Logic-Based Encodings for Ricochet Robots

Filipe Gouveia, Pedro T. Monteiro, Vasco Manquinho, and Inês Lynce(✉)

INESC-ID/Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal
{filipe.gouveia,pedro.tiago.monteiro,vasco.manquinho,
ines.lynce}@tecnico.ulisboa.pt

Abstract. Studying the performance of logic tools on solving a specific problem can bring new insights on the use of different paradigms. This paper provides an empirical evaluation of logic-based encodings for a well known board game: Ricochet Robots. Ricochet Robots is a board game where the goal is to find the smallest number of moves needed for one robot to move from the initial position to a target position, while taking into account the existing barriers and other robots. Finding a solution to the Ricochet Robots problem is NP-hard. In this work we develop logic-based encodings for the Ricochet Robots problem to feed into Boolean Satisfiability (SAT) solvers. When appropriate, advanced techniques are applied to further boost the performance of a solver. A comparison between the performance of SAT solvers and an existing ASP solution clearly shows that SAT is by far the more adequate technology to solve the Ricochet Robots problem.

Keywords: Ricochet Robots · Encodings · Boolean Satisfiability · Answer Set Programming

1 Introduction

Puzzles play a key role in artificial intelligence. N-queens problem, towers of Hanoi, Sudoku and Traveling Salesman Person (TSP) are only a few examples of puzzles which are commonly used to motivate and illustrate the use of artificial intelligence in problem solving [21]. Despite not being real problems, solving such puzzles has represented a landmark in the history of artificial intelligence. A recent example is the successful case of solving the GO game. Amazingly, some years ago GO was mentioned in textbooks (e.g. [21]) as an example of a game that was simple for humans and difficult for machines to solve.

Ricochet Robots is another example of such puzzles. Ricochet Robots is a simple board game where the goal is to find the smallest number of moves for a robot to move from its initial position to some target position on the board. Allowed movements are peculiar in the sense that the robot has to be stopped to stay at a given position. A robot movement is stopped either due to a barrier

This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.

© Springer International Publishing AG 2017
E. Oliveira et al. (Eds.): EPIA 2017, LNAI 10423, pp. 657–669, 2017.
DOI: 10.1007/978-3-319-65340-2_54

or due to another robot. What makes the Ricochet Robots game particularly interesting is the fact that it is easy for a human expert to find a solution, although it is very hard to find the best solution, i.e., the solution with as few moves as possible [6]. Finding a solution is an NP-hard problem [10].

The main contribution of this work is a set of logic-based encodings for solving Ricochet Robots. The encodings were carefully designed to achieve a good performance using logic tools, namely Boolean Satisfiability (SAT) tools. Specialized techniques have been used to further boost the efficiency of the solvers. The developed encodings are compared against recent successful solutions [12,13] encoding the problem into Answer Set Programming (ASP) [4]. Experimental results are clear: SAT is more competitive than ASP in this context.

This paper is organized as follows. Section 2 introduces Ricochet Robots. Logic-based encodings are described in Sect. 3. Experimental results are then presented in Sect. 4. Section 5 discusses other logic-based encodings and Sect. 6 concludes the paper and mentions future work.

2 Ricochet Robots

The *Ricochet Robots* board game, also known as *Rasende Roboter* or *Randolph's Robots*, was designed by Alex Randolph in Germany in 1999 [6]. The success of this game is testified by the possibility of playing Ricochet Robots online or even on your mobile.

The original board game is played by two or more players on a 16×16 grid. Each position of the board can have a barrier at the upper, bottom, left or right border. The board is limited by barriers all around it, *i.e.*, each position of the top row has a barrier on the top border, and analogous for the bottom row, and the leftmost and rightmost columns. In the original board game, the four positions in the middle of the board are inaccessible and surrounded by barriers. Figure 1 illustrates a Ricochet Robots board with some barriers placed and the four positions in the middle surrounded by barriers, and therefore inaccessible.

