Evolutionary Algorithm for Generation of Entertaining Shinro Logic Puzzles

David Oranchak

http://oranchak.com

Abstract. A Shinro puzzle is a type of deductive reasoning puzzle that originated in Japanese periodicals. To solve the puzzle, one must locate twelve hidden stones on an 8x8 grid using only clues in the form of stone counts per row and column, and arrows placed in the grid that point to some of the hidden stones. Construction of these puzzles by hand is tedious. We explore the use of a simple genetic algorithm that automates construction of Shinro puzzles with desirable qualities which improve their entertainment value.

Key words: genetic algorithm, logic puzzles

1 Introduction

Puzzability is a company that specializes in creating and selling a variety of puzzles. In 2007, the company developed and sold a new kind of logic puzzle to Southwest Airlines. The airline began publishing these new puzzles as a regular feature in *Spirit*, their inflight magazine. Named *Shinro*, a Japanese word that means "compass bearing", the puzzle is a simple 8x8 square grid containing twelve holes in unknown locations. Along the top of the grid is an additional row of eight squares. Each square indicates the total number of holes hidden in the column under that square. Similarly, along the left side of the grid is an additional column of eight squares, each indicating the total number of holes hidden in the row to the right of that square. Directional arrows within the puzzle point to some of the hidden holes. Example puzzles are shown in **Figure 1**. One solves the puzzle by making a series of reasoned deductions in a process of elimination, identifying squares that *must* contain holes, and squares that *cannot possibly* contain holes. The puzzle solver continues until all twelve holes have been located.

Henceforth, we refer to the holes as "stones" to reflect our own implementation of Shinro.

The popularity of the Shinro puzzles in *Spirit* has led to development of various Shinro video games [1][2][3] and web sites [4][5]. This suggests there is a market for creating new Shinro puzzles. Puzzles are created by selecting hidden stone locations, directional arrow locations, and a desired difficulty in deducing the solution. Construction of these puzzles via automation eliminates the time-consuming and tedious task of designing valid and entertaining puzzles

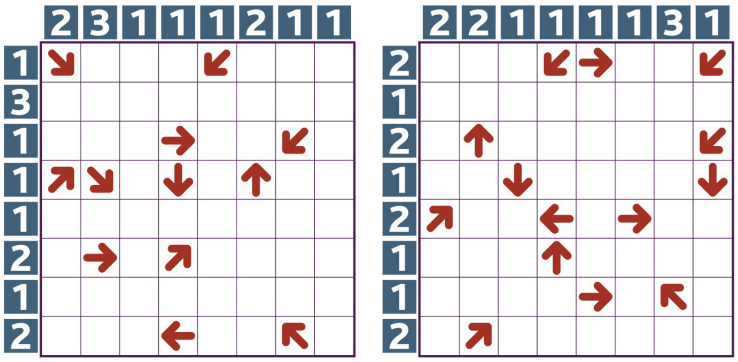

Fig. 1. Sample Shinro puzzles published in *Spirit* magazine [6]

by hand. The number of possible Shinro puzzle configurations is astronomical. Each of the 64 positions of an 8x8 puzzle grid can have one of ten possible values (empty, a stone, or one of eight kinds of arrows), bringing the total search space size to 10^{64}. We can reduce this by observing that every valid board must have exactly 12 stones. Thus there are $s =_{64} C_{12} = \frac{64!}{12!(64-12)!}$ possible ways to place stones, and $c = 9^{52}$ remaining possible combinations for the grid positions that lack stones, bringing the total search space down to $s \times c = 8.0 \times 10^{40}$ puzzles. Still much too large for exhaustive searches. A genetic algorithm is suitable for searching such a large space [7]. Mantere and Koljonen were able to show that a GA was an efficient method of generating another kind of constraint satisfaction problem: the sudoku puzzle [8].

2 Methodology

Automatic construction of Shinro puzzles requires development of an algorithm that can automatically solve them. This solver algorithm is used to measure the validity and difficulty of a constructed puzzle, and to estimate its entertainment value. This requires identification of each of the techniques used to solve the puzzles. A similar strategy employed by Ortiz-Garcia et. al. was effective for generating picture-logic puzzles [9].

