# The Bullets Puzzle: A Paper-and-Pencil Minesweeper

**Todd W. Neller, Hien G. Tran**

Gettysburg College
{tneller, tranhi01}@gettysburg.edu

## Abstract

In this paper, we introduce a technique for AI generation of the Bullets puzzle, a paper-and-pencil variant of Minesweeper. Whereas traditional Minesweeper can be lost due to the need to guess mine or non-mine positions, our puzzle is fully deducible from a minimal clue set. Puzzle generation is based on analysis and optimization of solutions from a human-like reasoning engine that classifies types of deductions. Additionally, we provide insights to subjective puzzle quality, minimal clue sampling trade-offs, and optimal bullet density.

## Introduction

A Bullets puzzle (Figure 1) is defined as (1) a grid with a subset of grid cells each containing a bullet, and (2) a subset of clue cells, i.e. non-bullet grid cells that are known to the puzzler and indicate the number of bullets orthogonally or diagonally adjacent to each clue cell. The puzzler initially knows only the clue cells (shown in Figure 1) and the number of hidden bullets. The solution, i.e. the positions of all bullets, can be fully deduced from the initial information. In the Figure 2 solution, bullets are indicated by circles, and other non-bullet, non-clue cells are marked with diagonal slashes.

Thematically, we imagine the clues as metal detector readings for one searching a field for bullets from the Civil War[1].

In this paper, we explore the use of AI techniques in the improved design and generation of this paper-and-pencil variant of the classic computer puzzle game Minesweeper. This variant was originally designed by the first author on December 14th, 2010. We begin with a history and description of Minesweeper, including a discussion of what we consider a design flaw: that randomly generated puzzles may not be fully deducible and may require guessing that results in a loss. Accordingly, we survey prior work towards guess-free Minesweeper variants.

Following this, we describe the knowledge representation and reasoning at the core of our computations using at-least constraints. We outline a human-like deduction system that produces an explanation of a human-like sequence of solution steps that we used to form an objective function for our subjective tastes in Minesweeper reasoning. We discuss how a single puzzle is generated, and how we employ stochastic local search to generate puzzles optimized for our subjective tastes.

Next, we share and interpret data regarding the choice for how many minimal clue subsets to sample for a given bullet configuration, as well as which bullet density we recommend for best puzzles.

Finally, we share future possible work in refined reasoning steps and non-paper-and-pencil Minesweeper, and summarize our conclusions.

## Related Work

Since this is a paper-and-pencil variant of Minesweeper, we survey Minesweeper's history, what we consider its main design flaw, related work to address that design flaw, and known computational complexity results.

In Windows® Minesweeper (Figure 3), like the Bullets Puzzle, one has a known number of mines that must be found. However, there is no initial information. A player may right-click to mark a position with a flag (indicating belief that a mine is there) or may left-click a position to reveal a clue or mine. If a mine is revealed, the player loses. Thus, the first click may reveal clue information dynamically, or end the game with the revelation of a mine. If a clue revealed would be a "0" clue (i.e. empty position with no adjacent mines), trivial revelations of adjacent cells are performed until all contiguous "0" clues are revealed. While many puzzles may be solved through logic alone, mines are randomly distributed and may present situations where the player must guess a safe position and reveal it. For example, in the upper-left corner of Figure 3, the last mine could be in either unmarked square to satisfy constraints. However, the player must choose one to reveal and has a 0.5 probability of losing despite having reasoned perfectly through all given information.

In logic puzzles like Minesweeper, it is generally presumed that logic alone is sufficient to solve the puzzle. From this perspective, the need to guess initially or during the solving process, admitting possibility of loss through no fault of the player, is a design flaw in Minesweeper.



[1]This is illegal on many federally recognized Civil War battle sites.

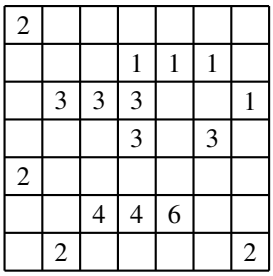


Figure 1: Example Bullets puzzle with 16 bullets

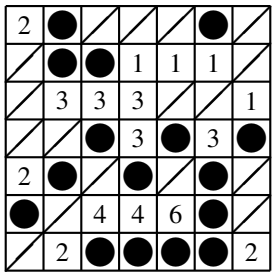


Figure 2: Solution to example Bullets puzzle

Figure 3: Minesweeper situation where a guess is necessary.