In addition to the 16×16 grid board, the Ricochet Robots game is composed by four robots of different colors. At the start of the game, the four robots are placed randomly on the board, such that:

- each position cannot have more than one robot;
- there are no robots in the four middle (inaccessible) positions.

A target is also placed randomly on the board, except in any of the four center positions. The color of the target corresponds to the color of one of the robots.

The goal of the game is to move any number of robots, one at a time, so that at the end the robot with the target color is on the target position. A player must place the robot in the target position with as few moves as possible. The player wins as many points as the number of moves used. Another target is randomly selected and another round with other player begins. At the end of the game, the player with fewest points wins.

The robots can move on the board with the following rules:

- A robot can move in any of the four directions (up, down, left, right);
- Once a robot moves, it **only** stops when it reaches: (i) a barrier or (ii) another robot.

Take for example the board in Fig. 1. If the blue robot moves up, then it stops in the position (12, 0) - row 12, column 0 - counting as a single move.

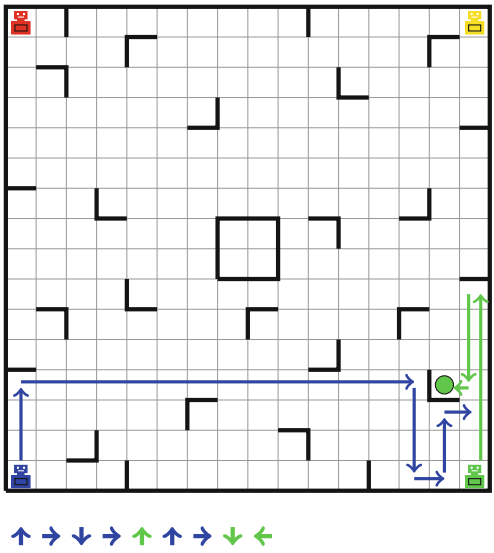


Fig. 1. 16 × 16 Ricochet Robots board. A solution is presented at the bottom. (Color figure online)

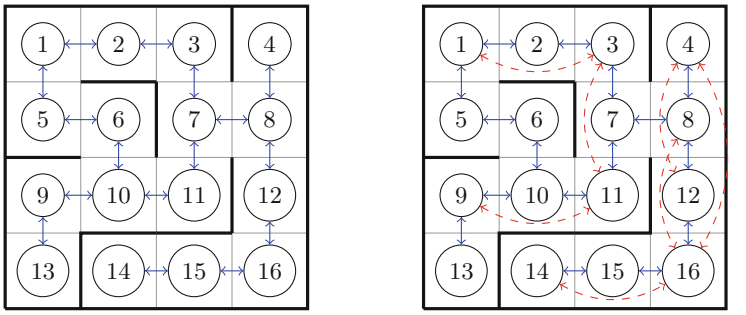
Figure 1 illustrates the minimum number of moves so that the green robot reaches the green target. In this case, 9 moves are required. Notice that in the fourth movement (blue robot going right), the blue robot stops in the position (15, 14) because the green robot in the position (15, 15) is blocking further movements.

3 Encodings of the Ricochet Robots

We start with a base model for the Ricochet Robots problem, followed by the description of a Boolean encoding for the problem.

3.1 Base Model

The Ricochet Robots board, described in Sect. 2, can be represented by a graph where each board position is a vertex. Each two adjacent positions are connected



(a) graph (b) graph with extended edges

Fig. 2. 4 × 4 board (Color figure online)

by an edge as long as there is no barrier between them. Figure 2a illustrates the graph representation of a 4 × 4 board.

Remember that a robot moves in a given direction until it reaches a barrier or another robot. The concept of extended edges is introduced taking this information into account. For each position and for each direction (vertical and horizontal), an edge to other positions in the same row or column is added if and only if there is no barrier between the two positions.

