result for the multi-lane case.

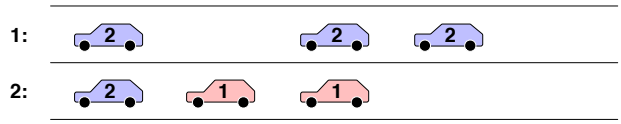
2012 ACM Subject Classification Theory of computation → Discrete optimization

Keywords and phrases Collaborative agents, vehicle scheduling, traffic optimization, approximation algorithms

Digital Object Identifier 10.4230/OASIcs.ATMOS.2018.9

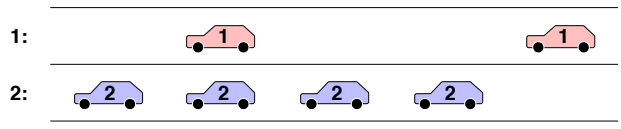
1 Introduction

Consider a fleet of autonomous vehicles driving on a two-lane highway:



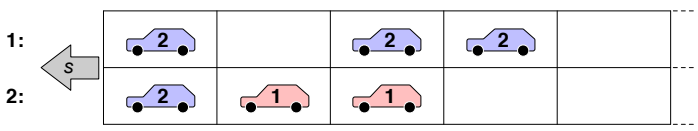
.

Each car is labeled with a lane number, 1 or 2, indicating where it needs to be before the next intersection. Our task is to instruct the cars to adjust their speed and change lanes so that all cars with label ℓ are on lane ℓ:



.

We discretize the traffic by assuming that there is a grid of *slots* that is moving at some fixed speed s (for example, s is the speed limit of the highway), and each car occupies one slot (there are infinitely many free slots behind the last cars):

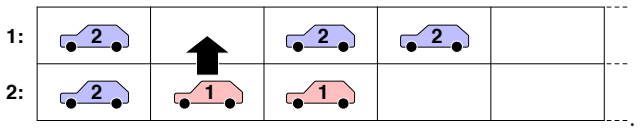


.

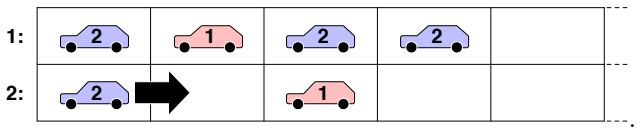
18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018).
Editors: Ralf Borndörfer and Sabine Storandt; Article No. 9; pp. 9:1–9:15
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

If there are no steering maneuvers, each car will remain in its current slot (i.e., it is driving along the current lane, at a constant speed s). We can use the following maneuvers to alter the relative positions of the cars.

First, a car that is **currently on the wrong lane** can **switch lanes**, assuming there is an empty slot next to it:



Second, any car can **slow down** a bit to move backwards relative to the traffic around it, assuming there is an empty slot behind it:



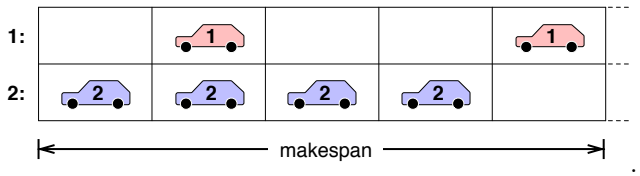
We emphasize that we do not allow cars that are on the right lane to switch lanes any more; permitting such maneuvers would also give rise to an interesting problem formulation to be studied in future work.

1.1 Objectives

It is easy to find a feasible solution by following a simple greedy strategy, e.g., for each car x that is on the wrong lane, slow down all cars behind x on either lane to make space for x to move to the right lane. However, this is not an optimal strategy in the general case.

We will consider the following objective functions that we would like to minimize:

- **Makespan**: What is the last non-empty slot that is occupied by a car in the final configuration? Intuitively, we measure here *how much do we stretch the traffic*, or equivalently, *how long does it take for all traffic to clear the road*:



- **Total cost**: What is the total number of steering maneuvers (switching lanes or slowing down) that we need to solve the problem? Note that in our problem formulation, the number of lane changes is simply equal to the number of cars on the wrong lane, so the interesting question is the number of slow-down operations. Intuitively, we measure here the *average delay for the traffic*.

To focus on the more interesting algorithmic aspects, we present our algorithms from the perspective of a global omniscient entity that has a full control over all vehicles. However, the same ideas can be applied in distributed and online settings.

1.2 Contributions and open questions

We develop a polynomial-time algorithm for the **two-lane** version of the lane-changing problem. The algorithm finds a solution that **minimizes the makespan** and that is also a **1.5-approximation of the minimum total cost**. Moreover, we show that a natural **multi-lane** extension of the problem is **NP-hard**.

Our work also suggests the following natural question for further research: is it possible to find an exact solution for the minimum-cost two-lane version in polynomial time?

2 Related work

Tile-sliding puzzles. We note the resemblance between the problem studied by us in this work and combinatorial puzzles such as the "15-puzzle" [8,17], which is a game that considers a four by four matrix that has 15 tiles, labeled with numbers from 1 to 15 in an arbitrary order. The goal of the game is to slide the tiles so that the tiles are ordered. For large-scale versions of the n-puzzle, finding an optimum solution is NP-hard [13,14]. Our problem can also be seen as a tile-sliding puzzle, but differs from the 15-puzzle in the following aspects: many tiles may have the same label, the label only determines the final row (and not column), and our moves are more restricted (for example, tiles cannot slide right). Feigenbaum et al. [3] formulated a number of graph problems for the semi-streaming model. Unlike their model, our problem does not allow a pair of nearby agents to swap cells.

