The Parameterized Complexity of Ricochet Robots

ADAM HESTERBERG¹,a) JUSTIN KOPINSKY¹

Received: November 7, 2016, Accepted: May 16, 2017

Abstract: Sliding maze puzzles like Ricochet Robots and Atomix are puzzles in which solvers must maneuver agents around a grid board subject to the constraint that whenever an agent moves in some direction, it must move as far as possible in that direction. In general, finding an optimal solution to these puzzles is known to be PSPACE-complete. This paper further shows that these puzzles are W[1]-hard with respect to the number of robots in the puzzle instance (and therefore unlikely to be Fixed Parameter Tractable).

Keywords: Computational Complexity, Parameterized Complexity

1. Introduction

Several recent puzzle games require the solver to maneuver sliding agents around a grid board filled with obstacles. In particular, a sliding agent can freely choose a (cardinal) direction to travel, but must move as far as possible in that direction until an obstacle or another agent is encountered. The most famous examples of these types of puzzles are Atomix (a playable version is available at Ref. [2]) and Ricochet Robots [10]. Atomix requires the player to assemble all agents into a specified formation (though that formation can be centered anywhere on the board) while Ricochet Robots asks for a single specified agent to be moved to a specified grid square. The puzzles are also contrasted by their obstacles: Atomix has ‘thick’ walls (impassable grid squares) while Ricochet Robots has thin walls (impassable edges).

For instance, Fig. 1 shows a Ricochet Robots board in which the yellow robot must reach the square with a yellow token in the upper left. One fastest solution (with solution length 5) is shown, in which the yellow robot moves up, then the blue robot moves down and then right to serve as a blocker, then the yellow robot moves back down, stopping when it hits the newly moved blue robot, and left into the target. There is one other equally good solution on this board, finding which is left as an optional exercise to the reader.

Early work shows the feasibility version of Atomix to be PSPACE-complete [7] by reduction from the non-empty intersection problem for finite automata. The proof essentially works for Ricochet Robots as well [5]. One of the proof techniques in Ref. [5] can be simply extended to be gap-producing, so the gap-problem versions of these puzzles are also at least NP-hard. Unsurprisingly, these proofs require a large number of agents taking a large number of moves to work, so it is natural to ask whether efficient algorithms exist if the number of agents or number of moves is small.

Fortunately, the field of parameterized complexity gives us a model in which we can answer such a question. Parameterized complexity theory is motivated by the observation that many NP- or PSPACE-hard problems are hard only for instances which have some particularly unpleasant attribute.

For example, consider the problem of Boolean Formula Satisfiability: given a boolean formula, does there exist an assignment of truth values to variables which causes the formula to evaluate to true? In the general case, this problem is well known to be NP-hard [8], but consider what happens when we restrict our attention to formulae with only a small number, say k, distinct variables. In this case, there is a simple algorithm which runs in time O(2^kn) (where n is the size of the formula) by checking the truth value of the formula for every possible assignment. This suggests that we might consider Boolean Formula Satisfiability ‘easy’ when parameterized by the number of variables.

In general, we will consider a parameterized problem of size n and parameter k to admit an efficient solution if there is an algorithm solving it in time f(k)O(n^c) for some computable function.
f and constant c. Such problems are known as “Fixed Parameter Tractable” (FPT) [4], defined more carefully in Section 2.

By contrast, there are many problems which are believed not to be FPT, including important graph theoretic problems like k-Vertex Cover (given a graph G, does there exist a clique of size k?) and k-Dominating Set (given a graph G, does there exist a set, S, of vertices of cardinality at most k such that every vertex in G is adjacent to some element of S?), as well as circuit complexity questions such as k-Weighted Boolean Formula Satisfiability (given a boolean formula, does there exist an assignment of variables at most k of which are assigned true which satisfies the formula?). These problems belong to a hierarchy of classes defined in Section 2. The situation is analogous to that of the polynomial hierarchy: it is unknown whether any classes in the W-hierarchy have FPT algorithms, but researchers generally believe that they do not.