When discussing the lineage of Minesweeper puzzle games, writers point to a progression from Jerimac Ratliff's 1979 "Cube" (Ahl 1979) where one guesses a mine-free path through $3 \times 3 \times 3$ cube vertices with no information for reasoning, through Ian Andrew's 1983 "Mined-Out" (Mined-Out) where number clues were first added to a 2D grid traversal challenge, to the July 1990 "Mine 2.9" (Mine 2.9), a beta version of Windows® Minesweeper.

Efforts to produce guess-free computer versions of Minesweeper include those of Paweł Marczewski (Marczewski 2021), Christian Czepluch (Czepluch 2021), Simon Tatham (Tatham 2021), and many others. Most do not document how they create guess-free puzzles, but Marczewski and Czepluch both approach the problem by not committing to a fixed mine configuration unless forced to by logic. Marczewski writes, "If you try to guess, the game will always choose the worst scenario. Except when you are forced to guess (there are no safe cells anywhere). Then, guessing is completely safe." This is reminiscent of the Nifty Assignment "Evil Hangman" (Schwarz 2011) where there is no initial commitment to a solution and the set of possible solutions narrows according to player guesses.

Richard Kaye has proven the Minesweeper consistency problem to be NP-complete (Kaye 2000b) and infinite Minesweeper to be Turing-complete (Kaye 2000a). For consistent boards, Minesweeper inference is co-NP-complete (Scott, Stege, and van Rooij 2011). However, like other NP-Complete constraint satisfaction problems such as 3-SAT (Gent and Walsh 1994), Dempsey and Guinn show a similar phase transition in computational complexity where a mine density over 20% of cells leads to exponential growth in Minesweeper reasoning (Dempsey and Guinn 2020).

The earliest published paper-and-pencil variant of Minesweeper we have found is that of Dan Moore's designs (Moore and Vallely 2012), and other similar books have been published since. Moore's puzzles do not provide the number of mines as a constraint, but do guarantee unique, fully-deducible solutions, usually involving 21–23 mines hidden in a $10 \times 10$ grid. The novelty of our contribution is in our detailed discussion of AI-assisted design of high quality puzzles of this type.

## At-Least Constraint Reasoning

In representing knowledge about a Bullets puzzle, we assign one variable per grid cell to indicate whether there is a bullet (true) or no bullet (false). A *literal* $c$ or $\neg c$ means that a bullet is or is not in the cell corresponding to variable $c$. A literal is *satisfiable* if that literal is true in a possible world consistent with known facts.

Consider a cell with clue 4 and no knowledge of the 8 adjacent cells. In propositional logic, one might represent this as a disjunction of $\binom{8}{4} = 70$ possible conjunctions of literals that represent all the ways the cells may be consistent with that clue constraint. While we could represent our knowledge in that way, this would lead to a bloated representation and inefficient reasoning. Furthermore, this form of expression for the fact that there are exactly 16 bullets in the 49 cells of Figure 1 would require $\binom{49}{16} \approx 3.348109 \times 10^{12}$ conjunctions.

We instead opt to represent our puzzle knowledge with at-least constraints that consist of (1) a number $k$, (2) a list $l$ of variables, and (3) a value $v$ (true/false), such that the at-least constraint expresses that "at least $k$ of the variables in $l$ must have value $v$.

Consider an interior clue 3 which expresses that there are mines in *exactly* 3 of its 8 adjacent cells. This may be equivalently expressed as the two at-least constraints: "At least 3 adjacent cell variables are true." and "At least 5 adjacent cell variables are false." We number the cell variable from 0 through $|C| - 1$, where $C$ is the set of all cells, and $|C|$ is the number of cells, in left-to-right, top-to-bottom western reading order as in Figure 5. Thus, the left-most 3 clue in Figure 1 would correspond to $c_{15}$ having value false and have associated at-least constraints $\mathrm{atLeast}(3, \{c_7, c_8, c_9, c_{14}, c_{16}, c_{21}, c_{22}, c_{23}\}, \mathrm{true})$ and $\mathrm{atLeast}(5, \{c_7, c_8, c_9, c_{14}, c_{16}, c_{21}, c_{22}, c_{23}\}, \mathrm{false})$.