Stones are located by identifying logical consequences to observations of the puzzle grid state. Similar observations lead to the identification of locations that cannot contain stones. Such locations are marked as "filled". Some of these identifications are easy or trivial to make, while others can be quite challenging. The logical deductions described below are used to automatically solve many Shinro puzzles. The automated solver is the basis for the evolutionary algorithm discussed later.

Count of unfilled positions equals number of remaining stones: If the number of free positions along a row (column) equals the stone count of that

row (column), less the number of stones that have already been placed in that row (column), then these positions must contain stones (**Figure 2**).

Row or column count is satisfied: If the number of stones placed in a row (column) exactly matches the stone count for that row (column), then all free positions can be marked as filled (**Figure 3**).

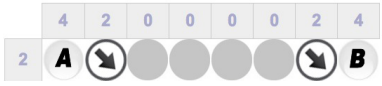

Fig. 2. A and B must be locations of stones since the stone count for that row is two.

Fig. 3. A, B, and C can be marked as filled because the row count is satisfied by the stone in the third column.

Arrow points to only one free square: An *unsatisfied* arrow is an arrow that points to no placed stone. Therefore, there is a hidden stone somewhere along the unsatisfied arrow's path. If there is exactly one free position along the arrow's path, then it must contain a stone (**Figure 4**).

One stone remains to be placed, and there is a horizontal or vertical arrow: Consider a row (column) whose stone count, less the number of placed stones along the row (column), is equal to one. If this row (column) contains an unsatisfied horizontal (vertical) arrow, it must point towards the one remaining stone. Therefore, all free positions the arrow points away from cannot contain stones, and thus can be marked as filled (**Figure 5**).

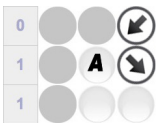

Fig. 4. A has to be a stone. It is the only free position the arrow to its upper right is pointing to.

Fig. 5. A, B, and C cannot possibly be stones, since the horizontal arrow points towards the region of the row that contains the one remaining stone.

Locations can be excluded based on identification of non- intersecting arrow paths: This type of deduction identifies regions of rows (columns) in the puzzle in which some number of possible stone placements can be marked as filled due to constraints imposed by unsatisfied arrows pointing into those regions. **Figure 6** depicts an example.

Let X denote all free positions within some subset of rows (columns) of the puzzle, where $|X| > 1$. Let n denote the total count of stones remaining to be placed within the rows (columns) of X. Let A denote some subset of unsatisfied arrows of the puzzle. Let us require that each arrow in A has free positions along its path that are contained entirely within X, and that no arrow in A has a path

whose free positions intersect the free positions in the path of another arrow in A. Let P denote the set of positions within X that are covered by the free positions along the paths of every arrow in A. If $|A| = n$, then we know that all remaining stones must be located somewhere in positions in P. Therefore, no stones will be found in $X \setminus P$, and these remaining positions can be marked as filled.

These moves can be quite difficult to locate.

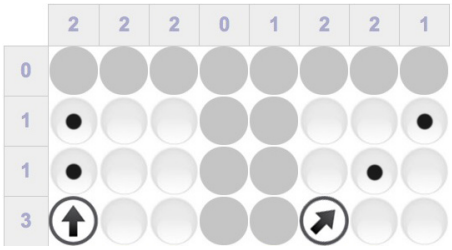

Fig. 6. Two arrows indicate stones along the dotted paths. Since the paths do not intersect, there must be a total of two stones represented by the paths. The covered rows have a total stone count of two. Therefore, none of the non-dotted positions along the covered rows can possibly be a stone, and thus they can all be marked as filled.

Locations can be excluded based on satisfiability of remaining arrows: A free position can be marked as filled if placing a stone there causes an arrow to become unsatisfiable (**Figure 7**).

