The Elusive Logic of Minesweeper

R. Michael Zerby

Abstract:

This paper provides a solution and the logic behind the solution to the game of Minesweeper. Minesweeper the game, included in the Microsoft's windows operating system, has been connected to the P-NP problem by Richard Kaye of the University of Birmingham, England, (Spring 2000).

1. Introduction

The game of minesweeper involves a certain amount of guessing which implies that you cannot always win the game. For example, the first move is always a guess where all of the cells have the same potential of being a bomb, so just select one at random. The object of the game is to uncover cells that do not contain bombs and note the locations that do.

But how is it possible to determine safe cells from ones that contain bombs? Safe cells are ones adjacent to uncovered cells that have a zero bomb potential. The value of an uncovered cell can be reduced by the number of adjacent bombs until it reaches zero. Cells containing bombs can be identified when a cell's bomb count is equal to the number of covered adjacent cells.

Where it is not possible to find any safe cells to uncover we must make a guess as to which cell to uncover next. Guessing is a complex problem involving the ratio of threats that a cell has of being a bomb. Cells still in play have a direct or an indirect potential of being a bomb. A direct potential is when the adjacent uncovered cell has a value other than zero. An indirect potential is when there are no directly adjacent uncovered cells. In the first move, all the cells have an indirect potential threat equal to the number of bombs divided by the number of cells that are not bombs.

It's just that simple with a few minor twists. Now to make a guess, find the sets of cells with the lowest potential threat of being a bomb. Do this by keeping a list of all the uncovered cells that have not been reduced to a potential of zero. Then randomly select one cell from the sets of cells having the lowest potential threat.

The indirect potential becomes more complex once there are cells with a direct potential. The bombs are now reflected as part of the direct potential and can not be part of the indirect potential. So we need to calculate the number of bombs that are part of the direct potential. Then reduce the number of bombs remaining by this value giving us the ratio of indirect cells to indirect bombs. We keep the list of all the covered cells that have a direct potential. For each cell in the list we keep its lowest potential threat and the reference to the uncovered cell that provided this value.

By grouping the covered cells by their uncovered reference we end up with either a complete or an incomplete set of bombs. The number of bombs in a complete set is a given. It is the incomplete sets of bombs that are the variable in the mix. We calculate its' potential by multiplying the number of covered cells by its own base ratio. We sum up these potentials and get the approximate number of bombs as a direct threat.

2. Background

L. Stewart (October 2000) published an article explaining that Minesweeper, the game, was extremely difficult or perhaps impossible to solve with a computer program. He also stated the Clay Mathematics Institute is offering a prize for the solution of seven infamous problems. One of these problem being the notorious P vs. NP question. He also noted that Minesweeper had been connected to the P vs. NP question by R. Kaye (Spring 2000). So by solving Minesweeper one might provide the answers to this long standing question.

3. Rules of play

There are three sets of rules. The first two sets of rules are those of *centric based dependence* and *centric based reflection*. These five rules work together as one and are key to solving the game. The last set of rules are those of *certainty* and serve to clarify the states of play.

3.1 Rules of centric based dependence:

1. When an uncovered cell's value is equal to the number of adjoining covered cells, then all of these cells must be bombs.
2. The bomb potential of uncovered cells can be reduced by the number of adjoining bombs until it reaches zero.
3. Covered cells adjoining an uncovered cell with a value zero are now safe to uncover.

3.2 Rules of centric based reflection:

1. Newly uncovered cells must reflect their state change to all their adjoining cells.
2. Cells that are identified as being a bomb must reflect this state to all their adjoining cells.

3.3 Rules of certainty:

1. Covered cells can have one of three states: *Safe*, *Unknown* and *Bomb*.
2. Covered cells begin with a state of *Unknown*
3. Uncovered cells can have one of two states: *Nullified* or *Not Nullified*.
4. Uncovered cells that have a value of zero are no longer in play and are *Nullified*
5. Uncovered cells remain in play as long as they have a value greater then zero and are *Not Nullified.*

4. Centric based circuits

There are three centric based circuits used in the solution of the game. Circuit one seen as Fig. 4.2 is used to locate adjoining bombs using rule 1 of *centric based dependence*. Circuit two seen as Fig. 4.3 is used to reduce the bomb potential of uncovered adjoining cells by using rule 2 of *centric based dependence*. Circuit three seen as Fig. 4.4 shows us cells that are safe to uncover by using rule 3 of *centric based dependence*. These three circuits work together to provide a solution to the game.

