On the parameterized complexity of a generalized Rush Hour puzzle *

H. Fernau[†] T. Hagerup[‡] N. Nishimura[§] P. Ragde[§] K. Reinhardt[†]

Abstract

We consider the parameterized complexity of a generalized version of the game *Rush Hour*[1], which is a puzzle requiring the player to find a sequence of moves by vehicles to enable a special target vehicle to escape from a grid-shaped game board that may contain obstacles. Although the problem is PSPACE-complete, we demonstrate algorithms that work in polynomial time when either the total number of vehicles or the total number of moves is bounded by a constant. Our contributions are two-fold, entailing the application of ideas of parameterized complexity to games and to motion-planning problems (albeit motion-planning problems of a very constrained nature).

1 Introduction and Definitions

The goal of the Rush Hour puzzle is to remove a target vehicle from a grid by moving it and other vehicles until eventually it reaches an exit on the perimeter of the grid. Each vehicle can move either horizontally or vertically, but cannot change its initial orientation, and a vehicle can be moved from one position to another only when all intervening grid positions are empty. The grid may also contain obstacles that do not move; we use the generic term "car" to denote a vehicle or an obstacle. The original puzzle consists of a 6×6 grid with a single fixed exit and cars of width one and length two or three.

The Rush Hour puzzle is one of many games and puzzles determined to be computationally difficult to solve [Dem01]; such problems include the (n^2-1)-puzzle [RW90] and Atomix [HS01, HEFN01], related to a variant of Rush Hour in which all cars are squares of shape 1×1. The original Rush Hour problem was generalized by Flake and Baum to allow arbitrary grid sizes and placements of the exit and shown to be PSPACE-complete [FB01].

We extend the generalization formulated by Flake and Baum [FB01]. In our generalization, the game board is an infinite plane, cars are arbitrary axes-parallel rectangles, and each vehicle must reach one of a set of goal positions in the plane.

The PSPACE-completeness result of Flake and Baum indicates that there is not much hope of finding a solution to a given Rush Hour instance in polynomial time. However, we show that when the number of cars or the number of moves allowed is small, the problem can be solved in polynomial time. More specifically, we employ the framework of fixed-parameter tractability as developed by Downey and Fellows [DF99], where a problem is *fixed-parameter tractable* (belongs to *FPT*) if it is solvable in time $O(f(k)p(n))$, where n is the instance size, k is the value of the *parameter* (here: the number of cars or the number of moves), f is an arbitrary function and p is a polynomial.

In still greater generality, an instance of the Rush Hour problem can be formalized as a tuple (C, S, p^0, d, Z), where C is a finite set (of *cars*) and the remaining components are functions mapping C to $\mathcal{P}(\mathbb{R}^2)$, $\mathbb{R}^2$, $\mathbb{R}^2$ and $\mathcal{P}(\mathbb{R})$, respectively. For each $c \in C$, $S(c)$ is the *shape* of c, $p^0(c)$ is its *initial position*, $d(c)$ is its *directional vector*, and $Z(c)$ is the set of its *goals*. A car c is an obstacle iff $d(c) = (0,0)$. A car c initially covers the points $p^0(c)+S(c)$ in the plane, and it can be moved to positions of the form $p^0(c)+ud(c)$, for $u \in \mathbb{R}$, where it will cover the points $(p^0(c)+ud(c))+S(c)$. Correspondingly, the *position* of car c (at a certain moment in time) can be described by a number $u(c) \in \mathbb{R}$, and the set of goals available to a car can be described by a set of *goal intervals*.

The object of the game is to move each car to one of its goal positions. A *configuration* is a function $u : C \to \mathbb{R}$, and it is *legal* if $(p^0(c)+u(c)d(c)+S(c)) \cap (p^0(c')+u(c')d(c')+S(c')) = \emptyset$ for all $c, c' \in C$ with $c \neq c'$. A *move* (defined formally in the full paper) is an operation that adds to (or subtracts from) the position of a car a multiple of its directional vector; it is *legal* if the initial and final configurations as well as all intermediate configurations are legal, that is, no other car blocks the passage. A *solution* to an instance is a sequence of legal configurations such that the first consists of the cars in their initial positions, the last consists of the cars in goal positions, and each pair of successive configurations is connected via a legal move.

Most of our work focusses on a special case of the general problem in which all cars are **a**xes-**p**arallel

*This work was initiated while the authors were participants in the Dagstuhl Seminar on Parameterized Complexity in July 2001. The first author is currently with University of Newcastle, Australia.