Prior work has considered the parameterized complexity of various games and puzzles [6], [9], [11], [12]. This paper addresses the question of whether parameterized versions of Ricochet Robots and Atomix admit FPT algorithms as follows. Ricochet Robots and Atomix are:

1. FPT when parameterized by the sum of the number of agents and solution length (Section 3).
2. Hard for W[SAT], the highest class in the W-hierarchy (defined in Section 2), when parameterized by the number of agents, and
3. In W[1], the lowest class in the W-hierarchy, when parameterized by the length of solution (but unresolved as to whether they are in FPT or hard for W[1]).

2. Parameterized Complexity

In this section, we will give a brief introduction to the relevant concepts within Parameterized Complexity. A more complete treatment of the topic can be found in Downey and Fellows [4], from which all definitions in this section are drawn.

Definition 1. The class FPT contains all parameterized languages (that is, pairs of a language and a parameter k) which can be decided by an algorithm which runs in \( f(k)O(|x|^c) \) time, where \( f \) is any computable function and \( c \) is any constant.

FPT generally represents problems which can be solved efficiently when the parameter is small, even if they can’t be solved efficiently in general. For example, consider the problem of k-Vertex Cover, which asks whether a given graph G has a vertex cover of size k, where a vertex cover of G is a set, \( S \), of vertices such that for every edge \( e = (v_1, v_2) \) either \( v_1 \in S \) or \( v_2 \in S \), possibly both. Although (unparameterized) Vertex Cover is known to be NP-hard [8], there is a simple algorithm solving k-Vertex Cover in time \( O(2^k|G|) \) (we leave this as an exercise to the reader; alternatively, the algorithm can be found in Section 3.2 of Ref. [4]). This characterization demonstrates how FPT is useful in describing when a problem which, in the general case, requires a large amount of resources to solve may admit efficient solutions in restricted settings.

On the other end of the spectrum, we have the parameterized class XP, defined as follows:

Definition 2. The class XP contains all parameterized languages which can be decided by an algorithm which runs in time \( O(|x|^{f(k)}) \) where \( f \) is any computable function.

XP is known to be intractable (that is, XP contains problems which are known not to be members of FPT) and thus serves mostly as an upper bound.

There is also a parameterized analog to classical reductions, called FPT reductions:

Definition 3. A language \( L \) is FPT reducible to another language, \( L' \) if there exists functions \( f, g : \mathbb{N} \rightarrow \mathbb{N} \) such that there is a mapping \( \langle x, k \rangle \mapsto \langle x', k' \rangle \) which is computable in time \( f(k)x^{g(k)} \) and which satisfies \( s(x, k) \in L \iff \langle x', k' \rangle \in L' \) and \( k' \leq g(k) \).

Note that FPT reductions look very similar to Karp style reductions, except that the reduction may transpose the parameter and can run in FPT time (rather than classical polynomial time). An important property of FPT reductions which is reflected in the above definition is that they are parameter preserving—that is, \( k \) has no dependence on \( |x| \), only on \( k \). Without this property, the reduction would be invalid because an FPT solution to \( L' \) running in time \( f(k')x^{g(k')} \) would only correspond to a time \( f(|x|, k)x^{g(|x|)} \) time solution to \( L \), which is not FPT.

The notion of parameterized reductions gives us the tools to introduce a set of classes intermediate between FPT and XP, known as the W[t] hierarchy. Fundamental to the W[t] hierarchy is a core family of problems called k-Weighted Weft-t Depth-h Circuit Satisfiability defined for any fixed \( t, h \) and parameterized by \( k \). In order to define these classes, we must briefly discuss some circuit terminology.

The depth of a circuit is the maximum number of gates encountered on any path from an input to the output. The fan-in of a logical AND- or OR- gate is the number of inputs to that gate. Similarly, the fan-out of a logical gate is the number of other gates which take the output of the gate in question as inputs. In general circuits, gates may have unbounded fan-in or fan-out. In this case, the weft of a circuit is the maximum number of gates with fan-in 3 or more encountered on any path from an input to the output. We can always convert a gate with fan-in 2 to p − 1 separate gates, each of fan-in 2, but this comes at the cost of increasing the depth by at least \( \log p \) (achieved by arranging the replacement gates in a binary tree). Thus, if we require our circuits to have depth \( O(1) \), then any gate with fan-in \( \omega(1) \) cannot be replaced in this way.

