Generating Solvable and Difficult Logic Grid Puzzles

Fiona Shyne[*]
shyne.f@northeastern.edu
Northeastern University
Boston, Massachusetts, USA

Kaylah Facey[*]
facey.k@northeastern.edu
Northeastern University
Boston, Massachusetts, USA

Seth Cooper
se.cooper@northeastern.edu
Northeastern University
Boston, Massachusetts, USA

ABSTRACT

Logic grid puzzles are a popular variety of pen-and-paper puzzles. In logic grid puzzles, players are given natural language hints that they use to mark relationships between entities on a grid. We demonstrate the ability of a Feasible-Infeasible Two-Population (FI-2Pop) genetic algorithm to produce high-quality logic grid puzzles. Hints are constructed using a hand-authored grammar that represents typical types of hints. Infeasible individuals are evolved to approach becoming solvable. Feasible individuals are optimized based on estimated difficulty and hint count. The final evolved puzzles require deductive reasoning skills of the player and were found challenging by the authors of this paper.

CCS CONCEPTS

• **Computing methodologies** → **Genetic algorithms**; • **Applied computing** → *Computer games.*

KEYWORDS

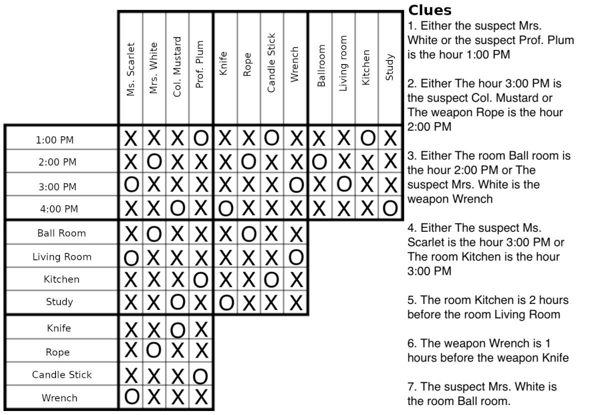
logic puzzles, procedural content generation, FI-2Pop

ACM Reference Format:
Fiona Shyne, Kaylah Facey, and Seth Cooper. 2024. Generating Solvable and Difficult Logic Grid Puzzles. In *Genetic and Evolutionary Computation Conference (GECCO '24 Companion), July 14–18, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3638530.3654337

1 INTRODUCTION

Logic grid puzzles (also called Einstein puzzles, zebra puzzles, or cross logic puzzles) are a popular variety of pen-and-paper constraint-based logic puzzles. Two example puzzles (generated by our system) are shown in Figures 1 and 3. In logic grid puzzles, natural language hints are given to help players complete a grid of relationships between entities.

In this paper, we use a Feasible-Infeasible Two-Population (FI-2Pop) genetic algorithm to generate new logic grid puzzles that are solvable and challenging. We constructed a grammar to represent the hint space that can be used for puzzles of any size. We then use a custom solver to determine if the hint list produces a solvable puzzle. For solvable hint lists, we can evaluate their difficulty based on how many loops the solver takes to solve the puzzle.



	Ms. Scarlet	Mrs. White	Col. Mustard	Prof. Plum	Knife	Rope	Candle Stick	Wrench	Ballroom	Living room	Kitchen	Study
1:00 PM	X	X	X	O	X	X	O	X	X	X	O	X
2:00 PM	X	O	X	X	X	O	X	X	O	X	X	X
3:00 PM	O	X	X	X	X	X	X	O	X	O	X	X
4:00 PM	X	X	O	X	O	X	X	X	X	X	X	O
Ball Room	X	O	X	X	X	O	X	X				
Living Room	O	X	X	X	X	X	X	O				
Kitchen	X	X	X	O	X	X	O	X				
Study	X	X	O	X	O	X	X	X				
Knife	X	X	O	X								
Rope	X	O	X	X								
Candle Stick	X	X	X	O								
Wrench	O	X	X	X								

Clues
1. Either the suspect Mrs. White or the suspect Prof. Plum is the hour 1:00 PM

2. Either The hour 3:00 PM is the suspect Col. Mustard or The weapon Rope is the hour 2:00 PM

3. Either The room Ball room is the hour 2:00 PM or The suspect Mrs. White is the weapon Wrench

4. Either The suspect Ms. Scarlet is the hour 3:00 PM or The room Kitchen is the hour 3:00 PM

5. The room Kitchen is 2 hours before the room Living Room

6. The weapon Wrench is 1 hours before the weapon Knife

7. The suspect Mrs. White is the room Ball room.

Figure 1: Puzzle A (solved). The "0"s represent links between entities and the "X"s represent entities that are not linked.