Figure 2b presents the board graph from Fig. 2a where extended edges, marked in dashed red, are added. For example, there is an extended edge connecting vertexes 1 and 3 (same row) since there are no barriers in the path between them. However, there is no extended edge between vertexes 2 and 10 (same column) due to the existence of the barrier between vertexes 2 and 6.

3.2 Boolean Encoding

We assume that a problem is encoded using the conjunctive normal form (CNF). A CNF formula is a conjunction ($\wedge$) of clauses, where a clause is a disjunction ($\vee$) of literals and a literal is a Boolean variable (positive literal) or its negation (negative literal). A formula is said to be satisfied if all of its clauses are satisfied. A clause is satisfied if at least one of its literals is assigned value `true`. A positive (negative) literal is assigned value `true` if the corresponding variable is assigned value `true` (`false`); otherwise it is assigned value `false` (`true`). The SAT problem is to decide whether there exists an assignment to the variables of a CNF formula that satisfies the formula.

The proposed Boolean encoding for the Ricochet Robots problem follows the SAT encoding for the Cooperative Path-Finding (CPF) problem [24]. The CPF problem is very similar to the Ricochet Robots problem, differing only on the rules of movement. In the Ricochet Robots problem each robot movement corresponds to a displacement over one or more positions, only stopped either by a barrier or another robot, whereas in the CPF problem each movement corresponds to a move to an adjacent empty position.

Consider a $d \times d$ board and $G = (V, E)$ the graph representing the board where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of $n = d \times d$ vertexes and E is the set of extended edges defined in Subsect. 3.1. In addition, consider $R = \{r_1, r_2, \ldots, r_\mu\}$ the set of μ robots and η the number of time steps required to achieve the goal, with $\mu, \eta \in \mathbb{N}$.

A Boolean encoding for the Ricochet Robots consists of the following propositional variables:

- $X^t_{j,k}$ with $0 \leq t \leq \eta$, $1 \leq j \leq n$ and $1 \leq k \leq \mu$. $X^t_{j,k}$ is `true` if and only if robot r_k is in position v_j at time step t.
- $Poss^t_{j,l}$ for every $0 \leq t \leq \eta - 1$ and $(v_j, v_l) \in E$. $Poss^t_{j,l}$ is `true` if it is possible to move from vertex v_j to vertex v_l at time step t. A move is possible if the path from vertex v_j to vertex v_l is clear, i.e. there are neither barriers nor robots in the path between the two vertexes and the vertex v_l is a limit when coming from vertex v_j, i.e., v_l is proceeded by a barrier or another robot considering the direction from v_j to v_l.
- M^t_k with $0 \leq t \leq \eta - 1$ and $1 \leq k \leq \mu$. M^t_k is `true` if and only if robot r_k moved at time step t.

For the sake of clarity, in what follows the concepts of position in the board and vertex in the graph are used indistinctly.

The encoding starts by encoding the facts representing the initial state of the board. For each robot r_k, a clause with one literal $X^0_{j,k}$ is added if the robot is placed in vertex v_j at the initial state. The goal state is also encoded with a clause with one literal. The clause $X^\eta_{j,k}$ is added if the goal is to have robot r_k in vertex v_j.

The following clauses are also added to the CNF formula[1]:

- A robot is placed in exactly one vertex at each time step. This is encoded with at least one and at most one constraints, respectively:
  (a) A robot must be in one vertex at each time step

$$\bigvee_{j=1}^{n} X^t_{j,k} \tag{1}$$

  for every time step $0 \leq t \leq \eta$ and for each robot $r_k, 1 \leq k \leq \mu$.
  (b) A robot cannot be in two vertexes at the same time step

$$\bigwedge_{j=1}^{n} \bigwedge_{l=j+1}^{n} \neg X^t_{j,k} \vee \neg X^t_{l,k} \tag{2}$$

  for every time step $0 \leq t \leq \eta$ and for each robot $r_k, 1 \leq k \leq \mu$.

[1] For the sake of clarity, some of the clauses are not written in CNF. In any case, the translation to CNF is trivial.

