The Computational Complexity of RaceTrack

Markus Holzer[1,⋆] and Pierre McKenzie[2]

[1] Institut für Informatik, Universität Giessen,
Arndtstraße 2, D-35392 Giessen, Germany
holzer@informatik.uni-giessen.de
[2] Département d'I.R.O., Université de Montréal, C.P. 6128,
succ. Centre-Ville, Montréal (Québec), H3C 3J7 Canada
mckenzie@iro.umontreal.ca

Abstract. Martin Gardner in the early 1970's described the game of RaceTrack [M. Gardner, Mathematical games—Sim, Chomp and Race Track: new games for the intellect (and not for Lady Luck), *Scientific American*, 228(1):108–115, Jan. 1973]. Here we study the complexity of deciding whether a RaceTrack player has a winning strategy. We first prove that the complexity of RaceTrack reachability, i.e., whether the finish line can be reached or not, crucially depends on whether the car can touch the edge of the carriageway (racetrack): the non-touching variant is NL-complete while the touching variant is equivalent to the undirected grid graph reachability problem, a problem in L but not known to be L-hard. Then we show that single-player RaceTrack is NL-complete, regardless of whether driving on the track boundary is allowed or not, and that deciding the existence of a winning strategy in Gardner's original two-player game is P-complete. Hence RaceTrack is an example of a game that is interesting to play despite the fact that deciding the existence of a winning strategy is most likely not NP-hard.

1 Introduction

RaceTrack is a popular multi-player simulation pencil-paper game of car racing. The origin of the game is not clear, but most people remember this game from high school, where great effort and time was spent to become the RaceTrack champion, even at the expense of the said champion's school results. Variants of the game appeared all over the world under various names like, e.g., *Le Zip* in France or *Vektorrennen* in Germany. Here is the game description, literally taken from Gardner [4]: The game is played on math-paper where a racetrack is drawn. Then the cars are lined up at a grid position at the start line, and at each turn a player moves his car along the track to a new grid position subject to the following rules:

⋆ Part of the work was done while the author was at the Département d'I.R.O., Université de Montréal, C.P. 6128, succ. Centre-Ville, Montréal (Québec), H3C 3J7 Canada, and the Institut für Informatik, Technische Universität München, Boltzmannstraße 3, D-85748 Garching bei München, Germany.

P. Boldi (Ed.): FUN 2010, LNCS 6099, pp. 260–271, 2010.

1. The new grid point and the straight line segment joining it to the preceding grid point must lie entirely within the track.
2. No two cars may simultaneously occupy the same grid point, i.e., no collisions are allowed.
3. Acceleration and deceleration are simulated as follows: a car maintains its speed in either direction or it can change its speed by only one distance unit per move—see Figure 1 for illustration. The first move following this rule is one unit horizontally or vertically, or both.

The first car to cross the finish line (point) wins. A car that collides with another car or leaves the track is out of the race.

Here we investigate the complexity of RaceTrack when played as a 1-player or 2-player game. Before describing our results, we note that the shape of the racetrack border is *a priori* arbitrary. Thus, in some far-fetched settings, merely verifying whether a move is valid, i.e., merely checking whether the new grid point and the straight line segment joining it to the preceding grid point lies entirely within the track, could be undecidable. To avoid such complications, we stick to a discrete version of the racetrack border, where the track is drawn along grid lines. This is a reasonable restriction, which does not change the practical appeal of the game. Figure 1 shows the discretization of an arbitrarily shaped track.

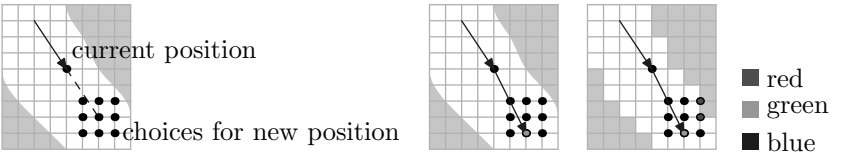

Fig. 1. (Left:) The arrow depicts the last move of the car—two squares east and three squares south—on the racetrack (drawn in white); the gray shaded area is outside of the racetrack. If the car maintains its speed it will follow the dashed line and go two squares east and three squares south again, but it can also reach one extra square north, south, east, or west of this point by changing speed. These extra points are marked by black dots. (Middle:) One out of nine legal moves are shown. (Right:) Discrete version of the RaceTrack game. Observe that certain movements of the car are not possible anymore if the border is *not* allowed for driving; here 7 out of 9 movements remain.