Locations can be excluded based on the *pigeonhole principle*: This is another type of move that can be difficult to locate. In this move, unsatisfied arrows impose constraints on a row (column) such that their satisfaction results in narrowing down the possible choices for filled positions along the row (column). This narrowing of choices for the filled position entails a reduction in the possible locations for remaining stones to be placed in that row (column). **Figure 8** depicts an example.

Let X denote a row (column). Let n be the number of stones remaining to be placed in X. Let $m > n$ be the number of unfilled positions in X. Let P be the set of unfilled positions in X, whose column (row) count of remaining stones, less the count of placed stones, is equal to one. A total of $m-n$ positions along X will be marked as filled. We seek $m - n$ unsatisified arrows, A, whose paths contain unfilled positions. Let us require that there is no arrow in A whose unfilled positions intersect the unfilled positions of another arrow in A, or whose unfilled positions intersect P. Let us also impose that every unfilled position represented by A must share the column (row) of a position in P. Thus, satisfaction of an arrow in A identifies a subset of P in which a filled position must be located. Let S be the set of all such subsets. Each time a subset is added to S, the possible stone positions in X is reduced by one. Once this count reaches $m - n$, then we know that stones must be located in any position in X that is not in P.

Stone placements can be excluded due to impossible scenarios: An attempt to place a stone at a given position can be excluded if the placement results in subsequent moves that lead to an impossible or invalid puzzle state. It is assumed that such brute force attempts to solve Shinro puzzles by exhaustion have little entertainment value. Therefore, we will not consider these moves, except for the simple form shown in **Figure 7**.

There may be further types of useful deductions that are not identified above.

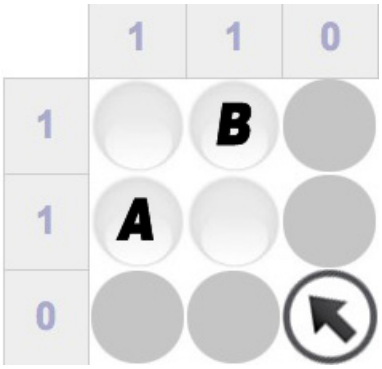

Fig. 7. Placement of a stone at A or B results in an arrow that is impossible to satisfy. Therefore, A and B can be marked as filled.

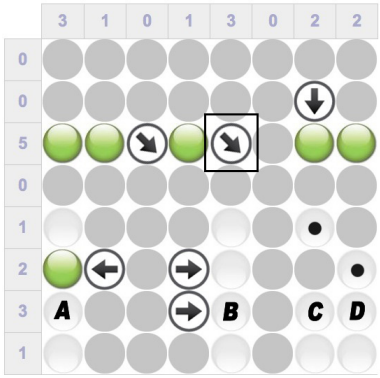

Fig. 8. *Pigeonhole principle*: The indicated arrow points to a stone along the dotted path. Either placement will satisfy its corresponding column count. Therefore, either C or D will be marked as filled. Thus there are only three possible stone placements remaining along the row $ABCD$. Since the filled position must be C or D, and the row count is also three, A and B must be stones.

2.1 Entertainment Value

We assume that completely randomized puzzle configurations are not very fun to play in the long term. We want the generator to create a variety of interesting and entertaining puzzles. Therefore, we need to measure or estimate these qualities. Some of the factors guiding the puzzle designs are listed below.

- We require a degree of control of the difficulty level of generated puzzles. They should neither be too difficult nor too easy to solve. Trivial puzzles and unusually difficult puzzles will quickly tire players.
- Puzzles should have exactly one valid solution.
- We should generate puzzles whose stones and/or arrows form a variety of interesting patterns and symmetries that are more pleasing to view than completely randomized puzzles.

3 Evolutionary Algorithm

A simple genetic algorithm is used to generate the Shinro puzzles. The values of several configuration parameters determine the type of puzzle that is generated. A selection of fitness functions is available to guide the development of an assortment of puzzle varieties. A *target pattern* constraint can be optionally configured, restricting the possible values for some set of positions of the puzzle grid. This permits evolution of puzzles that conform to pre-designed patterns, shapes, and configurations.

3.1 Genome Encoding

Generated puzzles are encoded in an nxn matrix of integer values representing the contents of puzzle grid squares. The possible grid square interpretations are: empty, hidden stone, and arrow pointing in one of eight possible directions.

