Solving Nurikabe with Ant Colony Optimization

Martyn Amos
Northumbria University
Newcastle upon Tyne, UK
martyn.amos@northumbria.ac.uk

Matthew Crossley
Manchester Metropolitan University
Manchester, UK
M.Crossley@mmu.ac.uk

Huw Lloyd
Manchester Metropolitan University
Manchester, UK
Huw.Lloyd@mmu.ac.uk

ABSTRACT

We present the first nature-inspired algorithm for the NP-complete Nurikabe pencil puzzle. Our method, based on Ant Colony Optimization (ACO), offers competitive performance with a direct logic-based solver, with improved run-time performance on smaller instances, but poorer performance on large instances. Importantly, our algorithm is "problem agnostic", and requires no heuristic information. This suggests the possibility of a generic ACO-based framework for the efficient solution of a wide range of similar logic puzzles and games. We further suggest that Nurikabe may provide a challenging benchmark for nature-inspired optimization.

KEYWORDS

Puzzle game, NP-complete, Combinatorial optimization, Ant colony optimization.

ACM Reference Format:
Martyn Amos, Matthew Crossley, and Huw Lloyd. 2019. Solving Nurikabe with Ant Colony Optimization. In *Genetic and Evolutionary Computation Conference Companion (GECCO '19 Companion), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3319619.3338470

1 INTRODUCTION

Nurikabe is a Japanese *pencil puzzle* [2], the wider set of which includes well-known problems such as Sudoku [3] and Hashiwokakero [1]. The puzzle is played on a rectangular grid of white cells, some of which initially contain numbers. A successful solution to the puzzle requires the player to shade in (colour black) non-numbered cells according to the following rules: (1) Black cells must form a single *continuous* region (the "wall"); (2) Every numbered cell must occupy its own disjoint white region (an "island") whose size, in terms of the number of cells it occupies, is the same as the number label of that cell; (3) There must not exist any 2×2 black regions.

In Figure 1 (left-middle), we show an example Nurikabe puzzle and a correct solution. Note that, in the solution, each island contains a number of white squares that is equal to its labelled value, the black wall occupies a single continuous region (with no 2×2 regions), and no islands are touching. We also show, in Figure 1 (right), an *invalid* attempt at a solution, with the following problems highlighted: (A) Numerous 2×2 blocks of black squares, (B) Island containing more than one value (which might be interpreted as touching "4" and "3" islands, (C) Island containing the wrong number of white squares, (D) Discontinuous wall.

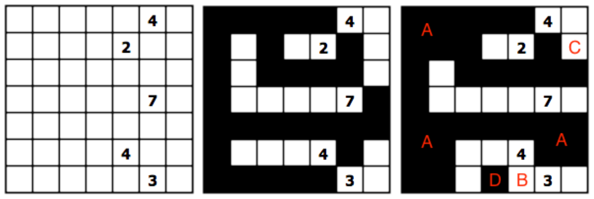


Figure 1: The structure of a Nurikabe puzzle instance (left), a correct solution (middle), and an invalid solution (right).

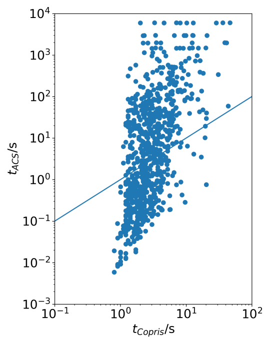
The problem of solving Nurikabe is known to be NP-complete [5]. As such, it presents a useful challenge for new algorithms. In this paper, we present a novel method based on the well-established Ant Colony System (ACS) algorithm [4], and compare its performance with an existing Constraint Programming algorithm [8]. To the best of our knowledge, the work in this paper represents the first attempt to solve Nurikabe using a stochastic optimization algorithm.

2 ALGORITHM OVERVIEW

Rather than "constructing" the wall around the islands, we instead colour *all* cells black at the outset (apart from numbered cells), and then individually grow the *islands*, by repeatedly colouring selected cells white. We now describe how the ACS algorithm is applied to Nurikabe. At each iteration, a number of "ants" are given their own local copy of the game board, and each ant is placed on a randomly-selected numbered cell (that is, the "seed" of an island). Each ant then moves around the board, gradually "growing" the island by colouring cells white, until either the island reaches the desired size, or no more moves are possible. Movement is informed by the global pheromone trail. The ant then moves to the next numbered cell, and the process continues. At the end of each generation, we therefore have a number of possible solutions to the problem (one per ant); we then select the best solution (details below) and "reward" its white island cells with additional pheromone. In this way, future generations of ants are biased towards those cells. Specifically, at each iteration within a generation, each ant constructs a set of possible *candidate cells* to which it might move, based on both the game rule constraints and the current state of the grid. The initial candidate set is constructed by taking *all* cells that border the *current island*, and is then *pruned* according to the following two rules: (1) Remove any cell that is a *cut cell* for the black "wall" region (i.e., any cell whose removal would lead to the wall becoming disconnected), (2) Remove any cell that is adjacent (horizontally/vertically) to an existing island. Once the candidate set has been pruned, the next cell is selected according to ACS principles, and is added to the current island (this process includes a local pheromone update). Once this island is completely filled, the ant moves to the next island, and the process repeats until all ants have completed their moves. The best-performing ant is then selected (according to a cost function which counts how many constraints are broken), the global pheromone matrix is updated, and the "best value evaporation" [7] operator is applied.