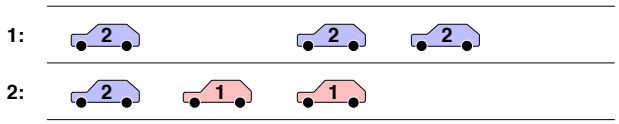
Vehicular control. Problems related to lane-change consider traffic streams from the point of view of vehicular control [2,6,12], traffic flow control [9], the scheduling of lane changes for autonomous vehicles [1], assessment of the situation before changing lane [15], and negotiation before lane changing [16] to name a few. It is often the case, as in [12], that these problems consider a small set of nearby vehicles that need to coordinate a single lane-change maneuver. A number of recent efforts, such as the European project AutoNet2030 [16], considers the need to perform lane changes in congested traffic situations, as we do in this paper. Their study focuses on distributed mechanisms, i.e., the communication protocols, for enabling coordinated lane changes whereas this work focuses on the algorithmic question of minimizing the number of maneuvers.

Traffic models. Cellular automata are often used for microscopic traffic flow prediction [10]. These models resembles the one of the studied problem in the sense that each vehicle occupies a single cell. However, Nagel [10] considers cellular automata that move the vehicles forward, whereas our model considers vehicles that can merely change their current lanes or delay – we do not aim at predicting traffic patterns and just aim at minimizing the number of delays and lane changes. The systematic approach presented in [11] shows that their lane change rules can provide "realistic results" with respect to the system ability to offer an accurate traffic prediction. A complementary approach for studying the effect of lane-change behavior via cellular automata [7] is the observation of driver behavior [4,18].

3 Preliminaries

Matrix notation. We will interpret the highway as a matrix with two rows and infinitely many columns; rows correspond to lanes and matrix elements correspond to slots. The rows are numbered $i = 1, 2$ and the columns are numbered $j = 1, 2, \ldots$. We use values 1 and 2 to

denote cars with target lanes 1 and 2, respectively, and we use the symbol ∘ to emphasize that a slot is empty. For example, the configuration



is represented as a matrix

$$\begin{bmatrix} 2 & \circ & 2 & 2 & \circ & \cdots \\ 2 & 1 & 1 & \circ & \circ & \cdots \end{bmatrix},$$

which we may write for brevity simply as

$$\begin{bmatrix} 2 & \circ & 2 & 2 \\ 2 & 1 & 1 & \circ \end{bmatrix}.$$

Legal moves. In the lane-changing problem, we can apply the following operations to any part of the configuration matrix.

- *Switch* (switch lanes, i.e., move up or down):

$$\begin{bmatrix} 2 \\ \circ \end{bmatrix} \mapsto \begin{bmatrix} \circ \\ 2 \end{bmatrix}, \quad \begin{bmatrix} \circ \\ 1 \end{bmatrix} \mapsto \begin{bmatrix} 1 \\ \circ \end{bmatrix}.$$

- *Delay* (slow down, i.e., move right):

$$\begin{bmatrix} 1 & \circ \end{bmatrix} \mapsto \begin{bmatrix} \circ & 1 \end{bmatrix}, \quad \begin{bmatrix} 2 & \circ \end{bmatrix} \mapsto \begin{bmatrix} \circ & 2 \end{bmatrix}.$$

Pairs. A *pair* is one column of the configuration; an *input pair* is a column that occurs in the input. For brevity, a column $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$ is called an (x, y)-pair.

Solution. A *feasible configuration* is a configuration in which all non-empty slots of row 1 contain label 1, and all non-empty slots of row 2 contain label 2. That is, each car is on its target lane.

A *feasible solution* to the lane-changing problem is a sequence of legal moves that turns the given input configuration into a feasible configuration. The *cost* of a solution is the number of moves. The *makespan* of a solution is the largest j such that column j of the final configuration contains a car.

4 Roadmap

Three problems. To develop an algorithm for solving the lane-changing problem, it will be helpful to also consider two variants of it:

P0. The original lane-changing problem, as defined above.

P1. A restricted version of P0: a solution is feasible only if each car that was involved in a $(2, 1)$-input pair has been delayed at least once.

P2. A relaxed version of P1: multiple cars may occupy the same slot in intermediate configurations.

In P2 we will use notation $1^a 2^b$ to denote a slot with a cars of label 1 and b cars of label 2. A legal P2-move is hence, for example,

$$\begin{bmatrix} 1^3 2^3 & 1^5 2^5 \end{bmatrix} \mapsto \begin{bmatrix} 1^3 2^2 & 1^5 2^6 \end{bmatrix}.$$

The final configuration in P2 has to be feasible in the usual sense: each slot contains at most one car, and each car is in the right lane. Also note that one move can only change the position of one car.

Simple examples. Consider the input

$$\begin{bmatrix} 2 \\ 1 \end{bmatrix}.$$

A feasible P0-solution might take, e.g., the following steps (cost 3, makespan 2):

$$\begin{bmatrix} 2 \\ 1 \end{bmatrix} \mapsto \begin{bmatrix} 2 & \circ \\ \circ & 1 \end{bmatrix} \mapsto \begin{bmatrix} \circ & \circ \\ 2 & 1 \end{bmatrix} \mapsto \begin{bmatrix} \circ & 1 \\ 2 & \circ \end{bmatrix}.$$

This would not be a feasible P1-solution, though, as there is a car with label 2 that was part of a $(2, 1)$-pair in the input but the car was not delayed. A feasible P1-solution might take the following steps (cost 4, makespan 2):

$$\begin{bmatrix} 2 \\ 1 \end{bmatrix} \mapsto \begin{bmatrix} 2 & \circ \\ \circ & 1 \end{bmatrix} \mapsto \begin{bmatrix} \circ & \circ \\ 2 & 1 \end{bmatrix} \mapsto \begin{bmatrix} \circ & 1 \\ 2 & \circ \end{bmatrix} \mapsto \begin{bmatrix} \circ & 1 \\ \circ & 2 \end{bmatrix}.$$

