Minesweeper: A Statistical and Computational Analysis

Andrew Fowler
Andrew Young
Advisor: Matthew Hudelson
Date: 5/1/2004

Honors Thesis
****
PASS WITH DISTINCTION

TO THE UNIVERSITY HONORS COLLEGE:

As faculty advisor for Andrew Fowler, Andrew Young

I have read this paper and find it satisfactory.

Faculty Advisor

5-10-2004

Date

I. Introduction

Every PC user is familiar with the computer game Minesweeper. It has been bundled with every Windows-based computer since the release of Windows 3.1, and countless players have been intrigued by its simple and logical gameplay. Any player can learn to play in minutes, but the simplicity and elegance of the game belie a deep and fascinating mathematical structure. As is often the case in mathematics, a seemingly trivial equivalent problem can become the framework for a legitimate and serious topic of study, even allowing for insights that could not otherwise be noticed. Playing Minesweeper, for example, can be shown to be equivalent to solving certain systems of linear equations. In this regard, looking at the problem of playing Minesweeper is essentially an interesting way of looking at a potentially dry subset of problems in Integer Programming and Linear Algebra.

Minesweeper becomes mathematical when one attempts to develop strategies for winning the game. Because the rules of the game are so simple, the resulting equations can be expressed easily and directly. Only a small leap of insight is required to equate gameplay into an equivalent mathematical system whose solution is comparable to playing and winning the game. Therefore, various strategies of solving these equations result in various strategies in playing Minesweeper, which vary in complexity and effectiveness. Some are trivial and result in a very low winning probability, and some are advanced and involved, requiring sophisticated mathematics and resulting in surprisingly high success rates.

As is the case with most nontrivial games, an exhaustive analysis of gameplay is computationally impractical, most often ridiculously so. Therefore, many of the methods and strategies herein described have been analyzed statistically rather than analytically. However, examining the game from a statistical and probabilistic standpoint is more reasonable because the game itself is inherently random. Even the best players and algorithms often encounter situations which require an element of chance, and the wrong choice can be made regardless of the skill of the player or mathematical strength of the algorithm.

Our research has taken two forms. The first is the design of algorithms for solving the game of Minesweeper, both directly and by means of examining the linear equations involved. The ultimate goal has been to create the *best* algorithm we can, one which wins the game consistently more often than any other for various game parameters. This attempt, however, has led us to the second branch of research, which is making these algorithms computationally viable. Often a simply described gameplay algorithm can be of an order far too vast to solve with any computer, let alone a desktop model. A very effective algorithm is useless if it cannot be undertaken in a reasonable time. We have studied computational complexity and have attempted to make each game strategy as time-efficient as possible using various mathematical techniques. In short, we have tried to design methods that are both effective and fast, making them applicable for both mathematicians and Minesweeper players.

II. Rules and Strategies

Minesweeper is played on an $m \times n$ grid of squares where $m, n \in \mathbb{Z}^+$. The contents of each square are hidden from the player. Before a game begins, the player knows the following:

- Each square conceals an integer or a single bomb.
- A nonnegative integer $b \leq mn$ representing the number of squares on the board containing a bomb.
- Clicking a square containing a bomb causes the player to lose the game.
- Clicking a square not containing a bomb reveals an integer $k \in [0,8]$ representing the number of squares containing bombs that are immediately adjacent to the clicked square, either horizontally, vertically, or diagonally. (Most versions of the game use a blank square rather than revealing the number zero. See Fig. II.1).
- Flagging, or marking, a square does not reveal its contents but distinguishes it from other squares.
- When every square containing a bomb has been flagged and all flags have been removed from squares not containing a bomb, the player wins the game.

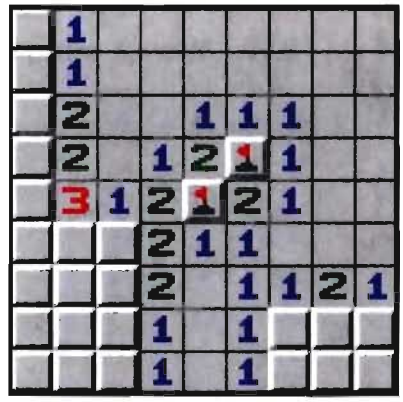


Fig. II.1. A typical Minesweeper board position.

The game itself is played by clicking and flagging squares using the numbers and existing flags on the board to determine where the remaining bombs are located. This process often involves both logical reasoning and an element of chance.

There are many strategies that players can use, but some basic elements are common to all of them.

II.1 Notation and Definitions

- A *square* is the basic unit of the game which conceals a single bomb or number.
- A *board* is the playing grid and its component squares. It can be thought of as an arranged set of squares.
- A *click* is a game action where a chosen square is revealed.
- A *flag* is a special game action whereby a chosen square can be distinguished but not revealed.
- A square is *legitimately flagged* if it is flagged and been determined to be a bomb.
- A square is *unknown* if it is neither clicked nor legitimately flagged.
- Two squares are *adjacent* if they share a corner or side.
- Define $A_{m,n,b}$ to be the set of all $m \times n$ boards with b bombs.
- For a board $B \in A_{m,n,b}$ define $B_{i,j}$ to be the square at coordinates (i, j) on B oriented in standard matrix row-column positioning, where $1 \le i \le m, 1 \le j \le n,$ and $i, j \in \mathbb{Z}$.