– A robot either stays in the same vertex v_l or comes from a vertex v_j, such that $(v_j, v_l) \in E$, from which a movement is possible

$$X_{l,k}^{t+1} \implies X_{l,k}^{t} \vee \bigvee (X_{j,k}^{t} \wedge Poss_{j,l}^{t} \wedge M_{k}^{t}) \tag{3}$$

for every time step $0 \leq t \leq \eta - 1$, for each robot $r_k, 1 \leq k \leq \mu$, for every edge $(v_j, v_l) \in E$.

– Given an edge $(v_j, v_l) \in E$, a path between vertexes v_j and v_l is clear if and only if there are no robots in any of the vertexes comprised between the source vertex v_j and the destination vertex v_l.

$$Poss_{j,l}^{t} \implies \bigwedge_{h \in p(j,l)} \bigwedge_{k=1}^{\mu} \neg X_{h,k}^{t} \tag{4}$$

for every time step $0 \leq t \leq \eta - 1$ and for every edge $(v_j, v_l) \in E$. Remember that $p(j, l)$ denotes the set of vertexes in the path from v_j and v_l.

– A vertex is a *stop* vertex for a given direction if there is a robot in the following vertex

$$Poss_{j,l}^{t} \implies \bigvee_{k=1}^{\mu} X_{m,k}^{t} \tag{5}$$

for every time step $0 \leq t \leq \eta - 1$ and for each $(v_j, v_l) \in E$, where v_m represents the next vertex adjacent to v_l considering the direction from v_j to v_l.

– Only one robot can move at each time step

$$\bigwedge_{k=1}^{\mu} \bigwedge_{h=k+1}^{\mu} \neg M_{k}^{t} \vee \neg M_{h}^{t} \tag{6}$$

for every time step $0 \leq t \leq \eta - 1$;

Constraint (5) is only applied if the target position of the movement is not a natural stop vertex, *i.e.*, if it is not a vertex followed by either a barrier or the end of the board.

Observe that (2) and (6) specify at most one constraints on robot locations or movements. In this paper, the pairwise encoding is used for encoding at most one cardinality constraints, since the board dimensions are small. Nevertheless, other encodings [1,7,11,16,17,20] could also be used.

The goal of the Ricochet Robots problem is to move a given robot from its initial position to the target position with the minimum number of moves.

The encoding proposed allows to check if a solution exists for a given time step limit. However, it is possible to obtain an optimal solution for this problem using this encoding following an iterative approach.

Iterative Approach. This approach consists in iteratively using a SAT solver to solve successive instances limited to a given time step η. One possible strategy

is to start by considering $\eta = 0$, and whenever the resulting instance is unsatisfiable, increase η by 1. This process is repeated until a satisfiable instance is found. The solution to the satisfiable instance is clearly an optimal solution of the problem. Observe that this strategy follows an UNSAT-SAT linear search on the value of the optimal solution.

It is known that most of the optimum solutions for the Ricochet Robots problem, from the benchmark used by Gebser *et al.* [12], have at most 20 time steps. Hence, a different strategy to solve the problem is to first encode an instance considering $\eta = 20$ (or another large value where a solution is conceivably to be found). In this case, whenever the generated encoding is satisfiable, η is decreased by 1. The process is repeated until an unsatisfiable instance is found, thus proving that the last satisfiable solution is also an optimal solution to the problem. This strategy follows a SAT-UNSAT linear search on the value of the optimal solution.

If an initial upper bound can be defined, another iterative strategy would be to use a binary search, considering 0 time steps as a lower bound and 20 (or another conceivable value) as an upper bound. In this case, we can first consider $\eta = 10$. If the instance is satisfiable, then 10 is the new upper bound and update η to 5. Otherwise, if the instance is unsatisfiable, then 11 is the new lower bound and update η to 15. The binary search continues by updating η to the average value between the lower and upper bound. The optimal solution has been found when the lower bound is equal to the upper bound.