Definition 4. For fixed \( t, h \), the k-Weighted Weft-t Depth-h Circuit Satisfiability problem asks if, given an input logical boolean circuit \( C \) and parameter \( k \) such that \( C \) has depth at most \( h \) and weft at most \( t \), can \( C \) be satisfied by an assignment of inputs, exactly \( k \) of which are set to true?

For example, consider the k-Independent Set problem of graph theory, in which, given a graph, \( G = (V, E) \), we must produce a set of \( k \) vertices, no two of which are adjacent. Figure 2 gives a boolean formula and corresponding circuit representation which represent this problem. Note that the circuit has depth 2 and weft 1, but we cannot reduce the weft while keeping the depth constant.

We can now define the W[t] hierarchy directly:

Definition 5. For fixed \( t \), the class W[t] consists of languages which are FPT reducible to k-Weighted Weft-t Depth-h Circuit Satisfiability for some constant \( h \).

© 2017 Information Processing Society of Japan
we can always try every one of the Weighted Boolean Formula Satisfiability constrained to formulae Monotone $k$-Independent Set—the parameterized analog in which the parameter represents the size of the Independent Set to be found—is certainly in $W[1]$.

Figure 3 gives a diagrammatic representation of the classes discussed in this section.

3. Basic Results: FPT and XP

This section of the paper proves that Ricochet Robots (with a target space and thin walls) and Atomix (with a target configuration and thick obstacles) are in FPT with respect to the number of robots and solution length and in XP with respect to the number of robots. Rather than prove these separately for both puzzles, we define a puzzle $RR$, with a target configuration and thin walls, prove that it’s harder than both, and then prove that $RR$ is nevertheless in FPT and XP for the appropriate parameters.

Definition 6. $RR$ is a class of 1-player games. A board state for $RR$ consists of:

(1) A board size $n$,
(2) A goal configuration $G \subseteq \{(i, j) : 1 \leq i, j \leq n\}$,
(3) A set $R$ of robot positions, $R \subseteq \{(i, j) : 1 \leq i, j \leq n\}$, and
(4) A subset $W$ ("walls") of $\{(i, j), (i', j') : |i - i'| + |j - j'| = 1\}$.

A legal move for the player consists of choosing a robot at position $(i, j)$ and a direction $d \in \{(0, 1), (0, -1), (1, 0), (-1, 0)\}$ and updates the board state by deleting the robot at position $(i, j)$ and adding it at $(i, j) + dt$, where $t$ is the smallest value such that $(i, j) + dt + d$ is in $R$ or $(i, j) + dt, (i, j) + dt + d$ is in $W$.

A board state is solved if and only if there exists $t \in \{(i, j) : 1 \leq i, j \leq n\}$ ("translation of the goal configuration") such that $|g + t : g \in G| \subseteq R$.

A board state is solvable in at most $t$ moves ($t \geq 0$) if and only if either it’s solved or $t \geq 1$ and there’s a legal move to a board state that’s solvable in at most $t - 1$ moves.

In human terms, $RR$ is Ricochet Robots (as described in the introduction) with a target robot configuration instead of a target space, or ATOMIX (as described in the introduction) with thin-wall obstacles.

The definition presented above does not distinguish between robots, as in the any-color mode of Ricochet Robots. The definition can be modified to distinguish between robots as in the specified-color mode of Ricochet Robots: instead of a set $R$ of robot positions, use a function from the set $S$ of robots to a set $R$ of robot positions, and instead of a goal configuration, use a function from a subset of robots to a set of positions on the board. All the results and proofs we present are valid for this version of the game as well, but neither game is reducible to the other.