The optional target pattern constraint can be specified as another nxn matrix of constraint values. This matrix is consulted during population initialization and mutation to prevent insertion of grid square values that violate the given constraints. The constraint types are: Square has no constraint, square must contain an arrow with a specific direction, square must contain a stone, square must be empty, square must contain any arrow, square must contain a stone or any arrow, and square must not be an arrow.

3.2 Initialization

The population of genomes is set to a small size, 10, due to the amount of time needed to perform the fitness evaluation described below. Genomes are initialized to random values. If a target pattern constraint is present, initialization of constrained grid squares is limited to values that do not violate the constraint.

3.3 Genetic Operators

At each generation, a new population is constructed using tournament selection of size three. Elitism is used to prevent loss of good individuals. This is implemented by simply tracking the best evolved individual for the entire run, and copying this individual into each new population.

Crossover is not implemented due to its assumed destructive effect on the satisfaction of constraints of the Shinro puzzles, namely the required fixed number of stones, the set of valid arrows that each point to at least one stone, and the optional target pattern constraint. Further research is necessary to determine the effectiveness of various crossover operators.

When the new population is constructed, mutation is applied by randomly selecting one of a number of available mutators:

- Loop through the puzzle grid and probabilistically change the value of a square. The mutation rate is itself randomly selected from the range $[0, 1]$.

- Swap two randomly selected puzzle grid squares. Repeat n times, where n is randomly selected from the range $[1, 3]$.
- Add an arrow to a randomly selected square.
- Randomly select an arrow and remove it.
- Add a stone to a randomly selected square.
- Randomly select a stone and remove it.

Further, the mutator randomly decides whether or not to enforce symmetry, and whether or not to enforce rotational symmetry. If symmetry is enforced, the selected mutator projects the resulting mutations about the horizontal and vertical axes of the puzzle, both located at the center of the puzzle grid. But if rotational symmetry is enforced, the selected mutator instead projects a puzzle quadrant's mutation into the remaining quadrants by repeatedly rotating the quadrant of the originating mutation by 90 degrees.

The algorithm stops when a minimum number of generations has passed without any improvement to the best genome. The run's statistics and best genome are noted, the population is again reset, and the evolutionary algorithm is repeated. This is done to collect many generated puzzles of high fitness. A final step runs a brute force search on the generated puzzle to ensure that only one unique solution is possible. This computation is not performed during fitness evaluation due to its significant performance impact.

3.4 Fitness Function

The automated solver for generated Shinro puzzles is the basis for fitness computations. The solver inspects each generated puzzle and attempts to solve it using a greedy approach. In the greedy approach, the algorithm looks for easy moves first before proceeding to more difficult moves. Once a move is found, the search for more difficult moves is aborted, and the found move is applied to the puzzle state. The search then repeats until all stones have been located, or no further moves are located. This greedy approach is used to reduce the execution time of the search. The algorithm tracks the counts and types of moves. These counts are used in fitness computations.

We used several fitness functions to evolve a variety a puzzles. Each fitness function shares a common factor: the normalized error count:

$$\epsilon = \frac{1}{1 + |s - s'| + a + v + |m - m'| + e} \tag{1}$$

s is the number of stones required for this puzzle. s' is the number of stones encoded in the genome. a is the number of arrows found that do not point to a stone. If a target pattern constraint is used, v is the number of violated constraints found; otherwise, it is zero. m is the minimum number of solver moves required for acceptance of this generated puzzle. m' is the actual number of moves (of any difficulty level) used by the solver to solve this puzzle. If we are evolving for symmetry, e is the number of grid positions that violate symmetry; otherwise, it is zero.