It might occur that the initial value for the upper bound in the binary search or the linear SAT-UNSAT might not result in a satisfiable instance. In this case, one can define a new upper bound (*e.g.* considering 20 more steps) and apply the same strategy.

4 Experimental Evaluation

In this section, we compare the proposed SAT encoding with the Advanced ASP encoding described in [12][2].

The experimental evaluation was performed using the 256 benchmark instances used by Gebser *et al.* [12]. These benchmarks are based on an authentic 16×16 Ricochet Robots board. In the initial state of the board, each of the four robots is placed on one of the four corners. The instances of the problem were generated by placing the target on each of the 256 possible positions. The target is to be reached always by the same robot. The experiments were run on a Linux machine with Intel Xeon CPU E5-2630 v2 2.60 GHz processors, considering a time limit of 600 s.

We consider the optimization problem of finding the optimum solution for a Ricochet Robots problem instance. For the ASP encoding, *gringo 3.0.5* [15] was used as the grounder and *clasp 3.1.4* [14] was used as the ASP solver[3]. For the

[2] This encoding is available from https://potassco.org/doc/apps/2016/09/20/ricochet-robots.html. Note that the more recent encoding described in [13] is not publicly available.

[3] http://potassco.sourceforge.net/.

incremental approach of the ASP encoding we used *iclingo 3.0.5*. For the SAT encoding, *glucose 4.0*[4] [3] was used.

For the optimization problem, the iterative algorithms described in Sect. 3.2 were implemented on top of the *glucose 4.0* SAT solver. Moreover, for each of the three iterative approaches (linear SAT-UNSAT, linear UNSAT-SAT and binary search), we considered both non-incremental and incremental implementations.

In the non-incremental implementations, the SAT solver is rebuilt at each iteration of the algorithm. On the other hand, in the incremental implementations the SAT solver is never rebuilt from scratch at each SAT call, thus allowing to reuse the learned clauses and other information from previous iterations. This can be achieved by using an assumption-based approach [2,9] that has proved to be effective in several domains [18,22,23].

In incremental implementations, a SAT formula ϕ is built considering an initial limit of ν time steps. In our case, we considered $\nu = 20$. Next, depending on the iterative algorithm being used, for each SAT call on ϕ, a set of assumption literals is defined such that it limits the allowed number of time steps for that specific call. If the number of allowed time steps is η for a board with μ robots, then literals $\neg M_k^t$ (with $\eta \leq t \leq \nu - 1$ and $1 \leq k \leq \mu$) are added as assumptions to the SAT solver call on ϕ. Observe that assumption literals are not unit clauses. Hence, at each SAT call we are only testing if ϕ can be satisfied by assuming that no robot can move beyond η time steps. As a result, the working formula of the optimization algorithm does not have to be rebuilt between iterations.

Table 1 shows the average run time for the solved optimization problem instances, as well as the number of timeouts considering a time limit of 600 s. First, results clearly show that SAT-based approaches outperform the ASP encodings for the tested instances of the Ricochet Robot problem, since all SAT encodings are able to optimally solve a larger set of instances. Moreover, the incremental algorithms also outperform the non-incremental, on both the number of solved instances and the average time spent on solving each instance.

Table 1. Average time (in seconds) of solved optimization instances.

	Time (s)	#Timeouts
ASP advanced optimization	59,34	81
ASP advanced incremental	41,84	79
SAT using binary search	108,37	51
SAT using incremental binary search	56,79	50
SAT using incremental linear search (UNSAT-SAT)	57,43	54
SAT using incremental linear search (SAT-UNSAT)	66,27	45

Figure 3 sheds more light on the data given in Table 1. For each one of the different approaches, represented by a different line, the CPU times used for

[4] http://www.labri.fr/perso/lsimon/glucose/.