Solitaire RaceTrack will refer to the problem of deciding whether a single player can reach the finish line within an input-specified number of legal moves. By RaceTrack *reachability*, we will mean the simpler problem of deciding whether the single player can reach the finish line at all. In the first part of this paper, we reduce the *touching* variant of RaceTrack reachability (i.e., with the track border considered part of the driving area) to the undirected grid graph reachability problem, and deduce from [1] that this touching variant is NC^1-hard and can be solved in deterministic logarithmic space. By contrast, and to our

initial surprise, we then show that the *non-touching* reachability variant is NL-complete, hence that the complexity of the RaceTrack reachability problem crucially depends on whether the car is allowed to touch the racetrack border or not. Finally, we settle that solitaire RaceTrack is NL-complete, too, regardless of whether driving on the track boundary is allowed or not.

In the second part of the paper, we turn to the 2-player game and prove that checking whether the first player has a winning strategy is P-complete. In particular, the 2-player game is efficiently solvable (in polynomial time). Consider the following "popular conjecture:"

Conjecture 1. All "fun and interesting" (2-player) games are NP-hard.

This "conjecture" attempts to capture when a "game" makes a game and it is wildly accepted in the algorithmic game theory community: in order to be interesting, a game purportedly needs enough complexity to be able to encode interesting (NP-hard) computational problems. And a game in P supposedly becomes boring because a player can quickly learn "the trick" to perfect play. The 2-player RaceTrack game, being fun and interesting to play, yet polynomial time solvable, is a rare example of a game that violates the implication of this conjecture.

2 Preliminaries

We assume familiarity with the basic concepts of complexity theory [7] such as the inclusion chain $\mathsf{AC}^0 \subset \mathsf{NC}^1 \subseteq \mathsf{L} = \mathsf{SL} \subseteq \mathsf{NL} \subseteq \mathsf{AL} = \mathsf{P}$. Here AC^0 and NC^1 refer to the sets of problems accepted by polynomial size uniform families of Boolean {AND, OR, NOT}-circuits having, respectively, unbounded fan-in and constant depth, and, bounded fan-in and logarithmic depth. L is the set of problems accepted by deterministic logarithmic-space bounded Turing machines. SL and NL can be taken to be the sets of problems logspace-reducible to the undirected graph reachability (UGR) and to the directed graph reachability (GR) problems respectively. AL is the set of problems accepted by alternating logspace bounded Turing machines and P is the set of problems accepted by deterministic polynomial time bounded Turing machines. All the relationships depicted in the inclusion chain have been known for a quarter of a century, except for $\mathsf{L} = \mathsf{SL}$, shown by Reingold [8] in 2008.

Another particularly relevant reachability problem is undirected grid graph reachability (UGGR): given an $n \times n$ grid of nodes such that an edge only connects immediate vertical or horizontal neighbors, is there a path from node s to node t, where s and t are designated nodes from the grid? UGGR is NC^1-hard under AC^0 reducibility, it belongs to L, yet it is not known to be L-hard [1]. Finally, we recall the GEN problem (generability problem), known to be P-complete [6] and defined as follows: given a finite set T, a binary operation $\circ$ on T presented as a table, a subset S of T, and an element g in T, determine whether g is contained in the smallest subset of T that contains S and is closed under the $\circ$-operation.

The racetrack in a RaceTrack instance is encoded as a 2-dimensional array $R[i, j]$ indicating whether the grid point (i, j) is on the racetrack, on the racetrack border, on the start line or on the finish line.[1] When the grid point (i, j) happens to be on the racetrack border, 8 further bits of information are required, as follows: Fix a positive distance $\delta < 1$. For each of the non-grid points $(i + \delta, j)$, $(i - \delta, j)$, $(i, j + \delta)$, and $(i, j - \delta)$ we specify whether that point lies within the racetrack or not, and whether that point lies on the racetrack border or not.