In a feasible P2-solution we could also take the following route in which we have multiple cars in one slot in an intermediate configuration (but this is not any cheaper; we still have cost 4 and makespan 2):

$$\begin{bmatrix} 2 \\ 1 \end{bmatrix} \mapsto \begin{bmatrix} \circ \\ 12 \end{bmatrix} \mapsto \begin{bmatrix} 1 \\ 2 \end{bmatrix} \mapsto \begin{bmatrix} \circ & 1 \\ 2 & \circ \end{bmatrix} \mapsto \begin{bmatrix} \circ & 1 \\ \circ & 2 \end{bmatrix}.$$

Preliminary observations. We emphasize that problems P1 and P2 are not interesting in their own right; we only care about problem P0. Both P0 and P2 can be seen as relaxations of P1, but they are relaxations of a very different nature:

- A feasible solution to P1 is also a feasible solution to P0, but it might take some additional steps that are only necessary to handle $(2, 1)$-pairs.
- A feasible solution to P1 is also a feasible solution to P2, but it might take some additional steps that are only necessary to ensure there is at most one car per slot.

At first, P2 and P0 seem to be incomparable. A P2-solution is not necessarily a P0-solution, or vice versa. But as we will see in this work, an algorithm for solving P2 can be a helpful starting point in solving P0, too.

Key ideas. The key insights of our work are these results that we will prove:

- For P2 there is always a solution that *simultaneously minimizes both makespan and cost*.
- Problem P1 can be solved with the *same makespan and cost* as problem P2.
- A makespan-optimal P1-solution is also a makespan-optimal P0-solution.
- A cost-optimal P1-solution is also a 1.5-approximation of a cost-optimal P0-solution.

We will use the above ideas to solve problem P0 as follows:

- In Section 5.1, we design **algorithm A2** that will find a P2-solution that is simultaneously cost-optimal and makespan-optimal.

- In Section 5.2, we design **algorithm A1** that finds a P1-solution that has the same cost and makespan as the P2-solution returned by A2. As P2 is a relaxation of P1, it follows that A1 returns a cost-optimal and makespan-optimal P1-solution.
- Now it is clear that A1 also returns a P0-solution, as P0 is a relaxation of P1. However, we will still need to prove that the solution returned by A1 is a makespan-optimal P0-solution and also a 1.5-approximation of a cost-optimal P0-solution. The proof is given in Section 5.3.

5 Algorithm details

Notation. Let W be the total number of cars that are on the wrong lane in the input configuration. Any feasible solution contains exactly W switch operations. Hence a minimum-cost solution is a solution that minimizes the number of delay operations.

5.1 Solving problem P2

Flow equations. Let us first develop some *necessary* conditions that characterize feasible solutions for P2 (and hence they are also necessary conditions for a feasible solution of P1).

Consider some feasible solution Y. Let $\ell = 1, 2$ be a label. We will consider the *flow* of cars of label ℓ:

- $s_\ell(j)$ is the number of cars of label ℓ in column j in the input configuration,
- $t_\ell(j)$ is the number of cars of label ℓ in column j in the final configuration,
- $f_\ell(j)$ is the number of times a car of label ℓ is moved from column j to column $j+1$.

Recall that the columns are numbered $j = 1, 2, \ldots$, but for convenience, we also define $f_\ell(0) = 0$ so that we can always refer to $f_\ell(j-1)$. Let us now define the grand total of flow that we will need to handle at column j:

$$g_\ell(j) = f_\ell(j-1) + s_\ell(j). \tag{1}$$

As no car is lost or created, flow is conserved:

$$g_\ell(j) = t_\ell(j) + f_\ell(j). \tag{2}$$

In the final configuration we have got at most one car per slot:

$$t_\ell(j) \leq 1. \tag{3}$$

Hence by (2) and (3) we necessarily have

$$f_\ell(j) \geq g_\ell(j) - 1. \tag{4}$$

By the definition of problem P1 (and hence P2), cars in $(2,1)$-input pairs are always delayed at least once. To capture this, define the indicator function p as follows:

$p_\ell(j) = 1$ if there is a $(2,1)$-input pair in column j in the input configuration.

Using this notation, we have for each $j = 1, 2, \ldots$

$$f_\ell(j) \geq p_\ell(j). \tag{5}$$

Now t and f may depend on the particular solution Y, but s and p only depend on the input configuration. For any given input, we can recursively calculate a *minimal* flow f^* that satisfies (1), (4), and (5):

$$g^*_\ell(j) = f^*_\ell(j-1) + s_\ell(j) \qquad \text{for all } j = 1, 2, \ldots, \tag{6}$$

$$f^*_\ell(j) = \max\bigl\{p_\ell(j), g^*_\ell(j) - 1\bigr\} \qquad \text{for all } j = 1, 2, \ldots. \tag{7}$$

Again we follow the convention that $f^*_\ell(0) = 0$ so that $f^*_\ell(j-1)$ is well-defined for every column j. Note that for all ℓ and j and for any feasible flow f, we have by construction $f^*_\ell(j) \le f_\ell(j)$. Hence we can make the following observations:

▶ **Lemma 1.** *The cost of any feasible P2-solution is at least* $W + \sum_{\ell,j} f^*_\ell(j)$.

▶ **Lemma 2.** *For all* ℓ *and* j*, if* $g^*_\ell(j) > 0$*, then the makespan of any feasible P2-solution is at least* j.

Algorithm A2. Now it is sufficient to design an algorithm that moves cars precisely according to the minimal flow f^*; if we can do that, the solution will be both cost-optimal and makespan-optimal.

But this is easy: First each car switches to the right lane; this takes W moves. Then we follow (6)–(7) for columns $j = 1, 2, \ldots$ in ascending order: first we move $f^*_\ell(1)$ cars of label ℓ from column 1 to column 2, then we have $g^*_\ell(2)$ cars of label ℓ in column 2, etc. If we always move first those cars that were already present in a given slot in the input configuration, we will satisfy all constraints of problem P2, including the special rule about $(2,1)$-pairs.