- Given a board B and a square $B_{i,j} \in B$, $K\left(B_{i,j}\right)$ is the contents of $B_{i,j}$.
- Given a board B and a square $B_{i,j} \in B$, denote $R_{i,j}^{B} \in B$ as the set of all squares adjacent to $B_{i,j}$.

II.2 Basic Strategy

Theorem 1

Given a board B and square $B_{i,j}$, if the number of unrevealed squares in $R_{i,j}^{B}$ is equal to $K\left(B_{i,j}\right)$ (minus the number of legitimately flagged squares in $R_{i,j}^{B}$), then all of those unrevealed squares contain a bomb.

Proof

Suppose not. Then there exists an unrevealed square in $R_{i,j}^{B}$ which does not contain a bomb. However, this would make $K\left(B_{i,j}\right)$ greater than the total number of flagged and unrevealed bombs in $R_{i,j}^{B}$. This is a contradiction by definition of $K\left(B_{i,j}\right)$.

Theorem 2

Given a board B and square $B_{i,j}$, if the number of legitimately flagged squares surrounding $B_{i,j}$ is equal to $K\left(B_{i,j}\right)$, every unrevealed square in $R_{i,j}^{B}$ does not contain a bomb.

Proof

Suppose not. Then there exists an unrevealed square in $R_{i,j}^{B}$ that contains a bomb. However, this would make the number of bombs in $R_{i,j}^{B}$ greater than $K\left(B_{i,j}\right)$, a contradiction.

These two theorems used in conjunction are the basis for the beginning player's strategy in playing Minesweeper. We will call it Basic Strategy (or BS), and it proceeds as follows:

A.

- Click a random square in the set of unrevealed squares on the board.
- If a bomb is revealed, the game is over.
- If a bomb is not revealed, go to step B.

B.

- Apply Theorem 1 to all revealed numbered squares on the board. If the conditions are satisfied, flag the appropriate squares.
- Apply Theorem 2 to all revealed numbered squares on the board. If the conditions are satisfied, click the appropriate squares.
- If either theorem resulted in squares being flagged or clicked, repeat step B.
- If neither theorem resulted in squares being flagged or clicked, go to step A.

Here is an example of how BS can be used (granted, most players do not think of gameplay in terms of theorems, but almost every beginning player implicitly understands this strategy):

At the beginning of the game, step A is applied and the following board is revealed after clicking in the upper-right corner (note, the Microsoft version of the game will automatically and recursively apply Theorem 2 to revealed zeros and often reveal much of the board).

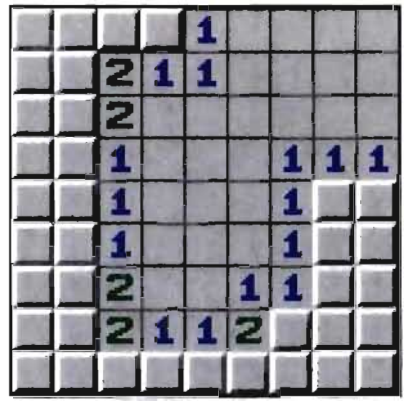


Fig. II.2 Game position after first click.

Now we go to step B. The squares labeled *c, d,* and *e* contain a bomb by Theorem 1. So they may be flagged:

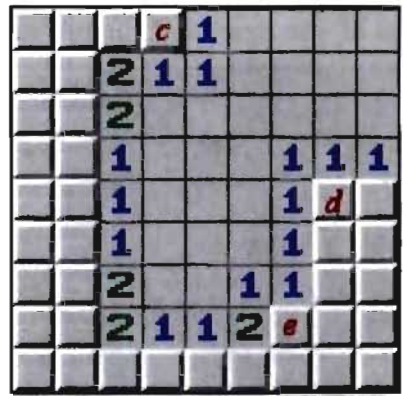


Fig. II.3 Theorem 1 gives bomb locations.

Now we apply Theorem 2. Because of the newly flagged squares, the squares labeled *f, g, h, i,* and *j* do not contain bombs, and they may be clicked safely.

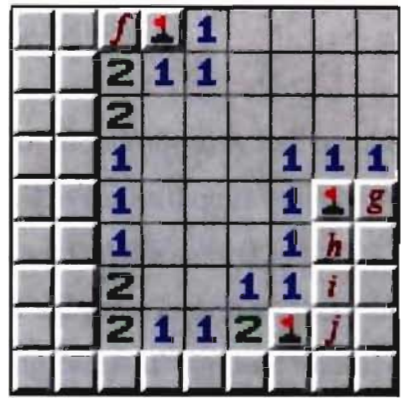


Fig. II.4 Theorem 2 gives numbered square positions.

Repeatedly applying step B results in the following game position. We attempt to apply Theorems 1 and 2 to the board, but there is no square which satisfies the conditions. Therefore we must return to step A and click randomly in the set of unrevealed squares.