3 EXPERIMENTAL RESULTS

In this Section we present the results of experimental runs of a Java implementation of our algorithm and, for comparison, the Copris Constraint Programming code [8]. All runs were carried out using a single core of a Xeon E5-2640 v4 2.40 GHz processor, on a machine running Ubuntu Linux. We use the collection of 911 Nurikabe instances available from [6], ignoring three instances in which some islands have an unspecified number (a total of 908 instances). The instances range in size (total number of cells $m \times n$), from 9 to 2500 cells, and all have a unique solution. We ran the ACS code 100 times per instance. In all runs we set a timeout of one minute of wall-clock time. For all ACS runs, we used the following parameters: $m = 10$, $\rho = 0.2$, $\xi = 0.1$, $q_0 = 0.9$, and $f_{\mathrm{BVE}} = 0.001$ (see [7] for the meanings of parameters). We ran the Copris solver [8] on each of the instances, using the same machine (and Java Virtual Machine) as for the ACS solver. Since the Copris solver is deterministic, we performed only one run per instance, again with a one minute wall-clock time limit. Figure 2 (left) shows a scatter plot of average solution times for all instances that were solved by both algorithms; for ACS this measure includes the timeouts, and is the total run time divided by the number of successful runs. The upper half of the plot shows instances where our ACS method performs worse, and the lower half shows instances where our ACS method performs better than the Copris solver. Figure 2 (right) shows the success rates as a function of instance size. For ACS, this shows the fraction of instances for which any run found a solution. In Figure 3, we show the largest solution found by our ACS solver.



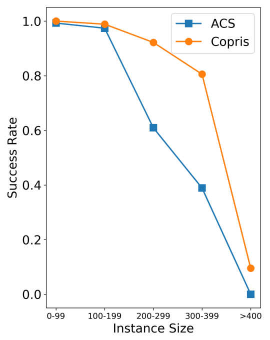


Figure 2: Scatter plot of solution times for all instances solved by both solvers (left), and solution rates as a function of instance size (right).

Our results show that the ACS-based algorithm out-performs the Copris solver in terms of runtime on the smallest instances, but the Copris solver performs better in terms of runtime and

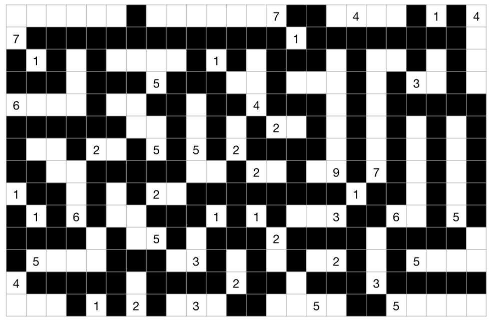


Figure 3: Largest solution found with ACS; instance with 336 cells.

success rate on the larger (greater than 200 cells) instances. On the smallest instances (0-99 cells), ACS is generally quicker to achieve a solution, with 100 instances solved in a shorter average time by ACS, compared to 35 solved in shorter time by Copris. For the medium-sized instances (100-199 cells), ACS is quicker for 259 instances, and Copris for 335. On larger instances, the failure rate for ACS within the one minute timeout is substantially greater than for Copris. It is worth noting that the runtime for the Copris solver is always of order a second or longer, whereas ACS runs on the small instances often complete in milliseconds. These results contrast with the results on Sudoku [7], in which ACS significantly outperformed the best direct solvers on harder instances. This may suggest that Nurikabe offers a far more challenging benchmark for ACO algorithms than Sudoku.

4 CONCLUSIONS

In this paper we have presented the first nature-inspired algorithm for the computationally hard Nurikabe pencil puzzle. We compared the performance of our method against that of an existing logic-based solver, and found that our algorithm was faster on smaller instances. Importantly, our method relies on next to no heuristic information about the puzzle (that is, "tips" for its solution), embedding only the game rules. We argue, therefore, that ACS offers a promising method for the rapid solution of such puzzles. Future work will focus on the development of a general-purpose pencil puzzle solver, incorporating Nurikabe and other games, and investigation of their use as benchmarks for nature-inspired optimization.

REFERENCES

[1] Daniel Andersson. 2009. Hashiwokakero is NP-complete. *Inform. Process. Lett.* 109, 19 (2009), 1145–1146.
[2] Alex Bellos. 2017. *Puzzle Ninja.* Guardian Books/Faber & Faber.
[3] Jean-Paul Delahaye. 2006. The science behind Sudoku. *Scientific American* 294, 6 (2006), 80–87.
[4] Marco Dorigo and Luca Maria Gambardella. 1997. Ant colony system: a cooperative learning approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation* 1, 1 (1997), 53–66.
[5] Markus Holzer, Andreas Klein, and Martin Kutrib. 2004. On the NP-completeness of the Nurikabe pencil puzzle and variants thereof. In *Proceedings of the 3rd International Conference on FUN with Algorithms.* 77–89.
[6] Angela Janko and Otto Janko. [n.d.]. Nurikabe instances. Available at https://www.janko.at/Raetsel/Nurikabe/.
[7] Huw Lloyd and Martyn Amos. 2018. Solving Sudoku with Ant Colony Optimisation. *arXiv preprint arXiv:1805.03545* (2018).
[8] Naoyuki Tamura. [n.d.]. Nurikabe solver in Copris. Available at http://bach.istc.kobe-u.ac.jp/copris/puzzles/nurikabe/.