For reasoning about such at-least constraints, we modified Donald Knuth's Dancing Links (DLX) algorithm (Knuth 2011) such that each row in the doubly-linked matrix corresponds to an at-least constraint and each column corresponds to a variable. This reasoner is at the core of type 1 and type 2 puzzle reasoning steps as defined in the next section.

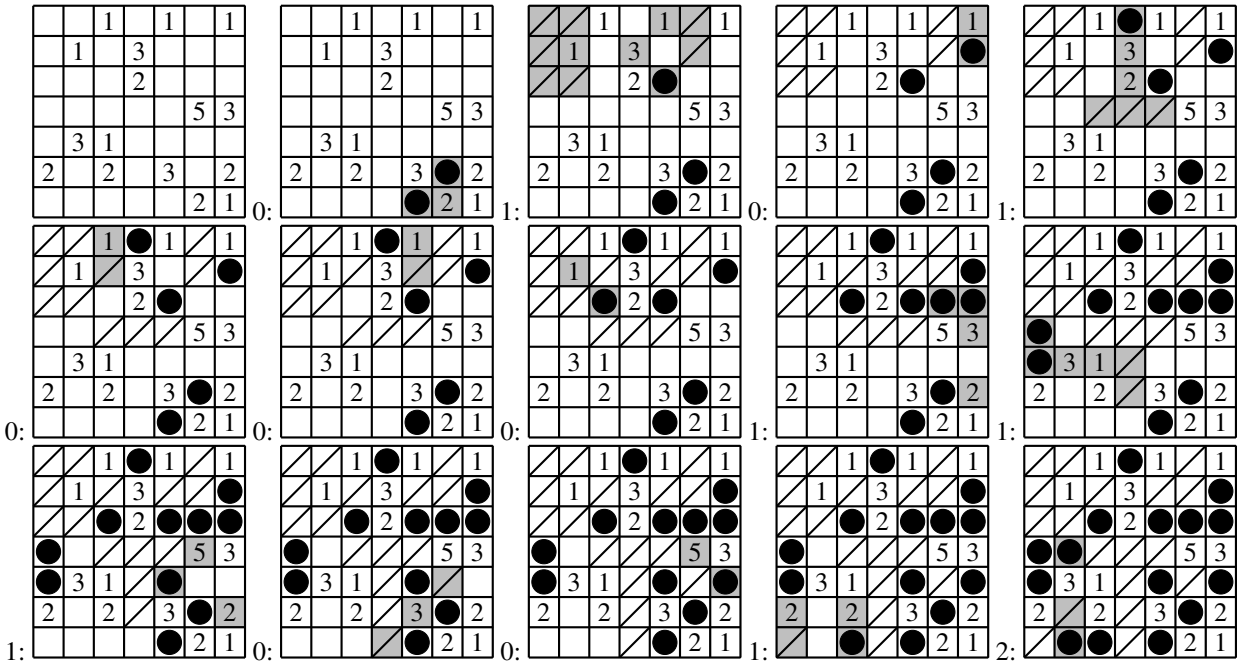

Figure 4: Example human-reasoning solution for a 7 × 7, 15-bullet puzzle with an energy of -11.0 that was generated with 10 minimal clue set samples over 1000 iterations of hill descent. After the initial puzzle board, each successive board is preceded by "$n$:" where $n$ is the deduction type. Highlighted in gray are both the minimal set of clues necessary for the deduction and the cells that were deduced. Clue cells and other cells where only bullet/no-bullet knowledge is used are not highlighted.

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| 28 | 29 | 30 | 31 | 32 | 33 | 34 |
| 35 | 36 | 37 | 38 | 39 | 40 | 41 |
| 42 | 43 | 44 | 45 | 46 | 47 | 48 |

Figure 5: Reference grid for the cell indexing of a 7 × 7 puzzle.

## Human Reasoning Steps

As a simple approach to puzzle generation, one can often optimize puzzle design for greatest computational solution time and expect puzzles to be generally difficult. However, the most computationally difficult puzzles are not always the most subjectively enjoyable puzzles to solve. Most constraint satisfaction algorithms we create employ some form of brute-force, depth-first search which bears no resemblance to human reasoning. We do not maintain deep call stacks for hypotheses, nor would we have fun simulating such on paper.

Our approach is then to create a solver that solves these puzzles with a human-like approach, study the puzzles with their human-like solutions, learn what we can about the characteristics of solutions we enjoy, and then optimize designs for those characteristics.