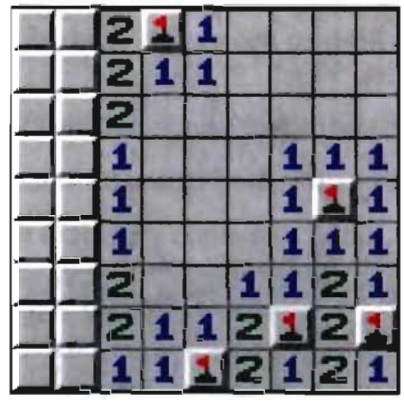


Fig. II.5 Conditions of Theorems 1 and 2 do not apply.

Unfortunately, a bomb is revealed and the game is over.

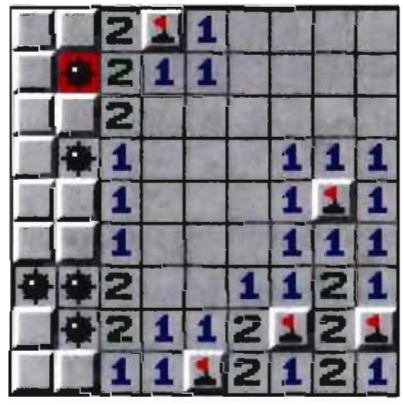


Fig II.6 Game over.

II.3 Advanced Logical Reasoning

This strategy is good for proportionately low numbers of bombs, but as the previous example demonstrated, it can still quite easily result in a loss. Could this loss have been prevented? It turns out that the answer is yes.

Experienced players of the game know that there are more advanced strategies than merely applying BS. Using "what-if" types of logical reasoning can give information about squares and their contents not attainable through BS. These strategies can therefore increase the likelihood of a winning game. For example, suppose a player has reached Fig. II.5 and does not want the next click to be a completely random guess. The following line of reasoning might be used:

Two of the squares *m, n,* and *p* are bombs because of the 2 at position (3,3). There are $\binom{3}{2} = 3$ ways that this can happen, but we can rule out the two bombs being under both of the squares *n* and *p* because that would make the number at position (4,3) at least two, which it clearly is not. Therefore, either both *m* and *n* contain bombs or both *m* and *p* contain bombs. Thus, square *m* definitely contains a bomb, and it may be flagged.

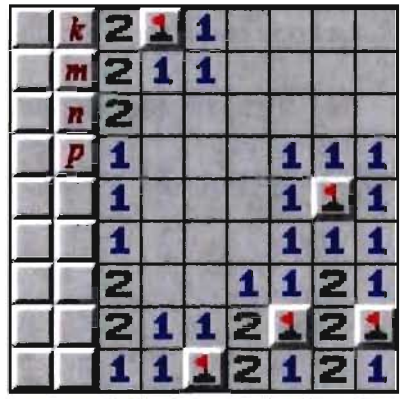


Fig. II.7 Logical reasoning gives new information.

By this line of reasoning, we have determined the location of a bomb despite the failures of Theorems 1 and 2 to do so. Having flagged a bomb, we may now continue to use BS until another such situation arises. Similar lines of reasoning can used to identify both squares containing numbers and those that contain bombs.

III. Application to Linear Algebra

Some of the published academic literature concerning Minesweeper attempts to extrapolate higher levels of logical gameplay into theorem-based strategies which take forms similar to BS, with specialized theorems for various game situations. *A Mathematical Introduction to the Game of Minesweeper*, by Philip Crow, describes no fewer than fifteen such theorems which, the author suggests, should be memorized by any serious player to improve their winning percentage (Crow 37-40). However, there is a way of modeling the game which supercedes all such specific board-position-based strategies and combines them all into a general form which can be symbolically described using a very common mathematical notation.

III.1 Notation and Definitions

- For a board B, define B^U as the set of all unknown squares.
- Given a board B, we call the set of unknown squares in B that are adjacent to any revealed numbered square the *perimeter* of B. We denote it symbolically by $P_B = \{B_{i,j} \in B^U : R^B_{i,j} \text{ contains at least one known numbered square}\}$.
- Define $|P_B|$ as the number of elements in P_B.
- Give a set of squares S, define $F(S)$ as the number of legitimately flagged squares in S.

III.2 Linear Equations

Any given board-position on a board B can be modeled with a system of linear equations. The system is generated as follows:

- Order the squares in P_B sequentially by row and column.
- Define P_B^i as the i-th element of the newly ordered P_B.
- Label the squares in P_B as $x_1, x_2, x_3, \ldots, x_{|P_B|}$.

- For every $B_{i,j}$ such that $R^B_{i,j} \cap P_B \neq \varnothing$, create a linear equation of the form:

$$\sum_{P^k_B \in R^R_{i,j}} x_k = K\left(B_{i,j}\right) - F\left(R^B_{i,j}\right).$$

- Call this system of equations D_B.

D_B can be converted into the form $\Gamma \mathbf{x} = \sigma$, where Γ represents each linear sum, x is a $|P_B|$-dimensional vector, and σ represents the $K\left(B_{i,j}\right) - F\left(R^B_{i,j}\right)$ solution column. Because each element of x represents whether or not a certain square contains a bomb, x is composed of only ones and zeros: $x = \left[x_1, x_2, \ldots, x_{|P_B|}\right]^T$, where $x_k \in \mathbb{Z}_2$.

For example, consider the following board B.