3 Complexity of Solitaire RaceTrack

The single-player variant of RaceTrack naturally relates to graph reachability. Here we first analyse the *touching* and the *non-touching* variants of RaceTrack reachability. Observe the following: (i) If a single car moves north, south, east, or west by precisely one square, then the next move of the car can be a complete stop, followed by another move north, south, east or west by precisely one square (this is *stop-and-go* mode). (ii) Since the car starts the game by moving horizontally or vertically by a single square, a car in stop-and-go mode can explore every portion of its track, including its borders, without leaving the game grid.

Theorem 2. RaceTrack *reachability, where the track boundary can be used for driving, is equivalent to UGGR under* AC^0 *reducibility.* □

Proof. The RaceTrack reachability problem reduces to UGGR by the above observations on the stop-and-go car operation mode. Conversely, consider a UGGR instance G. We first construct an equivalent UGGR instance G', to be easily transformed into a RaceTrack instance later. Assume line-column coordinates for the vertices in the UGGR instance. For each vertex (i, j) in G we add to G' four vertices $(2i, 2j)$, $(2i, 2j + 1)$, $(2i + 1, 2j)$, $(2i + 1, 2j + 1)$ and the four edges to form a square. Then a horizontal edge $((i, j), (i, j + 1))$ in G gives rise in G' to the two horizontal edges $((2i + 1, 2j + 1), (2i + 1, 2j + 2))$ and $((2i, 2j + 1), (2i, 2j + 2))$. A vertical edge $((i, j), (i + 1, j))$ in G gives rise in G' to the two vertical edges $((2i+1, 2j), (2i+2, 2j))$ and $((2i+1, 2j+1), (2i+2, 2j+1))$. The start and target vertices in G' are set accordingly. Finally, the RaceTrack instance is built from G' by taking *all* vertices (grid points) in G'.

It remains to identify the grid points sitting on the track border and to provide for those points the extra 8 bits of information required to locate the track relative to the track border. To this end we consider for each grid point its Moore neighborhood consisting of all grid points that are immediate vertical and horizontal neighbors. This forms a 3×3 subgraph of G'. By construction, each such 3×3 subgraph contains at least one square-shaped subgraph arising

[1] An alternative encoding of RaceTrack instances based on polygonal chains of vertices representing the racetrack border is discussed in [3] and [9]. Because the polygonal representation uses binary notation, the number of accessible grid points can be exponential in the input length. Thus, the complexity results for the polygonal encoding may vary significantly from the results presented here.

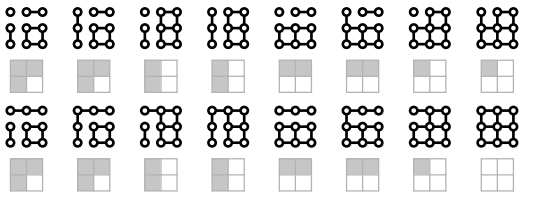

Fig. 2. All possible 3×3 subgraphs, up to mirroring and rotation, that can occur during the construction of G' and their local RaceTrack situations shown underneath

from the four vertices in G' corresponding to each vertex in G. Without loss of generality assume that the square-shaped subgraph resides in the lower right corner of the 3×3 subgraph. Notice that the pair of vertices above this square and the pair of vertices to the left of this square are each connected by an edge since each such pair belongs to a square-shaped subgraph that is adjacent to the 3×3 area. Then we have to consider all further possible edge connections of the nodes that are compatible with the construction of G'. Recall that a vertical (horizontal, respectively) point in G induces *two* vertical (horizontal, respectively) edges in G'. Hence we end up with $2^4 = 16$ possible 3×3 subgraphs that are compatible with the construction of the grid graph G', since we may introduce two independent single edges and two independent double edges. The upshot is that for each of these subgraphs one can easily determine whether the grid point at the center of the sugbraph is on the border or not, and where the non-racetrack part resides. These 3×3 subgraphs and their local RaceTrack situation are shown in Figure 2.