We have now algorithm A2 that finds simultaneously cost-optimal and makespan-optimal solutions for P2. However, this is clearly not a solution for P1, as we may have multiple cars in one slot in intermediate configurations.

5.2 Solving problem P1

Idea. We now develop algorithm A1 that follows the same minimal flow f^*, but schedules the operations differently so that it produces a feasible solution to problem P1:

- Algorithm A2 "pushes" cars starting from the first cars.
- Algorithm A1 "pulls" cars starting from the last cars.

Our basic idea is to show that – with a little bit of planning ahead – we can move cars according to f^* without putting multiple cars in one slot.

In the algorithm we will update s, p, f^*, and g^* as we move cars around so that they refer to the current configuration, and not the input configuration. Eventually all cars will be in their final positions, there is no need to move anything, and f^* will be zero.

Trivial and tricky pairs. A pair of type $(\circ, 1)$ or $(2, \circ)$ is called a *trivial pair*. As the first step of the algorithm, each trivial pair will switch lanes; hence we eliminate all trivial pairs. Our algorithm will ensure that whenever we create new trivial pairs, they are also eliminated immediately.

A pair of type $(2,1)$ is called a *tricky pair*. We will make sure that the algorithm only eliminates tricky pairs and never create new tricky pairs. We will use p to keep track of the tricky pairs that were present in the input: Initially, $p_1(j) = p_2(j) = 1$ if we have a tricky pair in column j. Then whenever we delay a car with label ℓ in column j, we set $p_\ell(j) \leftarrow 0$.

Active and hot columns. We say that a column j is *ℓ-active* if we have $s_\ell(j) > 0$ and $f^*_\ell(j) > 0$. A column is *active* if it is ℓ-active for some ℓ.

A column is *hot* if it is the rightmost (last) active column. The hot column is called *ℓ-hot* if it is ℓ-active. (Note that there is at most one hot column, and the hot column is 1-hot, 2-hot, or both. Also note that the rightmost 1-active column is not necessarily 1-hot, as there might be a 2-active column that is further right, and vice versa.)

Intuitively, an active column contains some cars that are not in their final positions, and the hot column contains the last cars that are not in their final positions. As long as f^* is somewhere nonzero, there has to be an active column, and hence also a hot column.

The following lemmas summarize the key properties that we use.

▶ **Lemma 3.** *Assume that*

- *there are no trivial pairs,*
- *column j is ℓ-hot,*
- *slot (ℓ, j) contains a car with label ℓ.*

Then slot $(\ell, j+1)$ has to be empty.

Proof. Assume w.l.o.g. that $\ell = 1$; the case of $\ell = 2$ is analogous.

If column $j+1$ contains a car of label 1, then $f^*_1(j) + s_1(j+1) \geq 2$, and therefore $f^*_1(j+1) \geq 1$. But this would mean that column $j+1$ is active, which contradicts the assumption that j is hot (i.e., the rightmost active column).

If column $j+1$ does not contain any car of label 1, but slot $(\ell, j+1)$ is not empty, the only possibility is that column $j+1$ contains a pair $(2, 2)$. But then we would have $s_2(j+1) \geq 2$ and $f^*_2(j+1) \geq 1$ and again $j+1$ would be active. ◀

▶ **Lemma 4.** *Assume that*

- *column j is hot,*
- *column j contains a tricky pair.*

Then column $j+1$ has to be empty.

Proof. The tricky pair implies that $f^*_1(j) \geq 1$ and $f^*_2(j) \geq 1$. If column $j+1$ contains a car with label ℓ in the current configuration, we will have $s_\ell(j+1) \geq 1$, and hence $g^*_\ell(j+1) \geq 2$ and $f^*_\ell(j+1) \geq 1$. Therefore, column $j+1$ would be active, which contradicts the assumption that j is hot (i.e., the rightmost active column). ◀

Algorithm A1. If we do not have any hot columns, we are done. Otherwise, let j be an ℓ-hot column. Our goal is to show that the algorithm can make progress and delay at least one car in the hot column. We have two cases:

1. Slot (ℓ, j) contains a car with label ℓ: By Lemma 3, we can delay the car with label ℓ in row ℓ.

$$\begin{bmatrix} 1 & \circ \\ x & y \end{bmatrix} \mapsto \begin{bmatrix} \circ & 1 \\ x & y \end{bmatrix}, \qquad \begin{bmatrix} x & y \\ 2 & \circ \end{bmatrix} \mapsto \begin{bmatrix} x & y \\ \circ & 2 \end{bmatrix}$$

   This cannot create tricky or trivial pairs in column $j+1$. If this resulted in a trivial pair in column j, the algorithm then eliminates it with a switch, e.g.:

$$\begin{bmatrix} 1 & \circ \\ 1 & y \end{bmatrix} \mapsto \begin{bmatrix} \circ & 1 \\ 1 & y \end{bmatrix} \mapsto \begin{bmatrix} 1 & 1 \\ \circ & y \end{bmatrix}.$$

2. Slot (ℓ, j) does not contain a car with label ℓ: By the definition of an ℓ-hot column, there has to be a car ℓ somewhere in column j, and as we do not have trivial pairs, we must have a tricky pair. By Lemma 4 column $j+1$ is empty. Hence we can move car 1 from column j to column $j+1$, and this creates trivial pairs in both column j and column $j+1$. Then we perform two switch operations to eliminate the trivial pairs:

$$\begin{bmatrix} 2 & \circ \\ 1 & \circ \end{bmatrix} \mapsto \begin{bmatrix} 2 & \circ \\ \circ & 1 \end{bmatrix} \mapsto \begin{bmatrix} \circ & \circ \\ 2 & 1 \end{bmatrix} \mapsto \begin{bmatrix} \circ & 1 \\ 2 & \circ \end{bmatrix}.$$

(Note that here car 2 is not in its final position, $p_2(j)$ is still nonzero, the column remains active, it will eventually become hot, and the car will be moved right.)