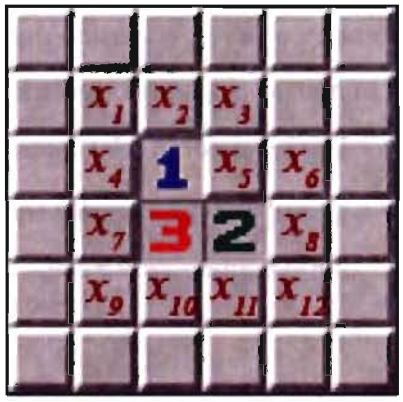


Fig. III.1 A simple perimeter example.

Because there are three numbered squares adjacent to the perimeter, we can create three linear equations D_B:

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_7 = 1$$
$$x_4 + x_5 + x_7 + x_9 + x_{10} + x_{11} = 3$$
$$x_5 + x_6 + x_8 + x_{10} + x_{11} + x_{12} = 2.$$

In matrix form:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}.$$

Every solution $\mathbf{x}^*$ of this matrix equation corresponds to a configuration of bombs in the perimeter that fits the constraints of the board. The only exception occurs when $\|\mathbf{x}^*\|_1 > b - F(B)$, where $\|\mathbf{x}^*\|_1$ is the sum of all elements in $\mathbf{x}^*$. (Recall that b represents the total number of bombs on B). These solutions are invalid because they require more bombs than there are remaining on the board.

Suppose Fig. III.1 is a 6×6 board with 11 bombs. An exhaustive computer search results in 19 possible arrangements of in the perimeter that fit the constraints of the board. However, none of perimeter squares is certain to contain a bomb or be safely clickable because all 12 x_k variables take on both 0 and 1 variously in these 19 arrangements. At this point, in the interests of developing a winning algorithm, we shall compute the probability of each square containing a bomb.

III.3 Computing Probabilities

It is tempting to compute the probabilities by counting the number of times each square contains a bomb within the 19 solutions and dividing by the total, but this is not correct. We notice that four of the solutions contain three bombs in the perimeter and the other fifteen contain four bombs in the perimeter. Because the 21 unknown squares can contain different numbers of bombs for these two cases, a weighted average must be computed:

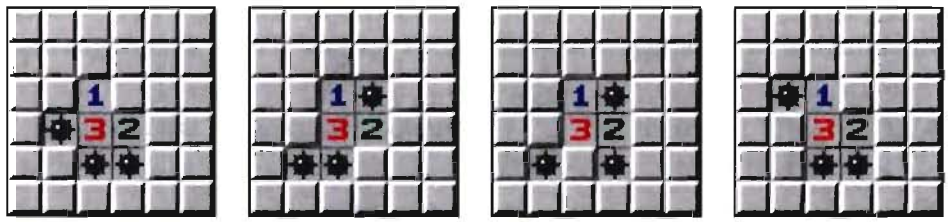


Fig III.2a All solutions with 3 bombs.