$RR$ is polynomial-time reducible from ATOMIX by inclusion (every ATOMIX problem is also an $RR$ problem) and from Ricochet Robots by adding a robot surrounded by obstacles at distance more than $n$ from the rest of the board and having the target configuration consist of two robots, separated by the same vector as the trapped robot is from the former target square, so the target configuration in $RR$ is reachable if and only if the target square in Ricochet Robots is.

Theorem 7. $RR$ is fixed-parameter tractable with respect to the
parameters of the number \( k \) of robots and the length \( \ell \) of the solution.

Proof. We’ll simply try all possibilities. The tree of possible moves has branching factor at most \( 4k \), since each of the \( k \) robots has at most 4 legal moves, and depth \( \ell \), the maximum permitted solution length, so the game tree consists of at most \( k^{\ell} \) board states, and calculating each one from the previous one takes time at most \( O(n) \). Determining whether a board state is a winning position takes time at most \( n^2 \), since there are at most \( n^2 \) possible positions for the target configuration on the board and evaluating whether a given configuration exists in a given position takes time at most \( n^2 \), so the game tree can be completely explored in time at most \( O(k^{\ell}n^2) \), making it FPT, as desired. □

Theorem 8. RR\(_{\ell}\) is in XP with respect to the number \( k \) of robots.

Proof. There are \( \binom{n}{k} = O(n^k) \) arrangements of \( k \) robots and hence \( O(n^k) \) possible board states. There are at most \( 4k \) possible successor board states of a given board state (at most 4 moves for each of the \( k \) robots), and finding each one takes time \( O(n) \), so calculating the entire directed graph of possible board states takes time \( O(kn^k) \). Also, determining whether a board state is a winning position takes time \( O(n^k) \), as above, so determining the directed graph of board states labelled by whether they’re winning positions takes time \( O(kn^k) \). Finally, graph search takes linear time, so we can do graph search to determine whether any winning position is reachable from the starting board state in time \( O(kn^k) \), making \( RR\(_{\ell}\) in XP, as desired. □


In this section, we show that Ricochet Robots (and hence also \( RR\(_{\ell}\) as defined in Section 5) is in W[1] with respect to solution length \(^1\). Consider the following parameterized problem regarding generic nondeterministic Turing Machines:

Definition 9. SnortTM\((M, k)\): Given a nondeterministic Turing Machine \( M \) and a parameter \( k \), does \( M \) have a computation path which accepts the empty string in at most \( k \) steps?

SnortTM\((M, x, k)\) is known to be W[1] complete with respect to \( k \) \(^4\). This result provides us a convenient framework to show Ricochet Robots \( \in \) W[1] with respect to the solution length, \( \ell \); in particular, we will provide a reduction from instances of Ricochet Robots with parameter \( k \) to instances of SnortTM with parameter \( poly(k) \). In practice, this constitutes a nondeterministic algorithm which runs in time \( f(\ell) \) (in particular, the runtime is independent of the size of the Ricochet Robots instance, although the size of the algorithm may depend on it) and accepts if the instance has a solution of length at most \( \ell \) and rejects otherwise.

Because the SnortTM instance is constructed by our reduction with knowledge of the instance to be solved, our algorithm is allowed \( O(1) \) access to any data in the input (achieved by ‘hard coding’ the instance into the states of \( M \)). In particular, we can encode a constant time lookup table to retrieve the location \(^2\) of

\(^1\) We don’t prove that \( RR\(_{\ell}\) or ATOMIX is in W[1] because even checking that a set of robots is in a target configuration takes time proportional to the number of robots, not the number of moves.

\(^2\) Note that the coordinates of the first obstacle may take space (and hence time) dependent on the size \( n \) of the instance to specify, but the alphabet size and number of states of the Turing Machine are allowed to depend on \( n \), so we can store locations.

the first obstacle (wall or robot) encountered when moving out of a given location represented by symbol \( x \) in a given direction \( d \) in the initial set up of the instance. We will call that obstacle which we will call \( obs(x, d) \). We can also encode another constant time lookup table to tell, for any two squares represented by symbols \( x \) and \( y \), which direction \( y \) is from \( x \) (or ‘none’ if \( x \) and \( y \) are not in the same row or same column).