This paper contributes an evolutionary approach to generating logic grid puzzles; the first, to our knowledge, academic contribution to generating this type of puzzle. Additionally, we contribute a grammar that can be used to produce hints for a given puzzle.

2 RELATED WORK

2.1 Constrained Evolutionary Algorithms

There are several problem spaces where the search space is divided into feasible and infeasible individuals, based on a constraint. While strategies like the death penalty [12] lead to loss of valuable information [17], a fitness function that incorporates feasibility can be difficult to construct [6]. Feasible-Infeasible Two-Population (FI-2Pop) [11] genetic algorithms utilize the fact that optimal solutions often are found on the border of feasibility [23] and have greater genetic variety [10]. In FI-2Pop, two populations are maintained. One population contains only infeasible individuals, which are selected to approach feasibility, and the other contains only feasible individuals, which are selected based on an optimization function. There is no cross-breeding, so genetic information is only shared between the populations when individuals move between them.

FI-2Pop has been used to generate a variety of game content, including video game levels [1, 27], game sprites [13], and dungeons [14, 26]. Scirea [24] use a variant of the FI-2Pop algorithm to generate puzzles for a game about getting a car to its destination while collecting flags. However, this is the first work to apply FI-2Pop to generating a logic grid puzzle.

2.2 Logic Grid Puzzle Solving

This work looks at logic grid puzzles, in which players are given a series of natural language clues and must find links between different entities. The representation of clues has made logic grid

[*] These authors contributed equally to this research.

GECCO '24 Companion, July 14–18, 2024, Melbourne, VIC, Australia

ACM ISBN 979-8-4007-0495-6/24/07
https://doi.org/10.1145/3638530.3654337

Type	Description	Production Rules with examples
is	The two entities are linked	The [cat1] [ent] is the [cat2] [ent] The suspect Ms. Scarlet is the weapon Knife
not	The two entities are not linked	The [cat1] [ent] is not the [cat2] [ent] The suspect Ms. Scarlet is not the weapon Knife
before	The first entity is $m < n - 2$ units smaller than the second entity in the numerical category	The [alph] [ent1] is at least 1 [num] before the [alph] [ent2] The [alph] [ent1] is [int] [num]s before the [alph] [ent2] The suspect Ms. Scarlet is 2 hours before the weapon Knife
or	Either the first or the second entity is linked to the third entity, but not both	Either the [cat1] [ent] or [cat2] [ent] is the [cat3] [ent] Either [cat1] [ent1] or [cat1] [ent2] is the [cat2] [ent] Either the suspect Ms. Scarlet or the suspect Col. Mustard is the weapon Knife
complex or	Either the first **is** statement or the second **is** statement is true, but not both.*	Either [is] or [is] Either the suspect Ms. Scarlet is the weapon Knife or the suspect Col. Mustard is the room Living Room

Table 1: The production rules for producing hints as discussed in subsection 3.2. Each line represents a different production rule. *Is statements were chosen to avoid confusion, even though any statement could be used.

puzzles of particular interest to NLP research. Several works have translated the natural language clues into computer logic using Pairwise Markov Networks [18], Blackburn and Bos frameworks [2], and DistilBERT models [9]. Most recently Large Language Models have been used [8]. Often these techniques transform the natural language clues into a discrete set of pre-defined clue types [4, 9]. The puzzles can then be represented with Answer Set Programming [8, 18], IDP knowledge systems [2], or prolog logic [9].