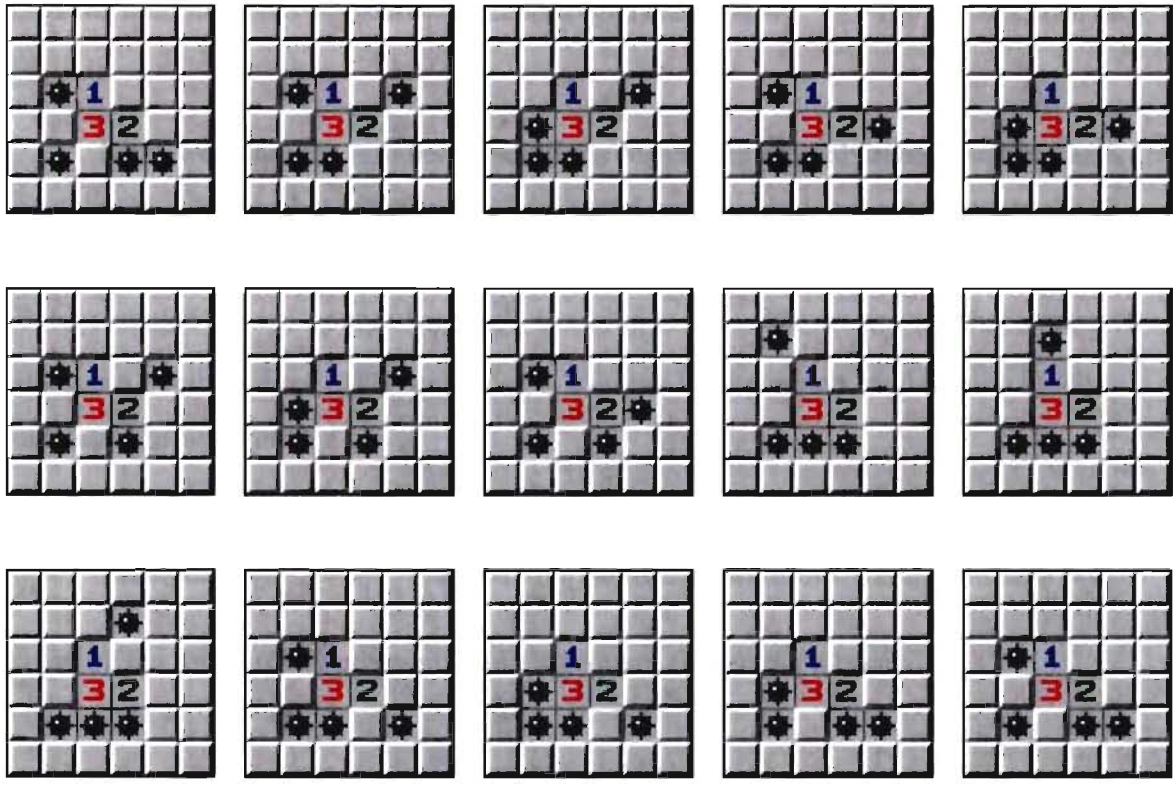


Fig III.2b All solutions with 4 bombs.

We know describe an exhaustive perimeter probability (EPP) algorithm.

Perimeter arrangements with 3 bombs: 4. Possible arrangements of the remaining $11-3=8$ bombs in the 21 unknown squares: $\binom{21}{8}$. So, there are $4\binom{21}{8}=813{,}960$ total bomb arrangements that have 3 bombs in the perimeter.

Perimeter arrangements with 4 bombs: 15. Possible arrangements of the remaining $11-4=7$ bombs in the 21 unknown squares: $\binom{21}{7}$. So, there are $15\binom{21}{7}=1,744,200$ total bomb arrangements that have 4 bombs in the perimeter.

Thus the total number of possible bomb arrangements T is $4\binom{21}{8}+15\binom{21}{7}=2,558,160.$

Now, we count the number of solutions containing a bomb in each square, and sort them by whether that solution has 3 or 4 total bombs. Now we can compute the weighted average. For example, suppose we want to compute the probability that square 10, represented by x_{10}, contains a bomb. Looking at the solutions, we see that square 10 contains a bomb in three of the three-bomb solution arrangements, and in nine of the four-bomb arrangements.

Therefore $3\binom{21}{8}+9\binom{21}{7}$ is the total number of solutions which contain a bomb in square ten, making the probability of that square containing a bomb

$$P(x_{10})=\frac{3\binom{21}{8}+9\binom{21}{7}}{T}\approx 0.648.$$

Now we use the EPP algorithm on the remaining 11 perimeter squares.

k	$P(x_k)$
1	0.0454
2	0.0454
3	0.0454
4	0.352
5	0.159
6	0.182
7	0.352
8	0.182
9	0.841
10	0.648
11	0.648
12	0.182

This information shows that the player is least likely to click on a bomb if he/she clicks on one of x_1, x_2, or x_3, and most likely to click on a bomb on x_9.

Suppose we had used the EPP algorithm after arriving at the board position in Fig. II.5. We can relabel the perimeter squares as follows.

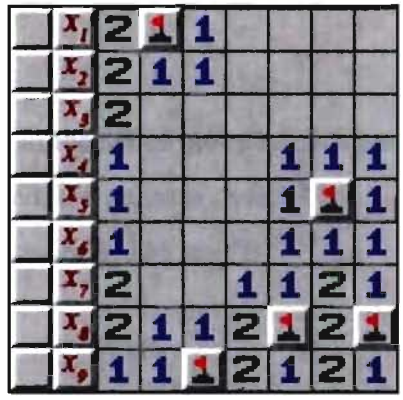


Fig. III.3 Another look at Fig II.5

We know that there are 10 bombs on the entire board, 5 of which have previously been flagged. We can now use the EPP algorithm.

k	$P(x_k)$
1	0
2	1
3	0
4	1
5	0
6	0
7	1
8	1
9	0

Squares with a probability of zero can be safely clicked and those with a probability of one can be legitimately flagged.

Very often the EPP algorithm results in probabilities of 0 or 1, allowing the player to proceed safely without the risk of clicking on a bomb. As such it is a very good algorithm for playing the game, and we can now describe a strategy called Perimeter Strategy (PS) which utilizes it:

A.

- Click a random square in the set of unrevealed squares on the board.
  - If a bomb is revealed, the game is over.
  - If a bomb is not revealed, go to step B.

B.

- Apply the EPP algorithm to the board.
- Flag all squares in the perimeter with probability of 1.
- Flag all squares outside the perimeter with a probability of 1.
  - If the number of newly flagged squares plus the old flags equals the total number of bombs on the board b, the player has found all the bombs and wins.
- Click all squares in the perimeter with probability of 0.
- Click all squares outside the perimeter with a probability of 0.
- If any square with a probability of 1 or 0 was found, repeat step B.
- If no square with a probability of 1 or 0 was found, go to step C.

C.

- If there is no perimeter, go to step A.
- Click on the square in the perimeter (or the unknown square) with the lowest probability. If any squares identically have the lowest probability, select one at random.
    - If a bomb is revealed, the game is over.
    - If a bomb is not revealed, go to step B.