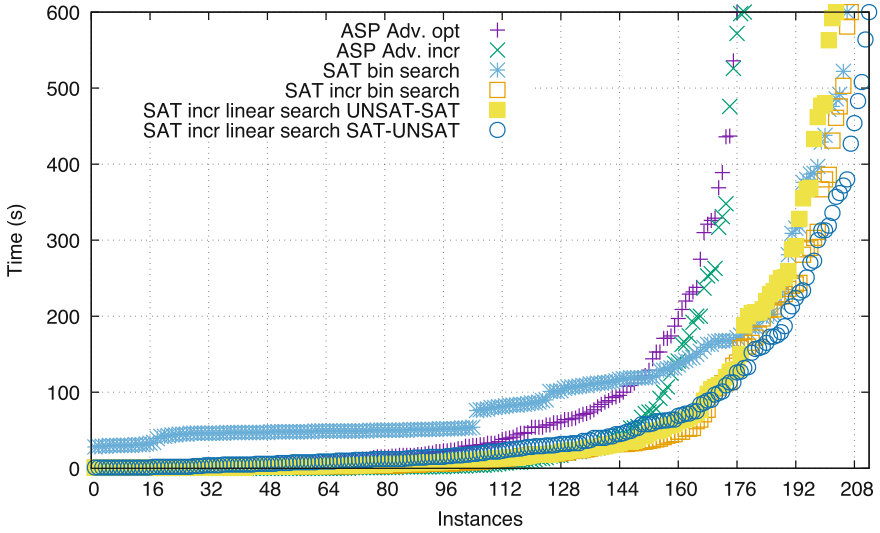


Fig. 3. Time comparison (in seconds) between ASP Advanced encodings and SAT encoding for optimization instances.

solving each of the problem instances are sorted and plotted in the graph. At the end, one has a clear picture of how many instances are solved by each approach for a given CPU time limit. Not only the SAT-based encoding is more effective, but also the incremental approaches are able to solve a larger number of problem instances. Observe that the non-incremental SAT-based binary search algorithm has a more significant overhead, but it is still a more robust approach than using ASP encodings.

Finally, it was observed that the binary search algorithm performs fewer SAT calls than the linear SAT-UNSAT algorithm. However, the time spent in each SAT call was larger. This might result from the fact that, in two consecutive iterations of the binary search algorithm, the set of assumptions has more changes since the time step limit can vary (increase or decrease) more than 1. Hence, the reuse of the previous SAT call context does not seem to be as useful in a binary search approach as when performing a linear SAT-UNSAT algorithm.

5 Other Logic-Based Encodings

The success of using SAT and ASP for solving the Ricochet Robots problem suggests the use of other logic-based tools as the next natural step. We have explored the use of Satisfiability Modulo Theories (SMT) and Constraint Programming (CP) for solving this problem. However, experimental results show that all of these encodings perform worse than ASP and SAT encodings in terms of CPU time. One can argue that these encodings can still be improved, but enough time has been invested for one to say that there should not be a straightforward SMT or CP encoding performing better than SAT or ASP encodings.

The SMT encoding developed for solving the Ricochet Robots problem corresponds to the implementation of the SAT encoding. Other SMT encodings have been tried, but at the end the SAT-like encoding was the best performing encoding. The experiments were run using *z3 4.5*[5] [8] as the SMT solver. Incremental implementations of the Boolean encoding presented were also developed on top of the *z3* SMT solver, similarly to the SAT approaches. Experimental results have shown that the SMT approaches solve fewer instances than the corresponding ASP (and SAT) approaches, and also have greater average times for solved instances.

The proposed CP encoding for solving the Ricochet Robots problem considers the CP language used by MiniZinc [19]. Experiments were run using *MiniZinc 2.1.0*[6] [19] with the Gecode solver provided. Experimental results have shown that the CP approaches solve 4 to 5 times fewer instances than the SAT and ASP approaches. These results are consistent with the results obtained by Barták *et al.* [5] on a problem with similarities to the Ricochet Robots problem. The results show that CP solves far less instances than SAT, despite the instances solved by CP requiring less CPU time. In what follows, the CP encoding is described with more detail.