[†]Universität Tübingen, Germany
[‡]Universität Frankfurt, Germany
[§]University of Waterloo, Canada

[1]The name "Rush Hour" is a trademark of Binary Arts, Inc.

rectangles and the only directional vectors allowed are $(1,0)$, $(0,1)$ and $(0,0)$. An *APR instance* satisfies the additional constraints that for all $c \in C$, $d(c) \in \{(1,0),(0,1),(0,0)\}$ and $S(c)$ is an open rectangle contained in the first quadrant of the plane and with the origin as a corner. In this case, a car c is called *horizontal* if $d(c) \in \{(1,0),(0,0)\}$ and *vertical* if $d(c) \in \{(0,1),(0,0)\}$. Flake and Baum's version is a special case of APR Rush Hour.

2 Parameterization by the Number of Cars

When the number of cars and the number of goal intervals for each vehicle are bounded by a constant k, we can solve the APR Rush Hour problem in polynomial time. The idea behind our algorithm is to restrict the search to a small set of special configurations. If configurations are viewed as nodes in a graph whose edges represent sequences of legal moves, finding a solution entails searching for a path from the initial configuration to a goal configuration. If the graph is guaranteed to be polynomial in size, then so is the cost of the search.

Define the *lane* of a vehicle as the set of all points in $\mathbb{R}^2$ that the vehicle could cover if all other cars were removed. The horizontal (vertical) lane of an obstacle is defined in the same way, but pretending that the obstacle can move horizontally (vertically). When c and c' are horizontal cars, we write $c <_h^* c'$ if the (horizontal) lanes of c and c' overlap and c is (always) to the left of c', and we define $<_h$ as the transitive reduction of $<_h^*$. We form a *horizontal conflict graph*, a directed labeled graph with the vertices C and the edges $<_h$, where the length of an edge (c, c') is $|\pi_1(S(c))|$, the length of the projection of $S(c)$ onto its first coordinate. A *vertical conflict graph* with edges $<_v$ is defined analogously.

We next explain what we mean by a *multi-move* by a vehicle v. Suppose first that v is horizontal. A multi-move by v starts as a usual move by v. As soon as v hits some other horizontal vehicle v', however, v' starts to move in the same direction as v and with the same speed. In general, at all times during the move, there is a set V of moving horizontal vehicles. Whenever some vehicle in V hits a horizontal vehicle, that vehicle is included in V and starts moving in synchrony with the other vehicles in V. A multi-move by a vertical vehicle is defined analogously, with the roles of horizontal and vertical vehicles switched.

A multi-move is *legal* if, during its execution, no point is ever occupied simultaneously by two or more cars. Although multi-moves are not provided as primitives by the definition of the game, it is easy to see that every legal multi-move that involves m vehicles can be implemented through m suitable legal moves by single vehicles, carried out in an appropriate order. For a multi-move towards the left, e.g., an appropriate order can be obtained from a topological sorting of the horizontal conflict graph.

For convenience, we will assume that the board is made "finite" through four special obstacles, sufficiently far away, that form a rectangular frame.

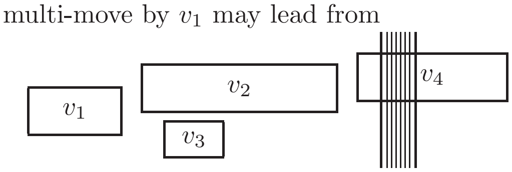
Given a sequence of legal moves in a copy G of the game, we will show how to simulate the sequence through a sequence of multi-moves of a special kind in another copy G' of the game. The simulation will be characterized through the following invariant:

> If a horizontal vehicle is entirely to the left (right) of the lane of a vertical vehicle or the vertical lane of an obstacle in G, the same is true in G'. Correspondingly, if a vertical vehicle is entirely above (below) the lane of a horizontal vehicle or the horizontal lane of an obstacle in G, the same is true in G'. Moreover, if a vehicle is in one of its goal intervals in G, it is in the same goal interval in G'.

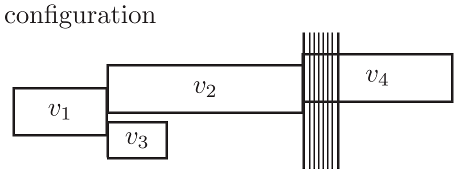
Initially, all cars occupy the same positions in G' as in G, so the invariant holds trivially.