IV. Algorithm Efficiency

The Perimeter Strategy is very effective, but it is computationally very taxing. In order to calculate every possible solution for a given perimeter (and thereby calculate the probabilities), every single possible arrangement of bombs in that perimeter has to be checked. This means that if a perimeter contains n bombs, 2^n arrangements need to be checked against the board constraints. These numbers get very large very fast. For example, in a 10×10 game, it is not uncommon to encounter perimeters of 30 squares or higher, which require checking at least $2^{30} = 1{,}073{,}741{,}824$ number arrangements, a number far beyond the reasonable time constraints of any desktop computer. Thus, even with the strength of the strategy in finding board probabilities, it is often limited by computational speed and time available.

IV.1 Linear Programming

Several methods can be used to increase the time efficiency of the Perimeter Strategy. Before apply the EPP algorithm we can create a linear program to determine the minimum and maximum number of possible bombs in all the perimeter arrangements.

Given a system of linear equations D_B, set up the pure integer programs $L_{\min}$:

$$\min \sum_{k=1}^{|P_B|} x_k \text{ subject to the constraints of } D_B \text{ and } x_i \in \mathbb{Z}_2$$

and $L_{\max}$:

$$\max \sum_{k=1}^{|P_B|} x_k \text{ subject to the constraints of } D_B \text{ and } x_i \in \mathbb{Z}_2.$$

Solving $L_{\min}$ and $L_{\max}$ using the Simplex Algorithm gives the minimum $(l_{\min})$ and maximum $(l_{\max})$ number of possible bombs in all perimeter arrangements, respectively. Now, when we can apply a modified EPP algorithm where instead of testing all 2^n arrangements of bombs, we must only test $\sum_{k=l_{\min}}^{l_{\max}} \binom{n}{k}$ arrangements, namely the ones that contain between $l_{\min}$ and $l_{\max}$ bombs inclusively.

Using this method, the example in Fig III.1 would only require testing $\binom{12}{3}+\binom{12}{4}=715$ perimeter arrangements rather than $2^{12}=4,096$. This is a savings of 83 percent.

IV.2 Preempting PS with BS

Often the Simplex Algorithm is not enough to decrease the search space to a reasonable size. However, in these cases with large perimeters, BS can often reveal squares that certainly contain or do not contain bombs. This can bring the player closer to solving the board without having to use PS. In fact, large perimeters are actually more likely to contain situations solvable by BS. BS is order $O(n)$ because it has to test at most 8 squares for each square on the board. However, because this method does not always work, the problem of overcoming an exhaustive search is not entirely solved.

IV.3 Partitioning the Perimeter

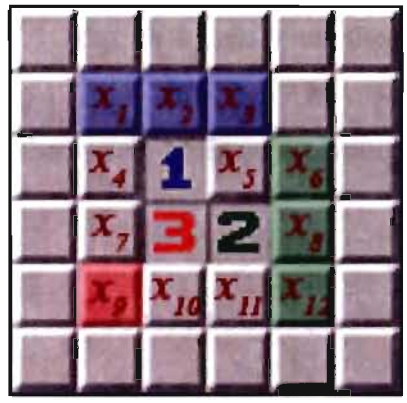
In many cases, the perimeter squares can be partitioned into the union of smaller subsets with individual special properties. Each subset possesses one of the following properties:

- Every square in it is commonly adjacent to exactly one known numbered square.
- It consists of one square adjacent to more than one known numbered squares.

IV.3.1 Notation and Definitions

- Each of these subsets we call a *partition of size k*, where *k* is the number of squares in that subset.
- Given a board *B*, denote K_α^B as the α-th partition of a perimeter P_B.

Using the board position in Fig. III.1, the partitions are highlighted and given below.



$K_1^B = \{x_1, x_2, x_3\}$
$K_2^B = \{x_6, x_8, x_{12}\}$
$K_3^B = \{x_9\}$

Fig. IV.1 Partitioning the perimeter.

Only the partitions where every one of its squares is commonly adjacent to exactly one known numbered square are shown. From this point, a new system of linear equations D_B' can be made:

$$K_1^B + x_4 + x_5 + x_7 = 1$$
$$x_4 + x_5 + x_7 + K_3^B + x_{10} + x_{11} = 3$$
$$x_5 + K_2^B + x_{10} + x_{11} = 2,$$

where $K_1^B \in \mathbb{Z}_2$, $K_2^B \in \mathbb{Z}_3$, $K_3^B \in \mathbb{Z}_2$, $x_i \in \mathbb{Z}_2$.

This can be simplified by relabeling the perimeter squares

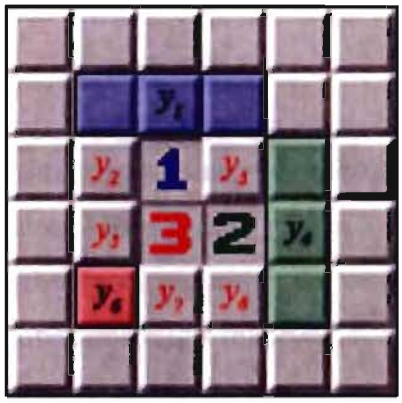


Fig. IV.2 Relabeling the perimeter.

and by again rewriting the system of linear equations D_B'

$$y_1 + y_2 + y_3 + y_5 = 1$$
$$y_2 + y_3 + y_5 + y_6 + y_7 + y_8 = 3$$
$$y_3 + y_4 + y_7 + y_8 = 2,$$

where $y_1, y_2, y_3, y_5, y_6, y_7, y_8 \in \mathbb{Z}_2$, $y_4 \in \mathbb{Z}_3$.