The algorithm is as follows: if there’s not already a robot at the target, guess \( k \) moves, and check that one of them moves into the target location. Each nondeterministic guess consists of a robot and a space for it to move to. To check that a move is legal, we check that:

1. The space \( y \) which the robot is attempting to move to is in a legal direction \( d \) from its last position \( x \) (the result of its most recent move, or its starting position if there was no such move).

2. The space is a legal stopping place for the robot, either because there’s a wall stopping it, or because there was initially a robot there to stop it and that robot has not yet moved, or because some previous move put a robot in position to stop it and no subsequent move moved that robot.

3. There are no obstacles in the way. We do this in several sub-phases:

i. We first check, for each previous move, that either that move didn’t put a robot in the way or, if it did, that some later move moved the same robot.

ii. We next check that any robots initially in the way have moved: if \( obs(x, d) \) is a robot and that robot is in position \( z \) between \( x \) and \( y \), we check that that robot has moved, and if so restart step 3 from \( z \), but reject if it hasn’t moved; we restart the search at most \( k \) times because at most \( k \) robots can have been moved out of the way. If \( z \) is past \( y \) (coming from \( x \)) then the verification step is complete.

iii. Finally, if \( obs(x, d) \) is a wall, we reject if the wall is between \( x \) and \( y \), otherwise the verification step is complete.

Step 1 takes time \( O(1) \). Step 2 takes time \( O(k^2) \) (the worst case is that \( \Omega(k) \) previous moves moved a robot in the way and we have to spend \( \Omega(k) \) time checking that that robot subsequently moved out of the way). Finally, any iteration of step 3 takes time \( O(k) \) but we may have to run \( \Omega(k) \) iterations, so in total step 3 takes time \( O(k^2) \). We run steps 1–3 a total of \( O(k) \) times to verify \( k \) moves so our algorithm takes time \( O(k^3) = poly(k) \), which suffices for a reduction.

5. Parameterized by Number of Robots: W[SAT]-hard

This section of the paper proves that Ricochet Robots (with a target space and thin walls) and Atomix (with a target configuration and thick obstacles) are W[SAT]-hard. Rather than prove these separately for both puzzles, we define a puzzle \( RR\(_{\ell}\) (with a target space and thick walls), prove that it’s easier than both, and then prove that \( RR\(_{\ell}\) is nevertheless hard.

Definition 10. \( RR\(_{\ell}\) is a class of 1-player games. A board state for \( RR\(_{\ell}\) consists of:
A board size $n$.

A target space $G \in \{(i, j) : 1 \leq i, j \leq n\}$.

A set of robot positions, $R \subseteq \{(i, j) : 1 \leq i, j \leq n\}$, and

A subset $W$ ("walls") of $\{(i, j) : 1 \leq i, j \leq n\}$.

A legal move for the player consists of choosing a robot in position $(i, j)$ and a direction $d \in \{(0, 1), (1, 0), (-1, 0), (0, -1)\}$ and updates the board state by deleting the robot at position $(i, j)$ and adding it at $(i, j) + dt$, where $t$ is the smallest value such that $(i, j) + dt + d$ is in $R \cup W$.

A board state is solved if and only if $G \in R$ (i.e., some robot has reached the target space).

A board state is solvable in at most $t$ moves ($t \geq 0$) if either it's solved or $t \geq 1$ and there's a legal move to a board state that's solvable in at most $t - 1$ moves.

In human terms, $RR$ is Ricochet Robots with impassible squares instead of thin walls or Atomix with a target square instead of a target configuration.

The definition presented above does not distinguish between robots, as in the any-color mode of Ricochet Robots. The definition can be modified to distinguish between robots as in the specified-color mode of Ricochet Robots: instead of a set $R$ of robot positions, use a function from the set $S$ of robots to a set $R$ of robot positions, and instead of a goal configuration, use a function from a subset of robots to a subset of positions on the board.