For this work, we categorize reasoning steps into three types:

**Type 0** Deductions involving a single clue

**Type 1** Deductions involving multiple clues without knowledge of the total number of bullets

**Type 2** Deductions involving multiple clues with knowledge of the total number of bullets

Our solver always seeks a deductive step of the lowest number type, preferring simplest-type next steps in the solution. Thus, a type 1 deduction is only attempted when there are no remaining type 0 deductions. Necessarily, a type 2 deduction will fully solve our puzzles and will only occur as a final step in our solution.

What follows is a description of the algorithm for each type. Two terms are important to understand for these descriptions: (1) *Hidden* cells are those that are initially empty, i.e. non-clue cells devoid of information. (2) *Unknown* cells are hidden cells where we have not yet deduced whether or not the cell has a bullet. For example, we can deduce that a cell has no bullet (becoming known) without knowing its associated number of adjacent bullets (remaining hidden). This is a necessary difference between this paper-and-pencil

puzzle and the dynamic Minesweeper solving experience.

### Type 0: Single Cell Clue

For type 0 reasoning, we need not employ any complex reasoning engine. We simply iterate through our clues that have adjacent unknown cells, looking for one for which all unknowns must have all or no bullets in order to satisfy the clue's constraints. Finding one, we mark all adjacent unknowns as known and store information of this type 0 step.

### Type 1: Multiple Cell Clues

This is the most computationally complex step of our solver. We will illustrate this step using the second-to-last type 1 deduction of the example solution in Figure 4. Contrast the second-to-last solution state with the state before to see the relevant clues and deduced cells of the reasoning step.

We first identify the subset of clues with adjacent unknown cells. With zero-based indexing left-to-right and top-to-bottom as in Figure 5, such clues are the "3" clue at cell 29, the "1" at 30, the "2" at 35, and the "2" at 37. For clue "2" at cell 35, we can express relevant facts as follows:

- "At least 2 of the cells 28, 29, 36, 42, and 43 have bullets."
- "At least 3 of the cells 28, 29, 36, 42, and 43 do not have bullets."
- "Cell 28 has a bullet." (At least 1 of cell 28 has a bullet.)
- "Cell 29 does not have a bullet." (At least 1 of cell 29 has no bullet.)

From these, the solver will deduce that there is exactly 1 bullet in adjacent unknown cells 36, 42, and 43.

We initially construct a solver that attempts to find a deduction *using only this subset of cell variables and the bullet/no-bullet knowledge of those cells adjacent to them*. If there is a type 1 deduction, we choose the possible deduction with minimal index, and test each of the clues sequentially for whether they can be removed from the clue list while still allowing that deduction, retaining or discarding each clue accordingly. We thus compute a *minimal* clue subset necessary for that deduction. In our second-to-last reasoning step of Figure 4, the least-index deduction is that there is no bullet at cell 42, and that both "2" clues at cells 35 and 37 are the minimal clue subset needed to deduce that fact.

Finally, using the minimal clue subset, we test whether further deductions are possible in other unknown cells from the minimal clue subset. Continuing with our example, we find that both "2" clues allow us to further deduce that there must be a bullet at cell 44. Thus all type 1 steps have a minimal clue subset and a resulting deduction set, the information of which are stored. These cells are highlighted with shading in Figure 4.

As an implementation detail, for efficient, minimal reasoning around subsets of cells, we wrap our at-least reasoner in an object that creates a mapping to an alternate cell numbers for only the variables involved.

### Type 2: Multiple Cell Clues and Number of Bullets

Assuming that there is a solution for the given puzzle, a type 2 step is guaranteed to deduce all remaining bullet positions and thus all remaining non-bullet positions. Given that we only apply our solver to puzzles with known solutions, this final solution step asserts the only possible configuration of remaining bullets, marks all cells as known, and stores information of this type 2 step.

## Single Puzzle Generation

When initially creating a single puzzle to optimize that has $n \times n$ cells with $b$ bullets, we create a list of cell indices and shuffle, with the first $b$ entries having the indices of the puzzle's bullets. Data concerning the bullets, the hidden cells, and the unknown states of cells are represented as single-dimensional arrays with row-major ordering of cells. For non-bullet cells, the number of adjacent bullets are computed (as potential clues), and we initially assume all clues are unhidden.