We now use D_B', which has fewer unknowns than D_B, in a modified EPP algorithm called the perimeter partition probability (PPP) algorithm that accounts for the variables that take on non-binary values. The results of PPP are exactly the same as if we had used D_B in the EPP algorithm, but are given to us relatively faster. Note that the Simplex Algorithm is still applicable and the methods described in IV.1 can be used on D_B'.

The problem with all of these methods is that they do not consistently improve the efficiency from $O(2^n)$. This is due to a result given by Richard Kaye, that determining the possible mine configurations for a perimeter is NP-Complete (Kaye 12). This means that finding a polynomial-time algorithm for solving any perimeter may very well be an impossible task.

V. Results

To statistically analyze these algorithms, we designed and implemented computer programs that played many random Minesweeper games using each strategy. This allows us to analyze the relative strength of each algorithm for various board configurations. Because of the computational time constraints involved, we used a relatively small board size for which $O(2^n)$ would not be prohibitive.

V.1. Levels of Strategy

We begin with an overview of all strategies previously mentioned, starting with an implicit base strategy.

Level 0: Random play

At this level, every square is clicked or flagged with an equal probability. The success rate of this "strategy" is far too low to be worth considering. This level of strategy is only included for completeness.

Level 1: Basic Strategy (BS)

This is the strategy where only immediately obvious adjacent bombs and open squares are clicked or flagged. When it fails to determine such squares random play is used. This level of strategy is practical for a human player.

Level 2: Perimeter Strategy (PS)

This strategy is a superset of all forms of the EPP greedy algorithm, which determines exact probabilities of each square in the perimeter having a bomb and clicks accordingly. This level of strategy is practical for a desktop computer if the parameters are carefully controlled.

Level 3: Perfect Strategy

This is the perfect strategy for playing Minesweeper. It proceeds as follows:

I. For each board state, consider every possible game which could proceed from this point.
II. Eliminate all losing games from this list.
III. Sort these games by the click that begins them.
IV. Make the move which begins the highest percentage of these games.
V. Repeat.

This level of strategy is not practical for any computer known to man. There are approximately 2.9×10^{115} games which can occur for the "beginner" board (9×9 with 10 bombs). This is vastly larger than the number of atoms in the universe.

V.2 Computer Programs

We implemented strategy levels 1 and 2 in Matlab and represented their results graphically. We programmed a computer to play games of Minesweeper on 1000 random 4×4 boards for each number of bombs ranging from 0 to 16. We then graphed the proportion of winning games for each number of bombs.

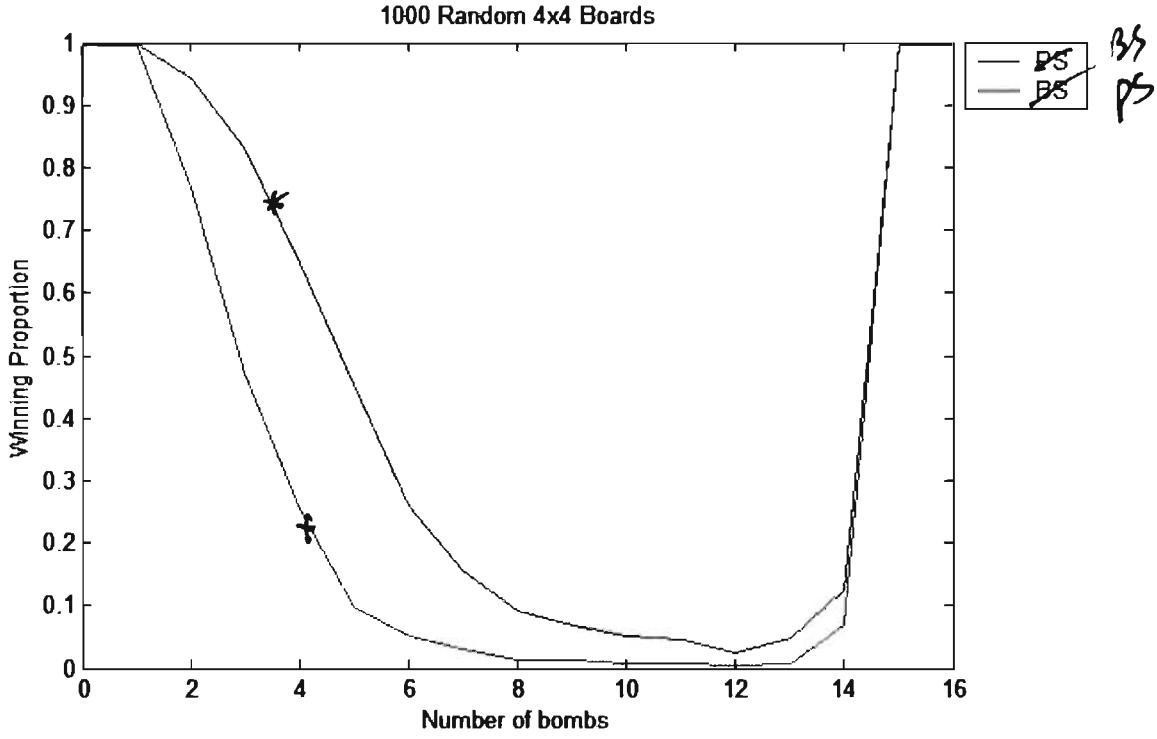


Fig. V.1 Winning proportion graph.