Moreover, the start and finish points in the racetrack are the grid points associated with the start and the target vertex in G' The two reductions are illustrated in Figure 3. □

The UGGR problem and many related problems were studied at length: UGGR is solvable in deterministic logspace by Blum and Kozen [2], which was known long before Reingold [8] showed that general undirected graph reachability (UGR)

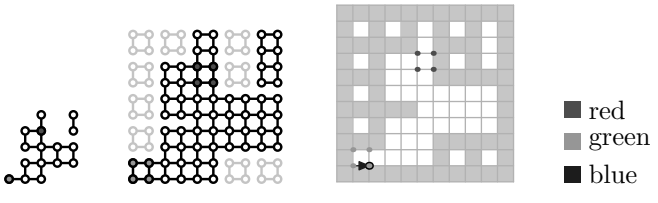

Fig. 3. (Left:) Undirected grid graph with source and target node marked green and red, respectively. Isolated vertices are not shown. (Middle:) Equivalent undirected grid graph were the RaceTrack instance can be easily read off. The square 1×1 subgraphs induced by construction from isolated vertices are drawn in light black. (Right:) RaceTrack instance with start and finish line (point).

is in L. In contrast to UGR, which is L-complete [8], UGGR is only known to be NC^1-hard [1] and thus seems to be of lower complexity, because even general grid-graph reachability (GGR) is not known to be hard for L under AC^0 reductions.

Corollary 3. RaceTrack *reachability, where the track boundary can be used for driving, is* NC^1*-hard under* AC^0 *reductions and belongs to* L. □

So why would disallowing the track borders (that is, causing a fatal crash and ending the game when the car hits the track border) make any difference on the complexity of RaceTrack reachability? The surprising answer is that stop-and-go mode then no longer suffices to handle every situation: the car may now need to balance its horizontal and vertical speeds in order to squeeze its way through narrow track portions. This is illustrated by Figure 4.

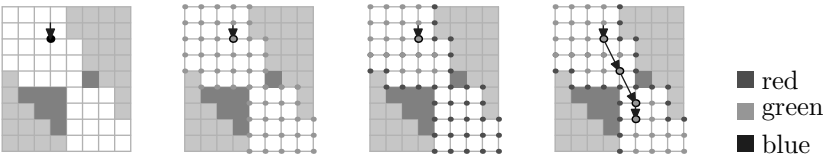

Fig. 4. (Left:) RaceTrack game situation. (Middle-left:) Reachable positions (green) when restricted to stop-and-go-mode, but with drivable track borders. (Middle-right:) Reachable (green) and unreachable positions (red) when restricted to stop-and-go-mode and forbidding driving on the track borders. (Right:) Extending the reachable positions (green) by using speed to traverse the narrow corridor diagonally.

In contrast to Corollary 3, a consequence of the following theorem is that under the usual conjecture that $\mathsf{L} \neq \mathsf{NL}$, the non-touching variant of RaceTrack reachability is provably harder than its touching variant:

Theorem 4. RaceTrack *reachability and solitaire* RaceTrack*, where the track boundary* cannot *be used for driving, are* NL*-complete.*

Proof. We only give the proof for RaceTrack reachability; handling the solitaire RaceTrack time bound adds no significant complication.

To prove that non-touching RaceTrack reachability is in NL, we reduce it to GR. From a game instance, we build a directed graph whose nodes represent all valid grid-position-speed-vector pairs and whose edges represent legal moves. Thus, a vertex $((i,j),\boldsymbol{v})$ is connected to $((i,j)+\boldsymbol{v}+\Delta,\boldsymbol{v}+\Delta)$, where $\Delta \in \{-1,0,1\}^2$, if and only if the line segment starting at (i,j) and ending at $(i,j)+\boldsymbol{v}+\Delta$ is entirely on the racetrack excluding the boundary points. We further add a source vertex s and an edge to the initial grid-position-speed-vector pair, a target vertex t, and edges for every grid-point-speed-vector for every grid-point on the finish line and any speed vector to vertex t. Then the car starting at the initial position can drive to the finish line not touching the boundary points if and only

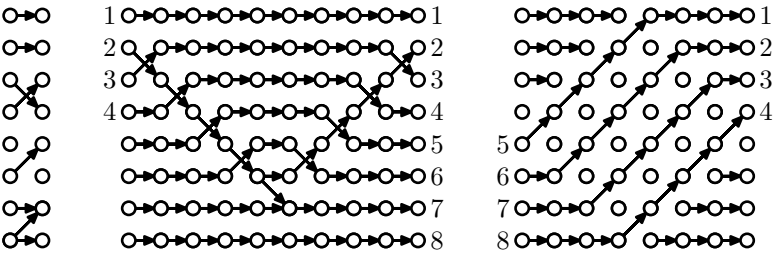

Fig. 5. (Left:) From top to bottom: straight edges, X-crossing edges, up-route edges (with dead ends), and split-join edges. (Middle:) Subgraph induced by an edge $(2, 3)$ of a directed graph G on four vertices. (Right:) Termination subgraph (slightly optimized in length to fit the line) after all edges of G were considered.

if there is a path from s to t in the constructed graph. The construction can be done in logspace since the grid positions and the speed vectors are polynomially bounded in the size of the RaceTrack instance.