By itself, this single factor applies evolutionary pressure to locate puzzles that satisfy fundamental constraints. We evolve other desirable puzzle properties by introducing other factors into the fitness function:

- Maximizing the number of moves required to solve the puzzle:

$$f = \epsilon \times [1 - \frac{1}{(1 + steps_d)}] \tag{2}$$

  where $steps_d$ is the number of moves used by the solver to reach a solution. d represents the difficulty factor. When evolving for maximum count of all move types, all move difficulties are counted, and $steps_d$ is equivalent to m'. Otherwise, we only count moves of the specified difficulty level; thus, $steps_d$ may be less than m'.
- Maximizing the clustering of stones and/or arrows:

$$f = \epsilon \times [1 - \frac{1}{(1 + steps_d)}] \times [1 - \frac{1}{1 + c}], \tag{3}$$

$$c = \sum_{i,j} s(i,j), \tag{4}$$

$$s(i,j) = \sum_{u=i-1,v=j-1,(u,v)\neq(i,j)}^{u=i+1,v=j+1} count(u,v) \tag{5}$$

  The following applies only if the item at position (i, j) is the type of item for which we are clustering: $count(u, v)$ returns 1 if the item at (i, j) is found at (u, v) and (u, v) is horizontally or vertically adjacent to (i, j). This represents a *strong* clustering. $count(u, v)$ returns $\frac{1}{4}$ if the item at (i, j) is also found at (u, v) and (u, v) is diagonally adjacent to (i, j). This represents a *weak* clustering. $count(u, v)$ returns 0 if the item at (i, j) is not found at (u, v).

4 Results and Conclusions

The first optimization task was to maximize the number of moves required to solve the generated Shinro puzzles. **Figure 9** shows a high-fitness evolved puzzle requiring 37 moves in the automated solver to reach a solution. Despite the high number of moves, the puzzles generated this way remain very easy to solve, because the high numbers of arrows generally give up many clues about stone locations.

Evolution of puzzles with horizontal, vertical, and rotational symmetry of stone and arrow positions was greatly improved by introducing symmetric operations to the standard mutation operators. For instance, instead of changing a single grid value, each of the symmetric grid values are simultaneously changed during a single mutation. **Figure 10** shows a generated symmetric puzzle. **Figure 11** shows a generated puzzle that has rotational symmetry.

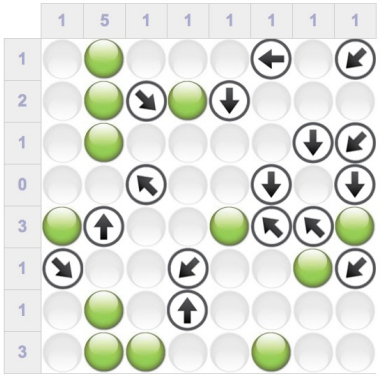

Fig. 9. Evolved puzzle configuration that requires 37 moves to solve.

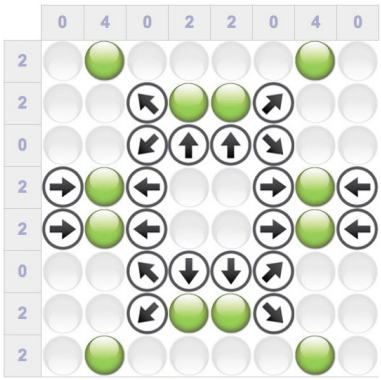

Fig. 10. Puzzle with symmetry. Requires 24 moves to solve.

Similarly, when a target pattern constraint is specified, we wanted to prevent mutation from introducing violations to the specified constraint (**Figure 12** depicts a sample constraint that the stones must form the letter "A"). The modified mutation operator checks the constraint prior to changing a grid square value. If the change violates the constraint, the change is suppressed.

To control the difficulty of generated puzzles, several evolutionary runs concentrated on maximizing specific types of moves. The most difficult move represented by the automated solver is the move based on the "pigeonhole" principle. The easiest of any other available move is greedily selected by the automated solver, reflecting the assumed bias of human players to locate the easiest moves first. Pigeonhole moves are not selected unless no other moves are available. Thus, puzzles that maximize these moves were challenging to discover. One such discovered puzzle, requiring four distinct pigeonhole moves, is depicted in **Figure 13**.