From the above graph, we can see that level 2 (PS) is on average 4.0 times more likely to result in a winning game than level 1 (BS). In both cases we follow the special rule that disallows a loss on the first click. As a result we win trivially when there is only one bomb and when there are all but one bomb. Furthermore, if the entire board is covered with bombs, then we know where they all are and win by default.

VI. Conclusion

The most important thing that our data have shown is that Minesweeper is still very much a game of chance, even when high-level strategies are used. Even using the exhaustive greedy algorithm, a 5/16 bomb proportion on the board results in a loss half the time. 5/16 is only 31% bombs, but if we look at the "expert" mode in Microsoft Minesweeper we see that it has only 21% bombs. This is in keeping with our data. A bomb percentage of about 10%-25% seems to be a good range for making the game fun

and interesting with a high level of challenge, while still retaining a high probability of winning when one plays well.

One shortcoming of our data is that the limits of our computer technology required us to use a 4x4 board, which is extremely small compared to what most would consider a normal game. However, we think that our data can be extrapolated to larger boards because the game is essentially the same on a larger grid. In fact, we theorize that certain factors like a smaller proportion of edge squares and potential for larger perimeter sets might even improve the algorithms if they are used at that level.

Because the algorithms seem to have such potential, the important future research would probably have to do with increasing their efficiency and making them applicable to larger boards and larger systems of equations. Despite the NP-Completeness of the general Minesweeper problem, it may be that for a very high percentage of board states the problem may be solvable in polynomial time. This means that it may be possible to use the greedy algorithm on boards of a size encountered in the actual game, using a normal desktop computer to calculate the percentages.

It may also be reasonable to develop heuristic strategies for playing the game in general, such as determining where it is best to make the very first click and what should be done if the greedy algorithm fails. However, these probably would not be applicable to the Linear Algebra of the problem, and would instead be important for casual players of the game. But we're sure the mathematicians wouldn't mind.

VII. References

Adamatzky, Andrew "How Cellular Automaton Plays Minesweeper." *Applied Mathematics and Computation.* 85 (1997), no. 2-3, 127-137.

Carrano, Helman, Veroff. Data Abstraction and Problem Solving with C++: Walls and Mirrors. Addison Wesley, 1998

Condon, Anne. Computational Models of Games. Cambridge, MA: MIT Press, 1989

Crow, Philip "A Mathematical Introduction to the Game of Minesweeper." *The UMAP Journal.* 18 (1997), no. 1, 35-42.

DeVore, Jay L. Probability and Statistics for Engineering and the Sciences. Pacific, Grove, CA: Brooks/Cole, 2000

Kaye, Richard "Minesweeper is NP-complete." *The Mathematical Intelligencer.* 22 (2000), no. 2, 9-15.

Minesweeper as a Constraint Satisfaction Problem. Studholme, Chris. 2001. <http://www.cs.toronto.edu/~cvs/minesweeper/minesweeper.pdf>

Mossel, Elchanan "The Minesweeper game: Percolation and Complexity." *Combinatorics, Probability and Computing.* 11 (2002), no. 5, 487-499.

any real world applications?

Andrew Fowler
Andrew Young

Précis for Honors Thesis

How hard is it to win the game Minesweeper, and what algorithms can be developed to play it efficiently? This question is interesting because it allows us to examine a problem in Linear Algebra, a branch of mathematics, in a fun an interesting way. This is because the question of playing the game Minesweeper can be shown to be equivalent to solving a certain math problem. This means that the results that we have found are applicable not only for players of the game but to mathematicians who study certain problems in Linear Algebra,

Our methods were both mathematical and computational. First we examined the game mathematically and developed formal terminology and symbology for equating the game to the mathematics involved. Then we determined mathematical algorithms and strategies for solving these systems and thereby playing the game. To do this we used a combination of existing ideas in combinatorial mathematics and new ideas which applied to the problem at hand. Then we programmed a computer to implement these algorithms and strategies. We encountered a problem with algorithm efficiency, where the time required for certain algorithms was prohibitively long for our computers, and we attempted to solve this problem (with some success) by redesigning the algorithms with time efficiency in mind. This allowed us to undertake a statistical analysis of the game of Minesweeper using our newly developed algorithms on a personal computer.

Our statistical analysis showed us the relative effectiveness of the algorithms and strategies we had developed. The most important implication of our results was that even with powerful algorithms the game is still very hard to win consistently. Often elements

of chance and guessing come into play even when the equations are entirely solved. However, at levels where the average player plays the game, the algorithms were found to be very effective.

Because so little about Minesweeper exists in the current mathematical literature, our results are in many cases the first of their kind. Our connection to Linear Algebra, however, makes our algorithms applicable and interesting to the mathematical side of the problem. If the research is to be continued, we plan to increase the efficiency of our algorithms so that they may be used with larger board sizes in reasonable amounts of time. We are confident that large increases in efficiency are possible, and are eager to see our algorithms at work on larger problems.