2.3 Pen and Paper Puzzle Generation

A wide variety of techniques have been used to generate new puzzles [3]. Sudoku [5, 15, 16, 21], Nonograms [20], and Shinro [19] are the most common types of pen-and-paper logic puzzles that have been generated in academic works. Techniques for generation are often search-based strategies such as genetic algorithms [15, 19], hill-climbing [5], or depth-first search [21]. Other techniques such as grammars [25], rule-based algorithms [16], or image processing [20] have also been used. Difficulty is often determined by an automatic solver either through total processing time [5, 15] or by the types of moves solvers make [19]. Difficulty has also been determined by the properties of the puzzles themselves [16, 20].

3 SYSTEM OVERVIEW

Our system produces new logic puzzles, given a set of entities to be linked. Each category has an equal number of entities made up of either categorical (e.g. names) or numerical (e.g. times) values. The source code, along with all generated puzzles are available on GitHub [1].

3.1 Logic Puzzles

Logic grid puzzles rely on players deducing which entities are linked by marking them on a grid (see Figures 1 and 3). In these puzzles players are given a list of natural language hints that describe relationships between entities across different categories. Players must use these hints, along with logical reasoning, to determine which entities are linked to each other in each category.

3.2 Hint Grammar

There are several types of hints provided by grid logic puzzles. We represent these hints with a grammar, shown in Table 1, that describes which types of entities and categories can be used to construct each hint type. Each hint type also has a distinct logical interpretation. While this is not necessarily all-encompassing of possible hints, they represent the most commonly occurring types of hints. The grammar is written such that it can be used for any puzzle.

There is a two-step process for randomly generating a hint given a set of categories and entities. First, production rules are randomly selected until only terminals are present. Then the terminals are replaced with random values based on the type of terminal that is present. "Before" hints require a numerical category, so for puzzles with only categorical categories these hints have to be regenerated. The resulting hints are human-readable but not necessarily grammatically correct and may require minor editing before publication. However, the hint examples in this paper are not edited.

3.3 Solver

Given a hint in this grammar, our system can return an updated grid where the logic of the hints, along with any deductive reasoning, is applied. The puzzle solver (algorithm 1) is a hand-coded function designed to mimic the way a human might solve a logic grid puzzle. Prior work has shown that players often believe these types of puzzles should not require making guesses and backtracking [7]; therefore our solver only makes moves it is confident in.

A list of hints is considered solvable if 1) the resulting grid has no empty spaces, and 2) the resulting grid is a valid solution (there are no logical contradictions). Solvability is determined only by the resulting puzzle grid, which means a hint may have a contradiction that is found after the puzzle is solved. In practice this is rare.

[1] https://github.com/flaneuseh/logic_puzzles/releases/tag/v1.0.0

Algorithm 1: Solve Puzzle

```
Input   : A list of hints $H$, and a puzzle $P$
create list $l$ containing all hints in $H$
while $l$ is not empty, $P$ was modified last loop, and no
  contradictions have been found do
    create an empty list $l_{temp}$
    for $h$ in $l$ do
        attempt to update $P$ using the logic of $h$
        if $h$ created a contradiction then
            set $P$ to be invalid
            break
        else
            if $P$ was changed then
                update $P$ based on deductive reasoning
            end if
            if $h$ is not completely satisfied by $P$ then
                add $h$ to $l_{temp}$
            end if
    end for
    set $l$ to $l_{temp}$
end while
return $P$
```

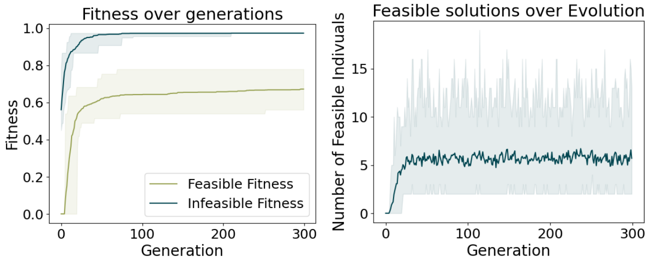


Figure 2: Fitness by generation (left) and the average size of the feasible population out of 50 (right), over 300 generations. The solid line represents the average across 30 trials, whereas the shaded area shows the minimum and maximum values across trials.