Hence in all cases the algorithm can move at least one car, and we can calculate each move efficiently. By construction, both the cost and the makespan of algorithm A1 are the same as in the solution returned by algorithm A2; as A2 solved P2 optimally, and P2 is a relaxation of P1, we conclude that A1 solves P1 optimally.

5.3 Solving problem P0

Recall that P0 is a relaxation of P1. Hence we can directly use algorithm A1 to solve also P0. The following lemma shows that any P1-optimal solution is also a relatively good solution for P0.

▶ **Lemma 5.** *Assume that there is a solution X for P0 that uses W switch operations and D delay operations and has a makespan M. Then it is possible to find a solution Y for P1 that uses W switch operations and at most $D + \min(D, W)$ delay operations and has a makespan M.*

To prove the lemma, we show how to modify X to construct Y. We begin with definitions.

Bad cars and bad blocks. Consider the trajectories of the cars in solution X.

We say that a car is *switch-only* if it only switches lanes once and is never delayed. For example, a switch-only car with label 2 was initially in slot $(1, j)$ for some j, and in the final configuration it is in slot $(2, j)$ for the same j. Note that each column contains at most one switch-only car.

We say that a switch-only car is *bad* if it is part of a $(2, 1)$-input pair. We may have such in P0-solution X but we must not have them in P1-solution Y.

A *bad block of type* ℓ is a range of columns $j, j+1, \ldots, k-1$ such that:

1. Column j contains a bad car with label ℓ.
2. Each of columns $j+1, \ldots, k-1$ contains a switch-only car with label ℓ. (Some of these cars may also be bad.)
3. Column k does not contain any switch-only cars with label ℓ. (Note that this column may contain a switch-only car of the opposite type, and it may be bad.)

For brevity, we write $[j, k)$ for the range of columns $j, j+1, \ldots, k-1$. Note that if we have a bad car in column j, we can always find some k such that $[j, k)$ is a bad block.

Eliminating bad blocks. Our plan is that we identify the first bad block, and manipulate the solution locally so that none of the columns $[j, k)$ contain any bad cars. Then we repeat this until there are no bad blocks (and hence no bad cars) left.

Consider the first bad block $[j, k)$ that we have not yet eliminated; we write $L = k - j$ for the length of the bad block. W.l.o.g., assume that the bad car in column j has label 2; the other case is analogous. The input configuration of the bad block looks like

$$\begin{array}{cccc} {\scriptstyle j} & {\scriptstyle j+1} & & {\scriptstyle k-1} \\ \left[\begin{array}{cccc} 2 & 2 & \cdots & 2 \\ 1 & ? & \cdots & ? \end{array}\right], \end{array}$$

and an output configuration of the block looks like

$$\begin{array}{cccc} {\scriptstyle j} & {\scriptstyle j+1} & & {\scriptstyle k-1} \\ \left[\begin{array}{cccc} ? & ? & \cdots & ? \\ 2 & 2 & \cdots & 2 \end{array}\right]. \end{array}$$

Consider the trajectory of the car 1 that was originally in column j; this is called the *leading car*. The leading car was moved from column j to column $j + 1$ before the switch-only cars in columns j and $j + 1$ moved. It was also moved from column $j + 1$ to column $j + 2$ before the switch-only cars in columns $j + 1$ and $j + 2$ moved, etc. In the final configuration the leading car has to be outside the bad block. Inside the bad block, solution X performs at least L switch operations (L switch-only cars) and at least L delay operations (one leading car moved L times). Note that in our bookkeeping, we associate the cost of a delay operation with the source column.

Case 1: Empty slot follows. Now first consider the possibility that slot $(2, k)$ is empty in the final configuration. Then we can eliminate all bad cars within the bad block with L additional delay operations: delay the cars in $(2, k - 1)$, $(2, k - 2)$, $\ldots$, $(2, j)$ in this order. In essence, we turn

$$\begin{array}{ccccc} {\scriptstyle j} & {\scriptstyle j+1} & & {\scriptstyle k-1} & {\scriptstyle k} \\ \left[\begin{array}{cccc|c} ? & ? & \cdots & ? & ? \\ 2 & 2 & \cdots & 2 & \circ \end{array}\right] \end{array}$$

into

$$\begin{array}{ccccc} {\scriptstyle j} & {\scriptstyle j+1} & & {\scriptstyle k-1} & {\scriptstyle k} \\ \left[\begin{array}{cccc|c} ? & ? & \cdots & ? & ? \\ \circ & 2 & \cdots & 2 & 2 \end{array}\right]. \end{array}$$

Note that we modify column k which is outside the current bad block, and there might be another bad block starting at column k. However, it can be verified that what we do with block $[j, k)$ is compatible with what we do with the block starting at k.

Case 2: Non-empty slot follows. Now assume that slot $(2, k)$ is occupied in the final configuration; let us call it the *trailing car*. It has to be a car with label 2:

$$\begin{array}{ccccc} {\scriptstyle j} & {\scriptstyle j+1} & & {\scriptstyle k-1} & {\scriptstyle k} \\ \left[\begin{array}{cccc|c} ? & ? & \cdots & ? & ? \\ 2 & 2 & \cdots & 2 & 2 \end{array}\right]. \end{array}$$

By definition, it is not a switch-only car. Also the trailing car was not there initially; otherwise it would have blocked the path of the leading car.