Next, we compute a random minimal clue subset. Taking non-bullet cell indices, we shuffle them and iterate through each. For each potential clue, we hide the clue and create a reasoner to check whether we can still deduce all bullet positions. If not, we unhide that clue cell and it is confirmed as a necessary clue in our minimal clue subset. Once we have iterated through all clues, those remaining are a minimal clue subset; hiding any one would leave us unable to solve the puzzle.

## Puzzle Optimization

We optimize a puzzle by treating bullet placement as a combinatorial optimization problem and applying simple stochastic local search in the form of hill descent on an energy (i.e. objective) function that is a numeric measure of puzzle badness according to our subjective puzzle tastes as detailed below.

The step function moves one bullet to a different non-bullet position by swapping a random pair of indices from the first $b$ entries and the last $n^2 - b$ entries in the aforementioned bullet-cell/non-bullet-cell list. The new state results in new clues, new sampling of minimal clue subsets, and new energy computations. If the minimal sampled energy of the new puzzle state is greater (i.e. worse) than the energy of the previous puzzle state, we undo the change, restoring the previous state. Otherwise, we retain the new state.

The energy of a given puzzle is defined as

$$(0 \times n_0) + (-1.5 \times n_1) + (n_2 \times (n_2 - 3))$$

where $n_i$ is the number of type i deductions in our human-like solution. This expresses that it is generally good to have more type 1 deductions whereas type 0 deductions are neutral. Also, 1-2 type 2 bullet deductions are desirable, but too many or no type 2 bullet deductions are less satisfying, especially far too many bullet deductions. We choose the -1.5 coefficient for $n_1$ as this coefficient produces the best subjective results for our puzzles in trial-and-error tuning of $n_1$, and we choose the 0 coefficient for $n_0$ because type 0 deductions are regarded by us as neutral. Large numbers of bullets in a type 2 deduction often imply a significant concentration or corner/edge dispersion of bullets. Note that this energy function can be modified according to the subjective preferences of any puzzler.

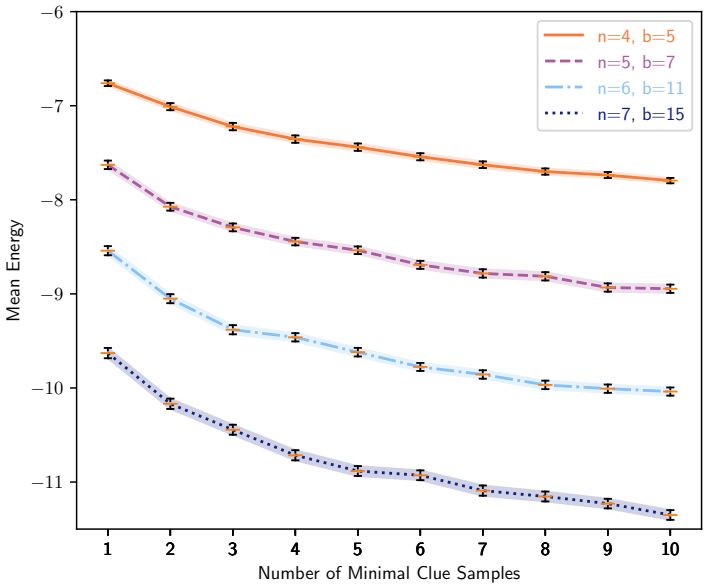


Figure 6: Energy decreases as the number of sampled minimal clue sets increases.

We now turn our attention to a complex design matter for puzzle generation: There are generally many different minimal clue subsets for any given bullet configuration, and it is not efficient to enumerate them for all but the smallest puzzles. We therefore *sample* $s$ minimal clue subsets and use the first one with the minimum energy. In the next section, we share data and conclusions concerning the computational time expense versus quality trade-off faced when computing with $s$ samples.

## Sampling Minimal Clue Sets

If we were to seek to enumerate and evaluate all possible minimal clue subsets, there would be considerable computational expense for each step of our stochastic local search. We therefore opt for a sampling approach. As our sample size increases, we better approximate the optimal energy for the given bullet configuration. However, this comes with a proportionally higher computational time for puzzle generation. In this section, we vary the sample size to measure how this affects puzzle quality.