3.4 Generating Hint Lists

Individuals in our population are represented as a list of hints as described in subsection 3.2.

We use a Feasible-Infeasible Two-Population (FI-2Pop) genetic algorithm to produce solvable and difficult hint lists. In this kind of genetic algorithm, two populations are maintained, an infeasible population where individuals are selected to approach feasibility (solvable puzzles), and a feasible population where individuals are selected for optimization (difficulty and hint count).

Initial Population The initial individuals are made up of hint lists with between 1 and 5 hints randomly generated using the hint grammar described by subsection 3.2. Each individual is then evaluated for feasibility and sorted into the appropriate population.

Mutation There are two mutation operations, addition and deletion. If the addition mutation is selected, a new random hint is added to the hint list. If the deletion mutation is selected, a random hint is removed from the hint list. If the hint list has fewer than 20 hints, the addition and deletion mutations are equally likely. However, for hint lists of 20 or larger, only the deletion mutation is applied.

Cross Over During cross-over, the hint lists from the two parents are combined and shuffled. The combined hints are then equally distributed between children.

Infeasible Fitness For infeasible individuals, fitness is determined by how close it is to becoming solvable. Two factors contribute to a puzzle being solvable: completion, whether all grid cells can be filled; and validity, whether the solution conforms to the rules of the puzzle. For logic grid puzzles, validity means that there is exactly one "O" per row and column per subgrid and that there are no violations (ex: A is linked to B and C, but B is not linked to C).

The fitness function is represented with the following equation:

$$0.33(\frac{f}{e+f}) + 0.33(\frac{c}{r}) + 0.33(\begin{cases} 0 & \text{if } v \geq 10 \\ 1-(v/10) & \text{otherwise} \end{cases})$$

Where

- f is the number of filled ("X" or "O") cells
- e is the number of empty cells
- c is the number of rows in a subgrid with exactly one "O"
- r is the total number of rows in all subgrids
- v is the number of logical violations

Feasible Fitness Feasible individuals are optimized for high difficulty (more solver loops) and low hint count as represented by the following equation:

$$0.5(1-\frac{h}{10}) + 0.5(\begin{cases} 1 & \text{if } l \geq 7 \\ l/7 & \text{otherwise} \end{cases})$$

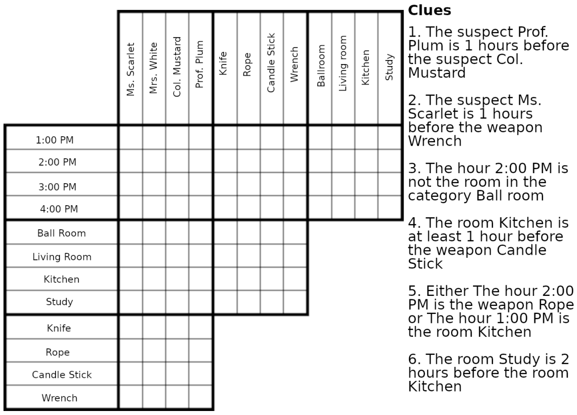
where

- h is the hint count
- l is the solver's number of outer loops through the hint list

4 GENERATING PUZZLES

Logic puzzles are generated with 4 categories (suspects, rooms, weapons, times), each containing 4 entities based on the board game Clue [22]. The genetic algorithm was run for 30 trials of 300 generations each, to produce 30 puzzles. We used a population size of 50, a mutation rate of 0.8, a cross-over rate of 0.3, and 2 elites being saved in each population.