Working backwards from the final position of the trailing car, we can see that the trailing car had to be the last car that occupied slot $(2, k - 1)$ before the switch-only car in column

$k-1$ moved there, and it also had to be the last car that occupied slot $(2, k-2)$ before the switch-only car in column $k-2$ moved there, etc. The trailing car could not have been originally position in any of these slots, as it would have blocked the way of the leading car. Hence at some point the trailing car followed the path $(2, j-1) \to (2, j) \to \ldots \to (2, k)$, and in each column the switch-only cars moved only after the trailing car was gone.

To recap, we have got in total $L+1$ cars with label 2 that were in some intermediate configuration placed as follows; we denote the trailing car with 2^*:

$$\begin{array}{c|cccccc|c} {\scriptstyle j-1} & {\scriptstyle j} & {\scriptstyle j+1} & & {\scriptstyle k-1} & {\scriptstyle k} \\ ? & 2 & 2 & \cdots & 2 & ? \\ 2^* & ? & ? & \cdots & ? & ? \end{array}. \tag{8}$$

The trailing car moves rightwards, and the switch-only car in column $k-1$ switches lanes. At some point we reach the following configuration; here 2? denotes a slot that is either empty (a switch-only car has not switched yet) or it contains a car with label 2:

$$\begin{array}{c|cccccc|c} {\scriptstyle j-1} & {\scriptstyle j} & {\scriptstyle j+1} & & {\scriptstyle k-1} & {\scriptstyle k} \\ ? & ? & ? & \cdots & \circ & ? \\ ? & 2? & 2? & \cdots & 2 & 2^* \end{array}. \tag{9}$$

Finally, the cars end up in the following positions:

$$\begin{array}{c|cccccc|c} {\scriptstyle j-1} & {\scriptstyle j} & {\scriptstyle j+1} & & {\scriptstyle k-1} & {\scriptstyle k} \\ ? & ? & ? & \cdots & ? & ? \\ ? & 2 & 2 & \cdots & 2 & 2^* \end{array}. \tag{10}$$

The key observation is this: at all points between configurations (8) and (9), the switch-only cars and the trailing car together form a barrier that blocks both lanes. Let us make this a bit more formal. Classify the cars in (8) as follows:

$$\begin{array}{c|ccccccc|ccc|c} & {\scriptstyle j-2} & {\scriptstyle j-1} & {\scriptstyle j} & {\scriptstyle j+1} & & {\scriptstyle k-1} & {\scriptstyle k} & {\scriptstyle k+1} & \\ \cdots & \text{left} & \text{left} & \text{middle} & \text{middle} & \cdots & \text{middle} & \text{right} & \text{right} & \cdots \\ \cdots & \text{left} & \text{middle} & \text{right} & \text{right} & \cdots & \text{right} & \text{right} & \text{right} & \cdots \end{array}.$$

Now *left cars* cannot move beyond column $k-2$ until we reach configuration (9), while all *right cars* will be in columns $k, k+1, \ldots$ in configuration (9). The left and the right cars do not interact between (8) and (9); they are always separated by the *middle cars* (which do not move beyond column k).

Let us modify the solution as follows: we skip all moves related to left and middle cars between (8) and (9); only right cars are permitted to move. We will reach the following configuration instead of (9); note that we have cleared the part below the switch-only cars:

$$\begin{array}{c|cccccc|c} {\scriptstyle j-1} & {\scriptstyle j} & {\scriptstyle j+1} & & {\scriptstyle k-1} & {\scriptstyle k} \\ ? & 2 & 2 & \cdots & 2 & ? \\ 2^* & \circ & \circ & \cdots & \circ & \circ \end{array}.$$

Then we move the rightmost switch-only car down and right:

$$\begin{array}{c|cccccc|c} {\scriptstyle j-1} & {\scriptstyle j} & {\scriptstyle j+1} & & {\scriptstyle k-1} & {\scriptstyle k} \\ ? & 2 & 2 & \cdots & \circ & ? \\ 2^* & \circ & \circ & \cdots & \circ & 2 \end{array}.$$

We repeat this for each switch-only car in columns $k-2$, $k-3$, …, j, and finally we move

the trailing car right. We reach the following configuration:

$$\begin{array}{c} \begin{array}{cc|ccccc|c} {\scriptstyle j-1} & & {\scriptstyle j} & {\scriptstyle j+1} & & & {\scriptstyle k-1} & {\scriptstyle k} \end{array} \\ \left[\begin{array}{cc|ccccc|c} ? & & \circ & \circ & \cdots & & \circ & ? \\ \circ & & 2^* & 2 & \cdots & & 2 & 2 \end{array}\right]. \end{array}$$

Now we have handled right cars (following their original schedule) and middle cars (following a new schedule). Finally we perform all operations between (8) and (9) that were related to the left cars, and we reach a configuration like this:

$$\begin{array}{c} \begin{array}{cc|ccccc|c} {\scriptstyle j-1} & & {\scriptstyle j} & {\scriptstyle j+1} & & & {\scriptstyle k-1} & {\scriptstyle k} \end{array} \\ \left[\begin{array}{cc|ccccc|c} ? & & ? & ? & \cdots & & \circ & ? \\ ? & & 2^* & 2 & \cdots & & 2 & 2 \end{array}\right]. \end{array}$$

Then we continue with the operations after (9), skipping those related to the middle cars, and we reach configuration of the following form:

$$\begin{array}{c} \begin{array}{cc|ccccc|c} {\scriptstyle j-1} & & {\scriptstyle j} & {\scriptstyle j+1} & & & {\scriptstyle k-1} & {\scriptstyle k} \end{array} \\ \left[\begin{array}{cc|ccccc|c} ? & & ? & ? & \cdots & & ? & ? \\ ? & & 2^* & 2 & \cdots & & 2 & 2 \end{array}\right]. \end{array}$$

The only difference in comparison with (10) is that the trailing car is left in column j, and switch-only cars have moved right by one step. Hence none of them are bad any more.