The CP encoding builds upon a set of integer variables. The input consists of the dimension of the board, the list of robots, the robots initial positions and goals, the list of barriers of the board and the number of time steps. The positions of the board are represented by 2 integers, one representing the line and another representing the column of the board. For each position, four variables are defined, with either value 1 or 0, representing the barriers. The four variables represent the north, south, east, and west directions and have value 1 if the position has a barrier in the corresponding direction and 0 otherwise.

Besides the variables that define the problem instance and, are therefore given as input, two sets of variables are also defined in order to have a solution for the problem. These variables are called *decision variables*. For each robot and for each time step, two variables with integer values are defined, representing the position of each robot at each time step. Similarly to the previous encoding presented, for each robot, and for each time step, a variable with value 0 or 1 is defined representing if either the robot is moving (1) or not (0) in that time step. The constraints of the problem encoding in CSP are:

- The position of a robot at time step 0 must be the initial position of the robot;
- The position of a robot at the limit time step must be the goal position, if that robot has a goal;
- At most one robot moves at each time step;
- If a robot does not move then it stays in the same position;
- If a robot moves, it moves to a valid position;
- If no robot moves in a given time step, then no robot moves in the next time step.

[5] https://github.com/Z3Prover/z3.
[6] http://www.minizinc.org/index.html.

Similarly to CP encoding of a similar problem [5] we found convenient to use an auxiliary predicate, `valid_movement`, to define if a robot moves to a valid position. The predicate `valid_movement` is defined according to the rules of the game, *i.e.*, it is a valid movement if the following rules are satisfied:

- The robot moves to a position on the same row or column;
- There are no robots nor barriers in the path between the initial position and the target position of the movement;
- There is a barrier on the target position considering the direction of the movement or there is a robot in the next position of the target position considering the direction of the movement.

Finally, iterative implementations were developed for the linear searches and binary search on top of the CP encoding. Moreover, the minimization native approach was considered in the experiments.

6 Conclusions and Future Work

Puzzles are common challenges in AI. Apart from their recreational interest, puzzles provide a testbed for the use and comparison of different techniques in problem solving. In this paper, new logic-based encodings for the Ricochet Robots problem are proposed. These encodings are given to SAT solvers using different algorithmic implementations such as binary search, linear search and incrementality. The new proposed encodings allowed to solve a larger set of problem instances than the previously proposed ASP encodings.

Despite clear improvements on previously proposed encodings, the use of a MaxSAT approach is yet to be fully explored, as well as the use of planning tools. Another direction will consider extending the current encodings (and the lessons learned from its development) to similar problems.

References

1. Ansótegui, C., Manyà, F.: Mapping problems with finite-domain variables into problems with Boolean variables. In: International Conference on Theory and Applications of Satisfiability Testing, pp. 1–15 (2004)
2. Audemard, G., Lagniez, J.-M., Simon, L.: Improving glucose for incremental SAT solving with assumptions: application to MUS extraction. In: Järvisalo, M., Van Gelder, A. (eds.) SAT 2013. LNCS, vol. 7962, pp. 309–317. Springer, Heidelberg (2013). doi:10.1007/978-3-642-39071-5_23
3. Audemard, G., Simon, L.: Lazy clause exchange policy for parallel SAT solvers. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 197–205. Springer, Cham (2014). doi:10.1007/978-3-319-09284-3_15
4. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, Cambridge (2003)
5. Barták, R., Dovier, A., Zhou, N.F.: Multiple-origin-multiple-destination path finding with minimal arc usage: complexity and models. In: International Conference on Tools with Artificial Intelligence (ICTAI), pp. 91–97. IEEE (2016)