The average fitness and size of the feasible population are shown in Figure 2. The first feasible individual was found on average by Generation 11, with the fastest being found in Generation 6 and the slowest being found in Generation 21. Once found, the number of feasible individuals averaged around 10% of the total population. Infeasible individuals had an average top fitness of 0.97 by Generation 300, while feasible individuals had an average top fitness of 0.67 (note that 1 is impossible for the feasible fitness). The hint lists have an average of 7.0 hints, with the largest having 9 hints and the smallest having 6 hints. The difficulty is an average of 4.87 solver loops, with a minimum of 4 and a maximum of 6. The most common kind of hint generated was "before" (average of 2.7



Clues

1. The suspect Prof. Plum is 1 hours before the suspect Col. Mustard

2. The suspect Ms. Scarlet is 1 hours before the weapon Wrench

3. The hour 2:00 PM is not the room in the category Ball room

4. The room Kitchen is at least 1 hour before the weapon Candle Stick

5. Either The hour 2:00 PM is the weapon Rope or The hour 1:00 PM is the room Kitchen

6. The room Study is 2 hours before the room Kitchen

Figure 3: Puzzle B (blank).

per puzzle), followed by "compound or" (1.7), "simple or" (1.47), "is" (0.87), and "not" (0.27).

4.1 Qualitative Analysis

The two first authors solved two generated puzzles, reffered to as A (Figure 1) and B (Figure 3).

The authors considered Puzzle A to have moderate difficulty and be more challenging than the average puzzle commercially available. Some of the deductions required were not intuitive, requiring careful evaluation. However, we considered the process fair, and no guesswork or backtracking were used to solve it. Puzzle B was found to be easier than Puzzle A, but did not include any redundancies and all deductions were fair.

5 CONCLUSION

This paper presents the first, to our knowledge, academic work in algorithmically generating logic grid puzzles via genetic algorithms. Further, we have demonstrated that a relatively simple two-population genetic algorithm was able to find solvable puzzles quickly. Even in the slowest trial, the first feasible solution was found by the 21st generation. The generated puzzles, when tested by the authors of this paper, were sufficiently difficult to be engaging. Further, selecting individuals with small hint counts seems to be sufficient to remove redundancies in hints.

However, high puzzle difficulty should not necessarily be the goal. It may be of interest to explore generating puzzles with a variety of difficulties, such as with MAP-Elites or other quality diversity algorithms. The notion of difficulty could also be extended beyond the number of deductions needed and could explore the types of deductions instead, or be based on other types of solvers. A user study should also be done to further evaluate puzzle quality.

REFERENCES

[1] Michael Cerny Green, Luvneesh Mugrai, Ahmed Khalifa, and Julian Togelius. 2020. Mario Level Generation From Mechanics Using Scene Stitching. In *2020 IEEE Conference on Games (CoG)* (Online). IEEE, 49–56.

[2] Jens Claes, Bart Bogaerts, Rocsildes Canoy, and Tias Guns. 2019. User-Oriented Solving and Explaining of Natural Language Logic Grid Puzzles. In *The Third Workshop on Progress Towards the Holy Grail.* Association for Constraint Programming.

[3] Barbara De Kegel and Mads Haahr. 2019. Procedural Puzzle Generation: A Survey. *IEEE Transactions on Games* 12, 1 (May 2019), 21–40.

[4] Guillaume Escamocher and Barry O'Sullivan. 2019. Solving Logic Grid Puzzles with an Algorithm that Imitates Human Behavior. (2019). arXiv:1910.06636

[5] Bahare Fatemi, Seyed Mehran Kazemi, and Nazanin Mehrasa. 2014. Rating and Generating Sudoku Puzzles Based On Constraint Satisfaction Problems. *International Journal of Computer and Information Engineering* 8, 10 (Sept. 2014), 1816–1821.

[6] Mitsuo Gen and Runwei Cheng. 1996. A survey of penalty techniques in genetic algorithms. In *Proceedings of IEEE international conference on evolutionary computation.* IEEE, 804–809.

[7] Ruth Hoffmann, Xu Zhu, Özgür Akgün, and Miguel A Nacenta. 2022. Understanding How People Approach Constraint Modelling and Solving. In *28th International Conference on Principles and Practice of Constraint Programming (CP 2022).* Association for Constraint Programming, 28:1–28:18.

[8] Adam Ishay, Zhun Yang, and Joohyung Lee. 2023. Leveraging Large Language Models to Generate Answer Set Programs. In *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning.* Principles of Knowledge Representation and Reasoning, 374–383.