To prove that non-touching RaceTrack reachability is NL-hard, we reduce GR to it. Let (G, s, t) be an instance of the GR problem, where $G = (V, E)$ is a directed graph with vertices V and edges $E \subseteq V \times V$, and s and t are the source and the target vertices, respectively. Without loss of generality we may assume that $V = \{1, 2, \ldots, n\}$, that $s = 1$ and $t = n$, and that node n has a self-loop, i.e., edge (n, n) is in E. We will proceed in two stages. In stage I, we reduce testing reachability in G to testing reachability in a layered directed graph (LGR) constructed from four types of edge gadgets, namely *straight* edges, *X-crossing* edges, *up-route* edges, and *split-join* edges. See Figure 5 for a drawing of these four gadget types. The semantics of the X-crossing gadget on Figure 5 is that crossing edges do not touch. In stage II, we will later implement the effect of these connection gadgets in a non-touching RaceTrack game.

STAGE I. The layered graph constructed will consist of a $2n \times O(n^4)$ rectangular grid of nodes whose lines ("rows") are numbered $1, 2 \ldots, 2n$ from top to bottom. This grid is divided up into n identical blocks of size $2n \times m$, where $m \in O(n^3)$. The construction will maintain the property that a path of length k with $k > 0$ exists from node i to node j in G if and only if a path of length at most k exists from node $(i, 1)$ to node $(j, m \cdot k)$ in the rectangular grid.

A block is itself the concatenation from left to right of $O(n^2)$ edge layers, followed by a single termination layer. The edge layers are obtained from left to right by considering every edge in the graph G (in any order). The layer corresponding to edge (i, j) in G is constructed by first using a sequence of X-crossing gadgets to "bend line i downwards" across the lines below it. As the (bent) line i crosses the line $n + j - 1$, a split-join gadget is inserted to create a path from line i to line $n + j$. Using further X-crossing gadgets, line i is then bent back upwards and made to return to its original vertical position. The final (termination) layer in the block uses up-route gadgets to safely create paths from line $n + \ell$ to line ℓ, for

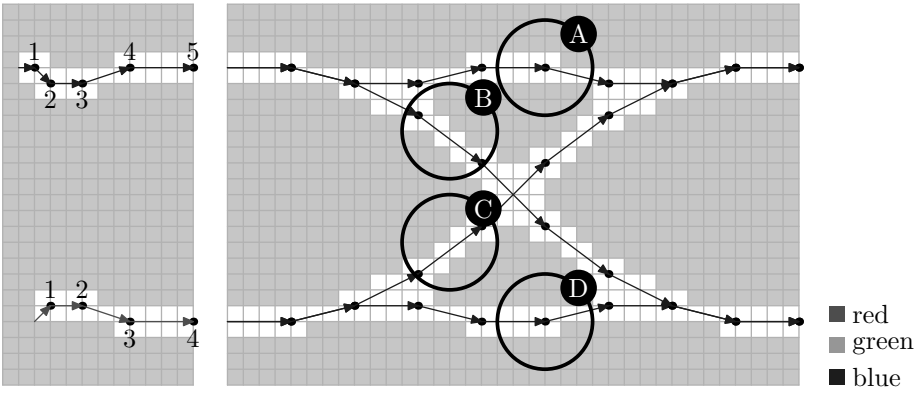

Fig. 6. (Left:) Acceleration tunnels for the 2-player RaceTrack game later used in the proof of Theorem 7—the acceleration tunnel for the 1-player RaceTrack is the top tunnel. (Right:) Blueprint of a 1-player RaceTrack subgame for implementing the four types of gadgets by blocking the racetrack tunnel passages at A, B, C, and/or D, respectively, with obstacles—see the description in the text.

$1 \leq \ell \leq n$. We illustrate the constructions of an edge layer and of a termination layer in Figure 5.

The upshot of concatenating n identical blocks is that node n is reachable from node 1 in G if and only if the rightmost node on line n is reachable from the leftmost node on line 1 in the layered graph. Clearly this construction can be done in logspace.