Every move in G by a vehicle v is simulated in G' through a multi-move by v in the same direction that continues for as long as the invariant remains true. Hence the invariant holds at the end of the multi-move by v in G', but an "infinitesimal" additional movement would violate it. Here, for the purpose of stating what it means for the invariant to hold during a move, we assume that the movements by v in G and G' are synchronized in the sense that if a position is occupied by v during the move in both G and G', this happens simultaneously.

Example. If all cars shown are horizontal vehicles, a multi-move by v_1 may lead from



to the configuration



The hatched rectangle symbolizes a vertical lane, which causes the move to stop because v_2 is about to enter the lane.

Lemma 1 *Every multi-move in G' is legal.*

Proof: If not, then at some time during the multi-move, some point in G' is covered by both a horizontal car v and a vertical car v'. This implies that v intersects the (vertical) lane of v', and vice versa. Since, by definition, the invariant holds throughout the multi-move, the same condition is true in G, a contradiction. ■

Lemma 2 *After each move in G and the corresponding multi-move in G', the invariant holds.*

Proof: Suppose without loss of generality that the move in G is by a horizontal vehicle v that moves to the right. The simulating multi-move stops while the invariant still holds, but v might continue to move in G and violate the invariant by leaving the lane of a vertical vehicle or the vertical lane of an obstacle or by reaching a goal position. We demonstrate that this is not the case by showing that in G', v moves at least as far to the right as in G.

The multi-move in G' stops because continuing it would violate the invariant. Because of the synchronization between G and G', this cannot be because of a condition in the invariant that involves v. Therefore the multi-move must stop because some horizontal vehicle v' other than v is about to enter the lane of a vertical vehicle or the vertical lane of an obstacle that is not intersected by v' in G or to leave a goal interval in G' that is occupied by v' in G. The invariant implies in either case that in G, v' is no further to the right than in G'. But since v' is "pushed" directly or indirectly by v and the length of a longest path from v to v' in the horizontal conflict graph is the same in G and in G', we may conclude that even in G, v cannot move further to the right. ■

Assume that k is an upper bound on the number of cars and on the number of goal intervals available to a single car. If a sequence of moves in G leads to a goal configuration, the invariant implies that the corresponding sequence of simulating multi-moves in G', starting from the same configuration, also leads to a goal configuration. Therefore if G has a t-step solution, there is a solution using at most tk steps of the special kind defined by the simulation.

In the simulation, the move by a horizontal vehicle v stops at a position determined solely by some horizontal vehicle v' "pushed" by v (at most k possibilities) and some lane that it is about to enter or goal interval that it is about to leave (at most $(k-1)+k$ possibilities). Hence v stops at one of at most $k(2k-1)$ positions. A vehicle "pushed" by v, including v itself, ends at a position determined solely by the identity of v (at most k possibilities) and by the position at which v ends (by the above, fewer than $2k^2$ possibilities). Hence every such vehicle ends at one of fewer than $2k^3$ positions. Including also the initial position of the car, we see that between moves, every car occupies one of at most $2k^3$ positions. We may conclude that all configurations used by the simulation belong to an easily computable set of at most $(2k^3)^k$ *discretized configurations*.

Lemma 3 *If an APR instance $\mathcal{A}$ has a t-step solution and k upper-bounds both the number of cars and the number of goal intervals available to a car, then $\mathcal{A}$ has a solution of at most kt steps that uses only discretized configurations.*

This result implies the correctness of an algorithm that consists of the formation of a configuration graph restricted to discretized configurations, followed by the search for a path from the initial to a goal configuration. The exact time required by such an algorithm depends on factors such as the representation of the input and the model of computation. If n denotes the input size, however, it is easy to see that any reasonable combination admits a running time of the form $O((2k^3)^{2k}p(n))$, where p is a polynomial (small modifications to the algorithm and a tighter analysis can lower the factor depending on k substantially). This shows that, with the number of cars as the parameter, the APR Rush Hour problem belongs to FPT.

3 Parameterization by the Number of Moves

To handle the case where the number of moves m is bounded by a constant, we compute a set of cars with the property that if all other cars are removed, we are certain that for every $t \le m$ this will not change an instance without a t-step solution into one that has a t-step solution. In other words, the computed set should contain all cars that might be relevant. It may contain additional spurious cars that, in actual fact, do not make any difference, but we bound the number of relevant cars by 3^m. Then we apply an algorithm similar to the one of the previous section, which leads to a running time of the form $2^{O(m^2 \cdot 3^m)}p(n)$.