[9] Elgun Jabrayilzade and Selma Tekir. 2020. LGPSolver - Solving Logic Grid Puzzles Automatically. In *Findings of the Association for Computational Linguistics: EMNLP 2020.* Association for Computational Linguistics, 1118–1123.

[10] Steven Orla Kimbrough, Gary J. Koehler, Ming Lu, and David Harlan Wood. 2008. On a Feasible–Infeasible Two-Population (FI-2Pop) genetic algorithm for constrained optimization: Distance tracing and no free lunch. *European Journal of Operational Research* 190, 2 (Oct. 2008), 310–327.

[11] Steven Orla Kimbrough, Ming Lu, and David Harlan Wood. 2004. Exploring the Evolutionary Details of a Feasible-Infeasible Two-Population GA. In *Parallel Problem Solving from Nature - PPSN VIII.* Springer, 292–301.

[12] Oliver Kramer and Hans-Paul Schwefel. 2006. On three new approaches to handle constraints within evolution strategies. *Natural Computing* 5, 4 (Nov. 2006), 363–385.

[13] Antonios Liapis. 2016. Exploring the Visual Styles of Arcade Game Assets. In *International Conference on Evolutionary and Biologically Inspired Music and Art.* Springer, 92–109.

[14] Antonios Liapis. 2017. Multi-segment evolution of dungeon game levels. In *Proceedings of the Genetic and Evolutionary Computation Conference.* ACM, 203–210.

[15] Timo Mantere and Janne Koljonen. 2007. Solving, rating and generating Sudoku puzzles with GA. In *2007 IEEE Congress on Evolutionary Computation.* IEEE, 1382–1389.

[16] Jixian Meng and Xinzhong Lu. 2011. The Design of the Algorithm of Creating Sudoku Puzzle. In *Advances in Swarm Intelligence.* Springer, 427–433.

[17] Zbigniew Michalewicz et al. 1995. Do Not Kill Unfeasible Individuals. In *Proceedings of the Fourth Intelligent Information Systems Workshop.* 110–123.

[18] Arindam Mitra and Chitta Baral. 2015. Learning to Automatically Solve Logic Grid Puzzles. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing.* Association for Computational Linguistics, 1023–1033.

[19] David Oranchak. 2010. Evolutionary Algorithm for Generation of Entertaining Shinro Logic Puzzles. In *Applications of Evolutionary Computation.* Springer, 181–190.

[20] Emilio G. Ortiz-García, Sancho Salcedo-Sanz, Jose M. Leiva-Murillo, Angel M. Pérez-Bellido, and José A. Portilla-Figueras. 2007. Automated generation and visualization of picture-logic puzzles. *Computers & Graphics* 31, 5 (Oct. 2007), 750–760.

[21] Barry O'Sullivan. 2007. Generating and Solving Logic Puzzles through Constraint Satisfaction. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence.* Association for the Advancement of Artificial Intelligence, 1974–1975.

[22] Anthony E. Pratt. 1949. Clue. [Board Game].

[23] Marc Schoenauer and Zbigniew Michalewicz. 1996. Evolutionary computation at the edge of feasibility. In *International Conference on Parallel Problem Solving from Nature.* Springer, 245–254.

[24] Marco Scirea. 2020. Adaptive Puzzle Generation for Computational Thinking. In *HCI in Games.* Springer, 471–485.

[25] Josep Valls-Vargas, Jichen Zhu, and Santiago Ontañón. 2017. Graph grammar-based controllable generation of puzzles for a learning game about parallel programming. In *Proceedings of the 12th International Conference on the Foundations of Digital Games.* ACM, 1–10.

[26] Breno M. F. Viana, Leonardo T. Pereira, Claudio F. M. Toledo, Selan R.f dos Santos, and Silvia M. D. M. Maia. 2022. Feasible–Infeasible Two-Population Genetic Algorithm to evolve dungeon levels with dependencies in barrier mechanics. *Applied Soft Computing* 119, C (April 2022), 108586.

[27] Adeel Zafar, Hasan Mujtaba, and Mirza Omer Beg. 2019. Search-based procedural content generation for GVG-LG. *Applied Soft Computing* 86, 3 (Nov. 2019), 105909.