Now let us see what we achieved. We constructed another solution in which one bad block of length L was eliminated. We performed L additional delay operations with the switch-only cars, but on the other hand we saved L delay operations with the trailing car; hence the new solution has the same cost as the original solution.

Concluding the proof. For each bad block we do either L or 0 additional delay operations, and the block already contained at least L delay and L switch operations. Summing over all bad blocks, if they contain D delay and W switch operations in total, we do at most $\min(D, W)$ additional delay operations. The claim related to the number of operations follows.

Finally, we observe that the modified solution has the same makespan as the original solution; note that if we have a bad block $[j, k)$, then the makespan of the solution has to be at least k, and we do not move any cars beyond column k. This concludes the proof of Lemma 5.

▶ **Corollary 6.** *Let Y be a solution for P1 that is simultaneously makespan-optimal and cost-optimal. Then Y is also a feasible makespan-optimal solution for P0. Furthermore, the total number of moves in Y is at most* 1.5 *times the total number of moves in a cost-optimal P0 solution.*

Proof. If the optimum cost of P0 is $W + D$ moves, by Lemma 5 there is a solution for P1 with at most $W + D + \min(D, W) \leq 1.5(W + D)$ moves. Hence the optimum of P1 is at most 1.5 times as expensive as the optimum of P0. By Lemma 5 we also have the same makespan. ◀

In particular, if can use algorithm A1 to find an optimal P1-solution Y, and then apply Corollary 6 to show that the solution is a good approximation also for P0.

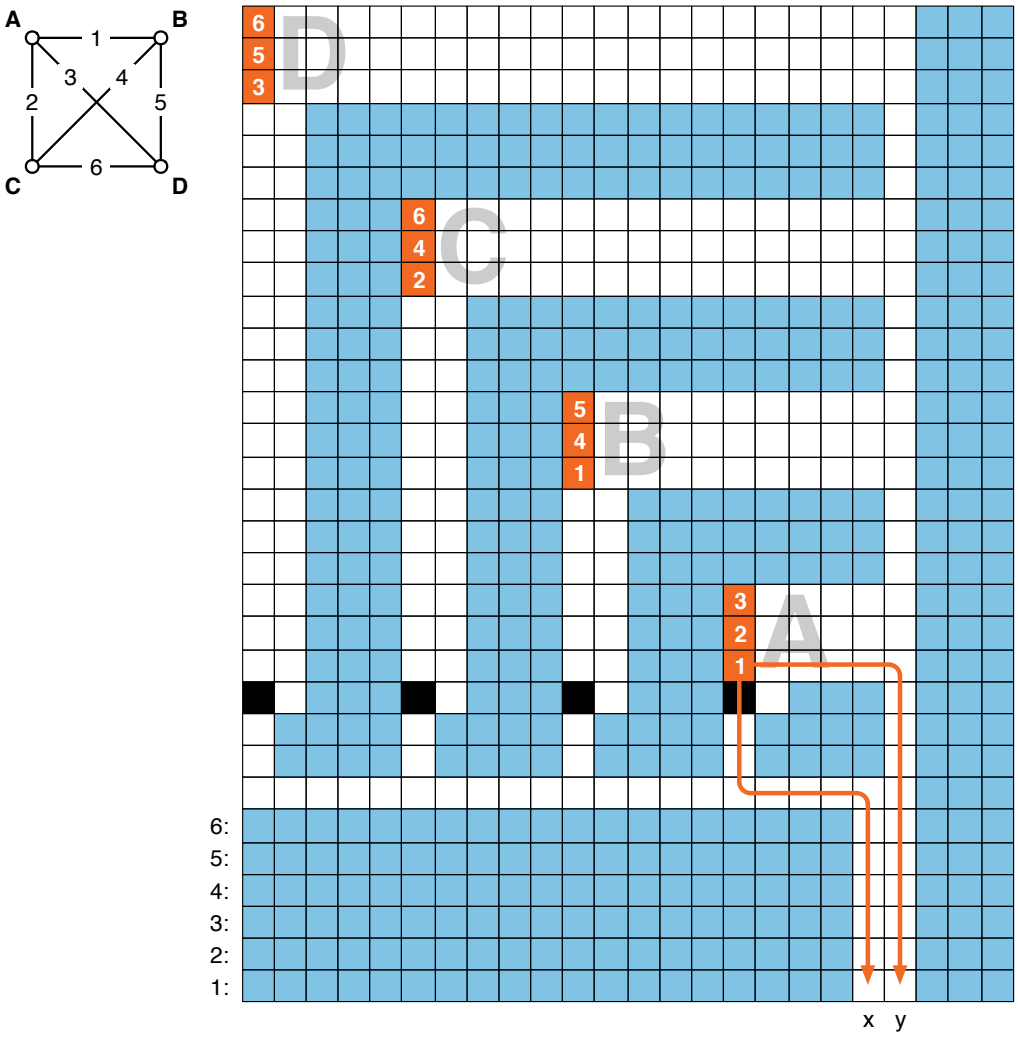


Figure 1 Reduction from the minimum vertex cover problem in 3-regular graphs. For each node (here labeled A, B, C, D) we construct a *cavity* that holds three orange cars, one per incident edge. Blue and black cells are cars that are already on their target lanes and hence they can only move right (delay). We label the edges arbitrarily with numbers $1, 2, \ldots, m$; these correspond to the lowest m lanes. If edge number ℓ connects nodes u and v, then there is one orange car with label ℓ in cavity u and one orange car with label ℓ in cavity v. These will need to reach lane ℓ. There are two good routes, shown with orange arrows, one that takes the orange car to column x and one that takes the orange car to column y. To reach column x we will need to delay the black car that is blocking the way. One of the cars has to reach column x; hence for each edge $\{u, v\}$ we will need to move the black car in front of cavity u or cavity v. The set of cavities in which we have moved black cars forms a vertex cover; conversely, if we have a vertex cover of size k it is sufficient to move only k black cars. To complete the proof, one has to check that the blue "walls" are sufficiently thick so that any solution that involves moving blue car is strictly worse than a solution that only moves black and orange cars.