STAGE II. Here we must simulate the edge gadgets depicted in Figure 5 with RaceTrack (sub)games. The idea is to build a tunnel, of small width to prevent U-turns, which can only be traversed if the car speed (after an appropriate acceleration phase) is within a certain interval. A blueprint of a RaceTrack subgame which can be used to realize all edge gadgets is depicted in Figure 6; the verification that there is no other way through the tunnels is left to the reader. For the actual realization of each edge gadget one has to block some tunnel passages by obstacles appropriately as follows:

- Straight edges: Block the tunnel passage at both B and C.
- X-crossing edges: Block the tunnel passage at both A and D.
- Up-route edges: Block the tunnel passages at A, B, and D.
- Split-join edges: Block only the tunnel passage at C.

Thus one builds in logspace a RaceTrack game for the above constructed graph—the only missing part is an acceleration tunnel that is connected to the game portion induced by the source node of the graph. The construction of an acceleration tunnel is drawn in Figure 6. □

The idea in the proof of Theorem 4 can be made to work for the touching variant of solitaire RaceTrack, proving NL-completeness for that variant also. Indeed

it suffices to slightly modify the gadgets in order to enforce maintaining the driving speed (mostly by shrinking the tunnels as much as possible); deviating from the prescribed speed results in the car not reaching the finish line in time. We must leave the details to the reader. Complementing Theorem 4, we thus have the following:

Theorem 5. *Solitaire* RaceTrack*, where the track boundary can be used for driving, is* NL*-complete.* □

4 Complexity of the Usual RaceTrack Game

In this section we consider the 2-player game. We first show the following (the proof uses the AL characterization of P and is omitted):

Theorem 6. *Deciding if the first* RaceTrack *player has a winning strategy can be done in polynomial time, regardless of whether driving on the track boundary is allowed or not.* □

Next we prove that the non-touching variant is in fact P-complete, thus capturing the precise complexity of the game.

Theorem 7. *Deciding if the first player has a winning strategy in the* 2*-player* RaceTrack *game, when the track boundary* cannot *be used for driving, is* P*-complete.*

Proof. By Theorem 6, it suffices here to show P-hardness. We will reduce the GEN problem to RaceTrack by adapting the reduction from GEN to GAME (two-player game) mentioned by Greenlaw *et al.* in [5, page 208, A.11.1]. In GAME, Blue attempts to prove that an element t is generated by S. Blue does this by exhibiting two elements r and s, also claimed to be generated by S, such that $t = r \circ s$, while Red attempts to exhibit an element of the pair that is not generated by S. The behaviours of Blue and Red will be simulated by (drum roll!) a RaceTrack game.

Let $T = \{1, 2, \ldots, n\}$, $S \subseteq T$, and $g \in T$ be the GEN goal. Notice that g is generated by S if and only if g would appear in a set X initialized to S after at most n iterations of the informal operation $X \leftarrow X \cup (X \circ X)$. The racetrack will be a $(n + n^2 + 2) \times O(n^5)$ rectangular grid of nodes whose lines ("rows") are named $w, 1, 2 \ldots, n, (1,1), (1,2), \ldots, (n,n), c$ from top to bottom, where w is mnemonic for the "$w$inning" (blue) line and c is mnemonic for the "$c$ontinuing" (red) line.

At the meta-level, imagine the virtual edges $(t, (r,s))$, $((r,s), r)$ and $((r,s), s)$, for $r, s, t \in T$ such that $t = r \circ s$. Initially, Blue's goal is to prove that $t = g$ is generated (suppose that $g \notin S$). The racetrack will consist of n identical blocks. Each block will implement the following. Blue will be forced to traverse one of the $O(n^2)$ virtual $(t, (r,s))$ edges. This will pin Blue to a choice of a pair (r,s) such that $t = r \circ s$. Red will come along and be forced to traverse either the virtual $((r,s), r)$ edge or the virtual $((r,s), s)$ edge. This will pin Red to a choice

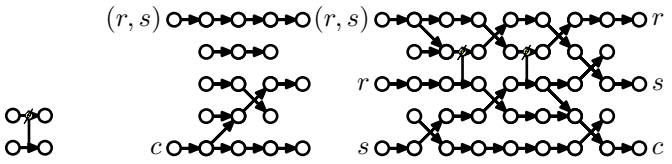

Fig. 7. (Left:) Killer edge gadget or killer edges, for short—Blue is driving at the top and Red at the bottom. (Middle and Right:) Parting gadget with selector gadget on the left. Again Blue drives on the top and Red on the bottom.