All results and proofs we present are valid for this version of the game as well, but neither game is reducible to the other.

$RR$ is polynomial-time reducible to Ricochet Robots by inclusion (every $RR$ problem is also a Ricochet Robots problem) and to Atomix by adding a robot surrounded by obstacles at distance more than $n$ from the rest of the board and having the target configuration consist of two robots, separated by the same vector as the trapped robot is from the former target square, so the target configuration is reachable if and only if the target square is.

So, to prove that either of Ricochet Robots and Atomix is hard, we only need to prove that $RR$ is hard.

**Theorem 11.** $RR$ is $\text{W[ SAT]}$-hard.

To prove this theorem, it is sufficient to show that, given any monotone boolean formula $F$ of size $n$ and a parameter $k$, we can construct an instance of $RR$, of size $\text{poly}(n)$ and $|R| = k + 1$ which is solvable if and only if $C$ has a satisfying assignment of variables with at most $k$ variables set to 1.

At a high level, our construction will build an instance of $RR$, which 'simulates' a computation of the circuit representation of the given boolean formula. There will be $k + 1$ robots in the construction, $k$ of which will represent the $k$ variables chosen to be true, and the remaining robot being the goal robot. Sections of the $RR$ instance will logically correspond to gates in the input circuit and, as we will show, the goal robot will only be able to traverse a gate (i.e., get from the entrance to the exit of the region corresponding to the gate) if the gate output is true for the chosen assignment of variables. Thus, the target space will be the exit of the region corresponding to the output gate, which the robot will only be able to reach if the output of the output gate is true.

We first show some basic gadgets for $RR$, which are fundamental to our construction. Figure 4 shows turn, crossover, one-way, and choice gadgets and their corresponding wire diagram depictions. A turn gadget allows wires or robot paths to turn; a one-way gadget is traversable by robots in one direction but not the other; a crossover gadget lets wires cross without intersecting; and a choice gadget lets an entering robot choose either path to exit. We note that in all diagrams, four way intersections will always signify crossover gadgets while three way intersections signify choice, and diagrams of wires as on the left of Fig. 4 are implemented by robot paths as on the right.

One slightly less straightforward gadget is the lock gadget, shown in Fig. 5. The lock gadget has two input robots, a door robot and a key robot. The door robot can pass through the gadget if and only if the key robot is present. The key robot can enter and exit the gadget freely.

Given these gadgets, we can proceed with the main construction. We will start from a circuit $C$ representing an arbitrary monotone boolean formula with input variables $x_1, \ldots, x_n$ combined by AND and OR gates. The main idea is to have $k$ variable robots which will be forced to commit to a choice of $k$ distinct variables to set to 1. Because $C$ is positive, it cannot help to set fewer than $k$ variables. Additionally, there is a goal robot which is trying to reach the target space and can do so if and only if $C$ is satisfiable. The goal robot will traverse some set of satisfied gates which are sufficient to satisfy the entire circuit. We will enforce that once the goal robot enters a gate, it can leave if and only if that gate is satisfied.

Figure 6 shows a gadget which forces $k$ robots $R_1, \ldots, R_k$ to choose $k$ distinct $x_i$. Logically, if robot $R_j$ enters the box labelled $x_j$, then it has chosen variable $x_j$. All boxes $x_j$ have paths converging to a single region which logically corresponds to variable $x_j$ (not pictured). Such paths can easily be constructed using
crossover gadgets as needed.

To enforce that the chosen robots make distinct choices, we enforce the equivalent constraint that if \( R_j \) chooses \( x_i \) and \( R_j' \) chooses \( x_{i'} \), then \( j > j' \Rightarrow i > i' \). Observe that \( R_1 \) is free to choose any \( x_{11} \) unconstrained. In doing so, \( R_1 \) can access a lock gadget (acting as the key) which allows \( R_2 \) to pass through and select any \( x_{2i} \) such that \( i' > i \). Similarly, \( R_2 \) can then open a lock allowing \( R_3 \) to select any \( x_{3i'} \) such that \( i'' > i' \), and so on. Ultimately, all robots \( R_1, \ldots, R_k \) will be forced to choose variables in increasing order. (Some suffix of robots could decline to make any choice at all, but as mentioned above, this cannot help.)