6 Hardness of the multi-lane version

To conclude this work, we will briefly look at what happens when we generalize the lane-changing problem from two lanes to multiple lanes. Assume that we have κ lanes, and the cars are labeled with targets $\{1, 2, \ldots, \kappa\}$. The operations are a natural generalization of the two-lane case: we can *delay* car if there is empty slot after it, and we can move an agent sideways if there is empty space in an adjacent lane. We can only move agents sideways towards their target lane, not away from it.

We will now show that minimizing the total cost for this generalization is NP-hard; we only sketch the key ideas of the argument. The proof is by reduction from the minimum vertex cover problem in 3-regular graphs – this special case of the vertex cover problem is known to be NP-hard [5]. Given a 3-regular graph G with n nodes, we construct a multi-lane instance as shown in Figure 1. If and only if there is a vertex cover of size at most k for graph G, we can route the orange cars to their target lanes so that (1) none of the blue cars are moved, (2) exactly k black cars are delayed once. The construction has sufficiently thick "walls" formed by blue cars such that if we try to move blue cars to make space for orange cars, the cost will be higher than the solution obtained by the above strategy for the trivial vertex cover of size $k = n$.

References

1 Maksat Atagoziyev, Klaus W. Schmidt, and Ece G. Schmidt. Lane change scheduling for autonomous vehicles. In *Proc. 14th IFAC Symposium on Control in Transportation Systems (CTS 2016)*, volume 49(3) of *IFAC-PapersOnLine*, pages 61–66. Elsevier, 2016. doi:10.1016/j.ifacol.2016.07.011.

2 Wonshik Chee and Masayoshi Tomizuka. Vehicle lane change maneuver in automated highway systems. Research Report UCB-ITS-PRR-94-22, UC Berkeley, California Partners for Advanced Transportation Technology, 1994.

3 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2–3):207–216, 2005. doi:10.1016/j.tcs.2005.09.013.

4 Li Feng, Li Gao, and Yun-hui Li. Research on information processing of intelligent lane-changing behaviors for unmanned ground vehicles. In *Proc. 9th International Symposium on Computational Intelligence and Design (ISCID 2016)*, volume 2, pages 38–41. IEEE, 2016. doi:10.1109/ISCID.2016.2018.

5 G. H. Fricke, S. T. Hedetniemi, and D. P. Jacobs. Independence and irredundance in k-regular graphs. *Ars Combinatoria*, 49:271–2719, 1998.

6 Cem Hatipoğlu, Ümit Özgüner, and Konur A. Ünyelioğlu. On optimal design of a lane change controller. In *Proc. Intelligent Vehicles '95 Symposium*, pages 436–441. IEEE, 1995. doi:10.1109/IVS.1995.528321.

7 Ding-wei Huang. Lane-changing behavior on highways. *Physical Review E*, 66(2), 2002. doi:10.1103/PhysRevE.66.026124.

8 Wm. Woolsey Johnson and William E. Story. Notes on the "15" puzzle. *American Journal of Mathematics*, 2(4):397–404, 1879. doi:10.2307/2369492.

9 Jorge A. Laval and Carlos F. Daganzo. Lane-changing in traffic streams. *Transportation Research Part B: Methodological*, 40(3):251–264, 2006. doi:10.1016/j.trb.2005.04.003.

10 Kai Nagel. *High-Speed Microsimulations of Traffic Flow*. PhD thesis, University of Cologne, Germany, 1994.

11 Kai Nagel, Dietrich E. Wolf, Peter Wagner, and Patrice Simon. Two-lane traffic rules for cellular automata: A systematic approach. *Physical Review E*, 58(2), 1998. doi:10.1103/PhysRevE.58.1425.

12 José Eugenio Naranjo, Carlos González, Ricardo García, and Teresa de Pedro. Lane-change fuzzy control in autonomous vehicles for the overtaking maneuver. *IEEE Transactions on Intelligent Transportation Systems*, 9(3):438–450, 2008. doi:10.1109/TITS.2008.922880.

13 Daniel Ratner and Manfred K. Warmuth. Finding a shortest solution for the $n \times n$ extension of the 15-puzzle is intractable. In Tom Kehler, editor, *Proc. 5th National Conference on Artificial Intelligence (AAAI 1986)*, pages 168–172. Morgan Kaufmann, 1986.

14 Daniel Ratner and Manfred K. Warmuth. The $(n^2 - 1)$-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10(2):111–138, 1990. doi:10.1016/S0747-7171(08)80001-6.

15 Robin Schubert, Karsten Schulze, and Gerd Wanielik. Situation assessment for automatic lane-change maneuvers. *IEEE Transactions on Intelligent Transportation Systems*, 11(3):607–616, 2010. doi:10.1109/TITS.2010.2049353.

16 F. Visintainer, L. Altomare, A. Toffetti, A. Kovacs, and A. Amditis. Towards manoeuver negotiation: AutoNet2030 project from a car maker perspective. In *Proc. 6th Transport Research Arena (TRA 2016)*, volume 14 of *Transportation Research Procedia*, pages 2237–2244. Elsevier, 2016. doi:10.1016/j.trpro.2016.05.239.

17 Richard M Wilson. Graph puzzles, homotopy, and the alternating group. *Journal of Combinatorial Theory, Series B*, 16(1):86–96, 1974. doi:10.1016/0095-8956(74)90098-7.

18 Tay Wilson and W. Best. Driving strategies in overtaking. *Accident Analysis & Prevention*, 14(3):179–185, 1982. doi:10.1016/0001-4575(82)90026-4.