of a challenge $\sigma \in \{r, s\}$. At this point, a gadget will force Blue to meet the challenge, that is, to proceed on to the next block with $t = \sigma$.[2]

We now give the details. The behaviour of Blue choosing a virtual $(t, (r, s))$ edge can already be simulated with the edge gadgets depicted in the middle of Figure 5, now viewing these gadgets as operating on virtual edges $(t, (r, s))$. To implement Red's behaviour, assume temporarily that we have at our disposal a RaceTrack subgame implementing the following *killer* gadget: the gadget has two entry points and two exit points, and whenever both cars enter, only Red makes it through while Blue experiences the inconvenience of a fatal accident; when a *single* car of either colour enters, that car makes it through safely. With the help of such a killer gadget, we can construct a controlled *parting* gadget with three input tunnels named (r, s), r, and s and with three output tunnels named r, s, and c satisfying the following property:

– If Blue enters at (r, s) and Red enters at r (s, respectively), then Blue can only exit at r (s, respectively) and Red exits at c.

See Figure 7 for the killer gadget icon and for the parting gadget. Assuming proper operation of the killer gadget, the correct operation of the parting gadget should be clear.

But the parting gadget is exactly the device required to implement Red's challenge of an element of the pair (r, s), in response to Blue's choice of $(t, (r, s))$, followed by the handing over of that challenge to Blue at the next move! In other words, the parting gadget implements Red's meta-level choice of a virtual edge $((r, s), r)$ or $((r, s), s)$: Blue can only proceed towards the line (r or s) selected by Red.

Constructing a block in the reduction from GEN to RaceTrack therefore consists, first, in aligning the devices for the virtual edges $(t, (r, s))$ in sequence, describing possible driveways for the blue car, and additionally installing a parallel street c for the red car. Next, for all pairs (r, s), the devices for the virtual edges $((r, s), r)$ and $((r, s), s)$ are aligned in sequence and the outputs r and s are redirected to their respective initial lines. Finally, for all elements $t \in S$, a branch-off to the line w is installed.

[2] Technically, the gadget will connect Blue to the σ line and will connect Red to the c line; this will be followed by an exit gadget connecting every ℓ line with $\ell \in S$ to the w line, thus allowing Blue to speed off to a win when $\sigma \in S$.

Once the n identical blocks are assembled, final details involve taking care of the trivial case in which g happens to belong to S from the start (this requires inserting an exit gadget before the first block), connecting appropriate acceleration tunnels for both players and joining the c line to the finish line (so that Red can make it to the finish line one step ahead of Blue if Blue after n steps has not proven his point that the initial GEN goal is generated).