Figure 6 visualizes our experimental data in generating puzzles with different numbers of sampled minimal clue subsets for different puzzle sizes. Each line represents a series of experiments each of which generated 1000 puzzles with a fixed $n \times n$ grid with $b$ bullets, optimized for 500 iterations of hill descent. Each $b$ was chosen to minimize mean generated puzzle energy with respect to $n$ according to the experiments of the next section. Most optimizations were observed to reach their minimum within 500 iterations. Mean energy 90% confidence intervals are shown for each 1000 puzzles generated.

As one can see, the puzzle quality increases as expected (with decreased energy) when we sample a greater number of minimal clue subsets. Of practical interest is the fact that, for $4 \leq n \leq 6$, much of the quality gain is from the first 3 samples. The decreasing energy slope levels considerably thereafter. For $n = 7$, one could argue that 5 samples might be a better number. As with the elbow method in clustering, there is a subjective judgement for the diminishing returns of increased sampling.

Of course, one's application constraints in computational time and space should inform the best trade-off. Generating a few, small high-quality puzzles would suggest a higher number. Generating larger puzzles with limited computational resources would suggest a smaller number. For our high-computational experimental needs, we have used 3 samples per bullet puzzle configuration for the next experiment.

## Bullet Density

Another design parameter of great interest is the bullet density that is expected to generate puzzles with minimal energy (and thus maximal subjective quality) according to our energy function. If we have too few or too many bullets, we observe more type 0 steps and fewer type 1 steps. Such puzzles lose both complexity and interest. For brute-force constraint satisfaction by computer, (Dempsey and Guinn 2020) showed that a mine density above 20% enters a phase transition for worst-case computational complexity for dynamic, computer Minesweeper. Given that our energy function correlates with *human-like* solution complexity for a static, paper-and-pencil variant of Minesweeper, it is of interest how this compares to computational complexity from a practical puzzle generation perspective.

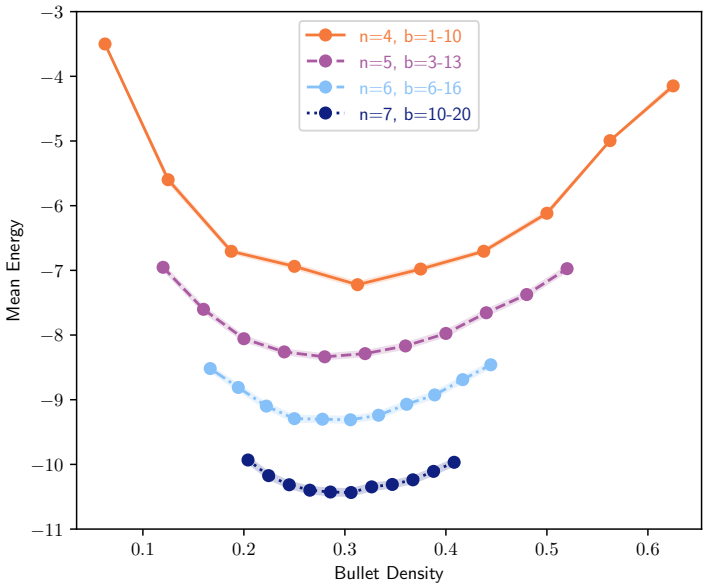


Figure 7: A bullet density of approximately 0.3 appears best for our energy function.

In Figure 7, we plot bullet density versus mean energy. Each line represents a series of experiments on an $n \times n$ grid varying the number of bullets $b$. Bullet density is $\frac{b}{n^2}$, i.e. the fraction of cells containing bullets. For each bullet density in an experimental series, we generate 1000 puzzles with a minimal clue set sample size of 3 for 500 iterations of hill descent. Our narrow 90% confidence intervals are visualized with shaded bands.

What is striking is how consistent the ideal bullet density is as we vary the grid size. A bullet density of approximately 0.3 appears ideal for all grid sizes for our energy function. For Minesweeper, this mine density would correspond to near the top end of the phase transition shown in (Dempsey and Guinn 2020).

Another observation is that the larger the puzzle grid size, the less sensitive the puzzle quality is to the number of bullets. The aforementioned similar published paper-and-pencil Minesweeper variant (Moore and Vallely 2012) had puzzles with densities ranging from 0.21 to 0.23. While such puzzles lack our type 2 deductions, these are likely suboptimal yet reasonable density choices for that variant.

## Future Work

There are two primary ways we would like to extend this work. First, we could extend this work to generate high-quality, no-guess Minesweeper puzzles. The only modification necessary would be to simulate the unhiding of clues and automated 0-clue revelation as one would experience with Minesweeper.