6. Butko, N., Lehmann, K.A., Ramenzoni, V.: Ricochet robots-a case study for human complex problem solving. Project thesis from the Complex System Summer School, Santa Fe Institute (2006)
7. Chen, J.: A new SAT encoding of the at-most-one constraint. In: International Workshop on Modelling and Reformulating Constraint Satisfaction Problems (2010)
8. Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). doi:10.1007/978-3-540-78800-3_24
9. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004). doi:10.1007/978-3-540-24605-3_37
10. Engels, B., Kamphans, T.: Randolphs robot game is NP-hard!. Electron. Notes Discret. Math. **25**, 49–53 (2006)
11. Frisch, A.M., Peugniez, T.J., Doggett, A.J., Nightingale, P.: Solving non-Boolean satisfiability problems with stochastic local search: a comparison of encodings. J. Autom. Reason. **35**(1–3), 143–179 (2005)
12. Gebser, M., Jost, H., Kaminski, R., Obermeier, P., Sabuncu, O., Schaub, T., Schneider, M.: Ricochet robots: a transverse ASP benchmark. In: Cabalar, P., Son, T.C. (eds.) LPNMR 2013. LNCS, vol. 8148, pp. 348–360. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40564-8_35
13. Gebser, M., Kaminski, R., Obermeier, P., Schaub, T.: Ricochet Robots reloaded: a case-study in multi-shot ASP solving. In: Eiter, T., Strass, H., Truszczyński, M., Woltran, S. (eds.) Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation. LNCS, vol. 9060, pp. 17–32. Springer, Cham (2015). doi:10.1007/978-3-319-14726-0_2
14. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: *clasp*: a conflict-driven answer set solver. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS, vol. 4483, pp. 260–265. Springer, Heidelberg (2007). doi:10.1007/978-3-540-72200-7_23
15. Gebser, M., Schaub, T., Thiele, S.: GrinGo: a new grounder for answer set programming. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS, vol. 4483, pp. 266–271. Springer, Heidelberg (2007). doi:10.1007/978-3-540-72200-7_24
16. Gent, I.P., Nightingale, P.: A new encoding of All different into SAT. In: International Workshop on Modelling and Reformulating Constraint Satisfaction Problems (2004)
17. Klieber, W., Kwon, G.: Efficient CNF encoding for selecting 1 from N objects. In: International Workshop on Constraints in Formal Verification (2007)
18. Martins, R., Joshi, S., Manquinho, V., Lynce, I.: Incremental cardinality constraints for MaxSAT. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 531–548. Springer, Cham (2014). doi:10.1007/978-3-319-10428-7_39
19. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: MiniZinc: towards a standard CP modelling language. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 529–543. Springer, Heidelberg (2007). doi:10.1007/978-3-540-74970-7_38
20. Prestwich, S.: Variable dependency in local search: prevention is better than cure. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 107–120. Springer, Heidelberg (2007). doi:10.1007/978-3-540-72788-0_14
21. Russell, S.J., Norvig, P.: Artificial Intelligence - A Modern Approach, 3rd edn. Pearson Education, London (2010)

22. Sharma, A., Sharma, D.: An incremental approach to solving dynamic constraint satisfaction problems. In: Huang, T., Zeng, Z., Li, C., Leung, C.S. (eds.) ICONIP 2012. LNCS, vol. 7665, pp. 445–455. Springer, Heidelberg (2012). doi:10.1007/978-3-642-34487-9_54
23. Shtrichman, O.: Pruning techniques for the SAT-based bounded model checking problem. In: Margaria, T., Melham, T. (eds.) CHARME 2001. LNCS, vol. 2144, pp. 58–70. Springer, Heidelberg (2001). doi:10.1007/3-540-44798-9_4
24. Surynek, P.: Simple direct propositional encoding of cooperative path finding simplified yet more. In: Gelbukh, A., Espinoza, F.C., Galicia-Haro, S.N. (eds.) MICAI 2014. LNCS, vol. 8857, pp. 410–425. Springer, Cham (2014). doi:10.1007/978-3-319-13650-9_36