The difference to the previous section is that to maintain the number of moves, we need a 1-to-1 correspondence between moves in G and their counterparts in G'. Given a legal sequence of moves in G, we find the corresponding sequence in G' by repeating the following procedure until no further changes are possible:

> Take a move to the right (resp. down) and make it as short as possible such that the invariant in Section 2 holds and the rest of the moves is still legal. Take a move to the left (resp. up) and make it as long as possible such that the invariant holds and the move is still legal.

For the sequence in G' thus obtained we can say the following: every position, which is used at any time during the sequence by a vehicle v is

> the initial position of v,

> a goal position of v,

the right (resp. lower) border of a vertical (resp. horizontal) lane or

$\pi_1(S(v'))$ right of (resp. below) a position which was used by a vehicle v' with $v' <_h v$ (resp. $v' <_v v$) at any time during the sequence.

If none of these were the case, then the above procedure would cause this position to be more to the left (resp. higher). Assume that 3^m is also an upper bound on the number of goal positions available to a single car, then the number of possible positions for a car in the first 3 cases is $2{\cdot}3^m$. The fourth case means in other words that v touched one of 3^m possible vehicles v' with $v' <_h v$ (resp. $v' <_v v$) at some time. The position of this vehicle itself is either one of of $2 \cdot 3^m$ possible positions for a vehicle in the first 3 cases leading to $2 \cdot 3^m 3^m$ possibilities or it is caused by touching a third vehicle v'' with $v'' <_h v'$ (resp. $v'' <_v v'$) at some time. Continuing this argument leads to a recursion on the length of a path in the horizontal (resp. vertical) conflict graph, where we can estimate each step in such a "chain of causes" by an additional factor of 3^m. Since we consider only m moves, a "chain of causes" cannot be longer than m. Thus we obtain an upper bound of $2 \cdot 3^m(3^m)^m$ for the number of possible positions for one car in G'. This leads to at most $(2{\cdot}3^m(3^m)^m)^{3^m}$ possible configurations and thus a configuration graph of size $2^{O(m^2 \cdot 3^m)}$ and a running time of the form $2^{O(m^2 \cdot 3^m)}p(n)$, for a polynomial p. Again, the problem under consideration belongs to FPT.

4 Conclusions and Future Work

We investigated the parameterized complexity of Rush Hour, asking whether a given instance of this game is solvable. We first gave a parameterized algorithm for solving APR instances with the number of cars as parameter; then, we showed a parameterized reduction from APR instances with the number of moves as parameter to the first-mentioned problem. In the more general case of cars being arbitrary polygons that move in arbitrary directions (as discussed in the full version of the paper), we were not able to exhibit a fixed-parameter algorithm for the number of cars as parameter. Nonetheless (and this might provide a more general guideline in the research on parameterized algorithms), it was possible to use the reduction technique mentioned above to get a fixed-parameter solution for the problem with the number of moves as parameter.

For future research, we propose the following generalizations and variations on Rush-Hour, bringing the game closer to motion-planning problems that occur in practice, such as those of robots moving in industrial plants. We could require that just one car reaches its goal or that all cars in a subset of vehicles reach their goals. Alternatively, we could let each vehicle move at an individual speed and ask for a solution within a fixed time, or let vehicles move "in parallel".

References

[Dem01] Erik D. Demaine. Playing games with algorithms: Algorithmic combinatorial game theory. In J. Sgall, A. Pultr, and P. Kolman, editors, *Mathematical Foundations of Computer Science (MFCS 2001)*, volume 2136 of *LNCS*, pages 18–32. Springer, 2001.

[DF99] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.

[FB01] Gary W. Flake and Eric B. Baum. Rush Hour is PSPACE-complete, or "Why you should generously tip parking lot attendants". Manuscript. `http://www.neci.nj.nec.com/homepages/flake/rushhour.ps`, 2001.

[HEFN01] F. Hüffner, S. Edelkamp, H. Fernau, and R. Niedermeier. Finding optimal solutions to Atomix. In F. Baader, G. Brewka, and T. Eiter, editors, *KI 2001: Advances in Artificial Intelligence*, volume 2174 of *LNCS/LNAI*, pages 229–243. Springer, 2001.

[HS01] M. Holzer and S. Schwoon. Assembling molecules in Atomix is hard. Technical Report TUM-I0101, Technische Universität München, 2001.

[RW90] D. Ratner and M. K. Warmuth. The (n^2-1)-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10(2):111–137, 1990.