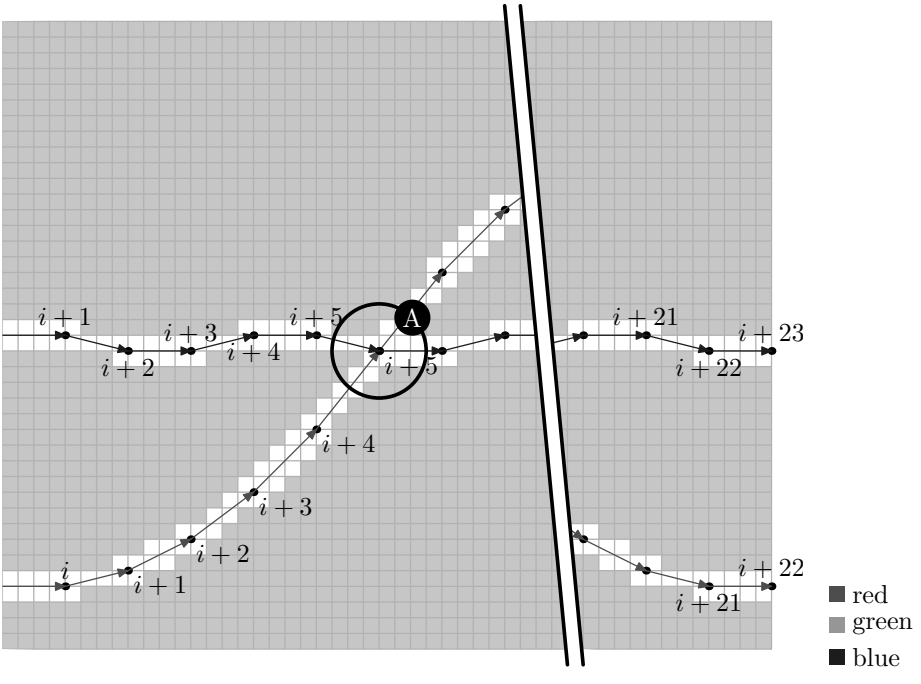

Fig. 8. RaceTrack subgame for the killer gadget—the blue car (player 1) is blocked by the red car (player 2) in the $(i + 5)$th turn at A. In the middle part of the gadget, which is not shown, the tunnel for the blue car continues snake like and the tunnel for the red car bends in an S-like fashion first slightly up, then down intersecting with the snake like tunnel of blue, and finally towards the exit at the bottom of the subgame.

It remains to describe the implementation of the killer gadgets. We make use of tunnels of small width and appeal to the RaceTrack rule that no two cars may simultaneously occupy the same grid point. Crossing tunnels are of course required. Note that we have used crossing tunnels before, namely when implementing gadgets requiring the RaceTrack blueprint shown on Figure 6. Collisions were not an issue when the blueprint was used in the single-player context, but here we need the same blueprint to implement the crossings implicit in Blue's behavior, as explained above: Figure 6 shows that for such a purpose, collisions can be avoided by construction. A killer gadget, on the other hand, requires a crossing tunnel that allows collisions, as shown in Figure 8. To obtain

the desired killer gadget effect, the red car must enter the gadget one step ahead of the blue car: this property can be maintained throughout the racetrack by fine-tuning the acceleration tunnels at the beginning of the game—see the drawing in Figure 6. This completes the description of the (logspace-computable) reduction from GEN to non-touching RaceTrack. □

5 Conclusion

We investigated the complexity of RaceTrack and obtained precise complexity characterizations in terms of completeness results for solitaire RaceTrack and for (the non-touching variant of) 2-player RaceTrack. As to (1-player) RaceTrack reachability, it turned out that having or not having access to the boundary of the racetrack for driving affects the complexity of the problem. More generally, we observed that 2-player RaceTrack is a polynomial time solvable game (hence unlikely to be NP-complete), yet interesting and fun to play.

We have not settled the P-hardness of the touching variant of 2-player RaceTrack. We leave the latter question, as well as the study of the wealth of game variants introduced by the gaming community over the years, as open problems for further research.

Acknowledgements. We thank Martin Beaudry and Martin Kutrib for useful discussions, and the anonymous referees for raising interesting issues, such as the need to distinguish solitaire RaceTrack from RaceTrack reachability.

References

1. Allender, E., Mix Barrington, D.A., Chakraborty, T., Datta, S., Roy, S.: Planar and grid graph reachability problems. Theory of Computing Systems 45(4), 675–723 (2009)
2. Blum, M., Kozen, D.: On the power of the compass (or, why mazes are easier to search than graphs). In: Proceedings of the 19th Symposium on Foundations of Computer Science, Ann Arbor, Michigan, USA, pp. 132–142. IEEE Computer Society, Los Alamitos (1978)
3. Erickson, J.: How hard is optimal racing? (2009), `http://3dpancakes.typepad.com/ernie/2009/06/how-hard-is-optimal-racing.html`
4. Gardner, M.: Mathematical games—Sim, Chomp and Race Track: new games for the intellect (and not for Lady Luck). Scientific American 228(1), 108–115 (1973)
5. Greenlaw, R., Hoover, H.J., Ruzzo, W.L.: Limits to Parallel Computation: P-Completeness Theory. Oxford University Press, Oxford (1995)
6. Jones, N.D., Laaser, W.T.: Complete problems for deterministic polynomial time. Theoretical Computer Science 3, 105–117 (1977)
7. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley, Reading (1994)
8. Reingold, O.: Undirected connectivity in log-space. Journal of the ACM Article 17, 55(4), 24 (2008)
9. Schmid, J.: VectorRace (2005), `http://schmid.dk/articles/vectorRace.pdf`