Figure 7 shows the gate gadgets which are to be traversed by the goal robot. It is important to note that gates in the circuit representation of any boolean formula always have fan-out 1. A gate with fan-out larger than 1 represents ‘replicated’ computation which, for a boolean formula, would mean the corresponding sub-formula must be entirely rewritten. Given this, we need only construct gates with fan-out 1.

The goal robot is initially at the output of \( C \) going into the top level gate. If \( C \) is satisfied by the assignment of variables corresponding to the choices made by the variable robots, the goal robot can exit the top level gate of \( C \) which leads directly to the target space. To do so, the goal robot must traverse some set of satisfied gates which is sufficient to satisfy \( C \).

In particular, for every OR gate traversed by the goal robot, the robot must successfully traverse exactly one of the inputs (see Fig. 7 (a); there might be more than one traversable input, but the robot only chooses one to traverse) and for every AND gate traversed, the robot must successfully traverse all of the inputs (see Fig. 7 (b)). If the input to a gate is a variable \( x_i \), the goal robot can traverse that input if and only if some variable robot has chosen \( x_i \) and therefore is present to act as the key to the lock gadget (see Fig. 7 (c)).

Note that if a variable robot has logically chosen variable \( x_i \) (which it has committed to via the one-way gadgets present in the distinct choice gadget; see Fig. 6) it is then able to move freely between all the locks corresponding to \( x_i \). The paths linking logical variable regions and locks are not pictured, but are easy to construct using crossover gadgets as needed (in case the graph matching variable regions to locks is not planar).

In this way, we have constructed an instance of \( RR_- \) which is solvable if and only if \( C \) has a satisfying assignment of weight at most \( k \), and furthermore the construction uses only \( k + 1 \) robots and is therefore parameter preserving. Additionally, we can extract a satisfying assignment to \( C \) from a solution to this \( RR_- \) instance by setting \( x_i = 1 \) if and only if some variable robot \( R_j \) is ever in any part of the region logically corresponding to \( x_i \).

6. Conclusion

We’ve proven the following:

1) One can determine whether a Ricochet Robots or ATOMIX board with a fixed number of robots is solvable in a fixed number of moves in time polynomial in the size of the board. (Section 3),

2) Ricochet Robots and ATOMIX are \( \text{W}[\text{SAT}] \)-hard, and therefore unlikely to be \( \text{FPT} \), and
Ricochet Robots is \( \text{W}[1] \)-hard with respect to some complexity classes:

Conjecture 12. Ricochet Robots is \( \text{W}[1] \)-hard with respect to some complexity classes.

Conjecture 13. It’s as hard (“\#P-hard”) to count the number of solutions to Ricochet Robots which use at most \( n \) moves as it is to count the number of satisfying assignments to a SAT formula.

Conjecture 14. If there’s no solution to QSAT (SAT with “there exists” and “for all” quantifiers) taking less than exponential time, there’s no sub-exponential time solution to Ricochet Robots.

Acknowledgments Adam Hesterberg conducted this research with Government support under and awarded by DoD, Air Force Office of Scientific Research, National Defense Science and Engineering Graduate (NDSEG) Fellowship, 32 CFR 168a. Justin Kopinsky conducted this research with support from the National Science Foundation under grants CCF-1217921, CCF-1301926, and IIS-1447786, the Department of Energy under grant ER26116/DE-SC0008923, and the Oracle and Intel corporations. Both authors thank Erik Demaine for advice and for teaching them about parameterized complexity. [3]

References


Adam Hesterberg received his A.B. degree summa cum laude from Princeton University in 2011 and is now a Ph.D. student at the Massachusetts Institute of Technology studying computational geometry and computational complexity.
Justin Kopinsky graduated from the University of Illinois with a degree in computer science and is now a Ph.D. student at the Massachusetts Institute of Technology.