The simple genetic algorithm was very effective for the construction of a wide variety of interesting, challenging, and fun Shinro puzzles. Several aspects of our approach warrant further study. The first is the impact of the greedy aspect of the automated solver on evolutionary search. To what extent does the order of moves affect the optimization of puzzle metrics? Another area of future study is the identification of move types that are not represented herein. Can they be discovered by evolutionary search? Would their discovery greatly affect the current evaluation of desirable qualities of generated puzzles? Also, other metrics of desirable puzzle qualities, such as balance of move types, or arrow density, may be useful in further research.

Acknowledgments. I wish to thank Arash Payan, creator of the wonderful iPhone game *Jabeh* which is based on Shinro, and Amy Goldstein of Puzzability, the company that created Shinro puzzles, for their contributions to my research on the background and origins of Shinro puzzles.

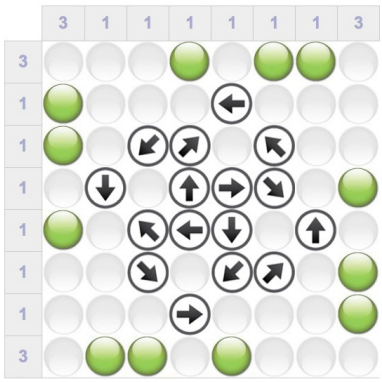

Fig. 11. Puzzle with rotational symmetry, including arrow direction. Requires 30 moves to solve.

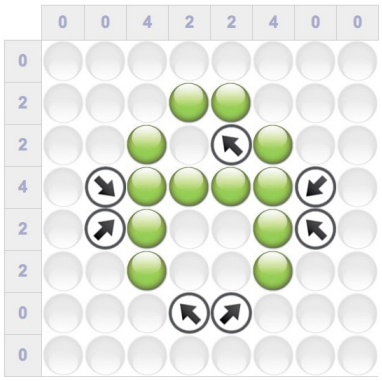

Fig. 12. Puzzle evolved with target pattern constraint: Stones must form "A" shape. Requires 19 moves to solve.

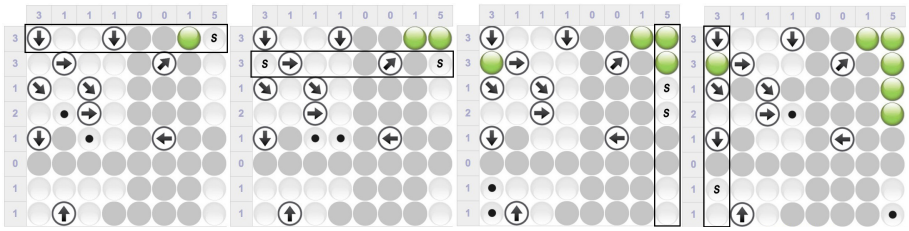

Fig. 13. Evolved puzzle that requires four "pigeonhole" moves. Moves proceed from left to right. Dots indicate sets of possible stone positions that further constrain the indicated row or column. Each *s* indicates the deduced location of a stone due to the necessary placement of another stone somewhere among the dots.

References

1. Jabeh - Logic Game for iPhone and iPod Touch, `http://jabeh.org/`
2. Shinro Mines for iPhone and iPod Touch, `http://www.benjiware.com/main/Shinro_Mines.html`
3. Shinro - the Next Sudoku (iPhone game), `http://web.me.com/romancini/Far_Apps/Shinro.html`
4. Shinro Puzzles, `http://shinropuzzles.web.officelive.com`
5. Sternenhimmel, `http://www.janko.at/Raetsel/Sternenhimmel/index.htm`
6. Fun and Games. In: Southwest Airlines Spirit Magazine (2007)
7. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley (1989)
8. Mantere, T. Koljonen, J: Solving, rating and generating Sudoku puzzles with GA. In: IEEE Congress on Evolutionary Computation (2007)
9. Ortiz-García, E.G., Salcedo-Sanza, Sancho, Leiva-Murillob, J.M., Pérez-Bellidoa, A.M., Portilla-Figuerasa, J.A.: Automated generation and visualization of picture-logic puzzles. Computers & Graphics 31(5):750-760, 2007.