The second extension would be to further refine our Bullets puzzle's classification of human reasoning steps. For example, with $n$ clues, one could have a new reasoning step numbering of type 1 through $n + 1$ where type $n + 1$ involves all clues and knowledge of the number of bullets, and other type numbers reflect the minimum number of clues needed to make the deduction. (Thus, type 1 in the new system would be type 0 in this paper.) This approach essentially differentiates between different possible type 1 reasoning steps of this paper, and we might prefer having some larger, more difficult clue interactions.

Given that one would differentiate between type 1 steps of this paper, one would likely need to modify our type 1 step approach to compute all available type 1 steps and prefer the one with a minimal subset of variables needed, making the solver more human-like still by preferring simplest next steps for a given solution state.

## Conclusions

In this work, we have approached the generation of Bullets puzzles, a paper-and-pencil variant of Minesweeper, by expressing subjective aesthetics of what we most enjoy in such puzzles as an objective energy function to be optimized by stochastic local search. The energy of a given puzzle is defined as

$$(0 \times n_0) + (-1.5 \times n_1) + (n_2 \times (n_2 - 3))$$

where $n_0$ is the number of deductions requiring the knowledge of a single clue and its adjacent cells, $n_1$ is the number of deductions requiring the interaction of multiple clues, and $n_2$ is the number of bullets deduced at the end using the constraint of the known number of bullets in the grid.

Since it would be computationally expensive to enumerate and evaluate all minimal clue subsets, we sampled subsets and experimented to observe the effect of the number

of samples on puzzle quality, observing the expected diminishing returns with increased samples. For highly repetitive experimental generation of puzzles, we found 3 samples to be adequate for good quality, small grid puzzles.

Further, we experimented with varying bullet density and found that, for our objective energy function, a bullet density of approximately 0.3 yielded best puzzles.

## References


Ahl, D. H., ed. 1979. *Basic Computer Games - TRS-80 Edition*. Creative Computing Press. ISBN 0916688402.

Czepluch, C. 2021. Mines-Perfect. https://sourceforge.net/p/mines-perfect/wiki/Home/. Accessed: 2021-09-01.

Dempsey, R.; and Guinn, C. 2020. A Phase Transition in Minesweeper. *CoRR*, abs/2008.04116.

Gent, I. P.; and Walsh, T. 1994. The SAT Phase Transition. In Cohn, A. G., ed., *Proceedings of the Eleventh European Conference on Artificial Intelligence, Amsterdam, The Netherlands, August 8–12, 1994*, 105–109. John Wiley and Sons.

Kaye, R. 2000a. Infinite versions of Minesweeper are Turing complete. http://web.mat.bham.ac.uk/R.W.Kaye/minesw/infmsw.pdf. Accessed: 2021-09-01.

Kaye, R. 2000b. Minesweeper is NP-complete. *Mathematical Intelligencer*, 22(2): 9–15.

Knuth, D. E. 2011. Dancing Links. In *Selected Papers on Fun and Games*, 437–472. Cambridge University Press. ISBN 978-1-57586-584-3.

Marczewski, P. 2021. Kaboom. https://pwmarcz.pl/kaboom/. Accessed: 2021-09-01.

Mine 2.9. 2021. Minesweeper Wiki Windows Minesweeper Version History. http://www.minesweeper.info/wiki/Windows_Minesweeper#Version_History. Accessed: 2021-09-01.

Mined-Out. 2021. Minesweeper Wiki Mined-Out Page. http://www.minesweeper.info/wiki/Mined-Out. Accessed: 2021-09-01.

Moore, D.; and Vallely, J. 2012. *Minesweeper Puzzles: 100 explosive Minesweeper Puzzles*. Clarity Media. ISBN 9781479230433.

Schwarz, K. 2011. Evil Hangman. http://nifty.stanford.edu/2011/schwarz-evil-hangman/. Accessed: 2021-09-01.

Scott, A.; Stege, U.; and van Rooij, I. 2011. Minesweeper may not be NP-complete but is hard nonetheless. *Mathematical Intelligencer*, 33(4): 5–17.

Tatham, S. G. 2021. Mines: from Simon Tatham's Portable Puzzle Collection. https://www.chiark.greenend.org.uk/~sgtatham/puzzles/js/mines.html. Accessed: 2021-09-01.