To illustrate the use of these three circuits. We will use five by five representation of the game, where rows 1 through 5 are vertically aligned and columns 1 through 5 are horizontally aligned. Positions are given as row then column separated with a comma enclosed in square brackets. So the first cell of the board is represented as [1,1]. Covered cells are represented by the letter C and uncovered cells by a number. Four illustration purposes cells at: [2,3], [3,1], [4,2] and [5,4] contain bombs.

We begin by randomly selecting any cell because they all have the same potential threat of being a bomb.

Fig. 4.1 Shows the results of the first move that uncovered a cell [1,3]. This move does not reveal any bomb, so we must make another guess. All the cells adjoining the newly uncovered cell have a 1 in 5 chance of being a bomb. All the other cells have a 3 in 19 chance of being a bomb. So we randomly select one of the 19 cells.

Fig. 4.1

C	C	1	C	C
C	C	C	C	C
C	C	C	C	C
C	C	C	C	C
C	C	C	C	C

# - Adjoining bomb potential
C - Covered

Fig. 4.2 Shows cell [1,1] being the cell that was uncovered at random. Notice there was a series of cells automatically uncovered by the game. All adjoining

nullified cells are uncovered in a cascading manner until reaching a *not nullified* cell. This move reveals the location of our first bomb. Apply rule 1 of *centric based dependence* to cell [1,2] having a potential of one and one adjoining covered cell [2,3]. These two cells are circled indicating their relationship. We have determined that cell [2,3] contains a bomb and will use the letter B to represent this fact.

Fig. 4.2

0	1	1	C	C
1	2	B	C	C
C	C	C	C	C
C	C	C	C	C
C	C	C	C	C

# - Adjoining bomb potential
B - Bomb
C - Covered

Fig. 4.3 Applying rule 2 of *centric based dependence* to all the uncovered adjoining cells surrounding our newly discovered bomb and are circled to indicate this relationship. We are now able to reduce the bomb potential of any adjoining uncovered cells, which are underlined. We can see that there are two cells whose values were reduced to zero; these two cells are now considered to be *nullified*.

Fig. 4.3

0	$\underline{0}$	$\underline{0}$	C	C
1	$\underline{1}$	B	C	C
C	C	C	C	C
C	C	C	C	C
C	C	C	C	C

# - Adjoining bomb potential
$\underline{\#}$ - Reduced adjoining bomb potential
B - Bomb
C - Covered

Fig. 4.4 Applying rule 3 of *centric based dependence* to all the newly neutralized cells allows us to determine the location of any safe cells present. Out of the two *nullified* cells only one of them have any adjoining covered cells. These cells have been circled to indicate their relationship. There are two covered cells that adjoin the newly *nullified* cell. These cells are safe to uncover and we will use the letter S to indicate this fact. These cells will be uncovered and the game will continue.

Fig. 4.4

0	$\underline{0}$	$\underline{0}$	S	C
1	$\underline{1}$	B	S	C
C	C	C	C	C
C	C	C	C	C
C	C	C	C	C

# - Adjoining bomb potential
$\underline{\#}$ - Reduced adjoining bomb potential
B - Bomb
C - Covered
S - Safe

5. The solution

The solution begins with 5.1 *The initial move* and repeats 5.2 *The five steps of play* until we win or lose the game.

5.1 The initial move

The first move is always a guess and may succeed or fail. All cells have the same potential of failure *f* being the ratio of cells *c* to bombs *b*.

$f = b/c$

5.2 The five steps of play

5.3 Reading the new display states
5.4 Bomb detection
5.5 Nullified cell detection
5.6 Guessing
5.7 Uncovering cells

5.3 Reading the new display states

Each turn begins by reading the display state of all the cells from the board. Cells that have just been uncovered must reflect this change to all its' adjoining cells. We also need to collect the set of cells that are uncovered and still in play. The following runtime analogy embodies this process for us.

$f1(s1) = [s1](op1+op2)$

f1 - Function of reading the new display states

f1.s1 - Set of cells that comprise the game
f1.op1 - Setting the display state
f1.op2 - Collecting uncovered cells in play

f2(f1.op1) = t1{op1+[s1](op2)+t2{op3}}

f2 - Function of setting the display state
f2.t1- If this cell has just been uncovered
f2.s1- Set of adjoining cells
f2.op1 - Record the new display value
f2.op2 - Decrement number of adjoining covered cells
f2.t2 - If the display value is zero
f2.op3 - Set state to *Nullified*

f3(f1.op2) = t1{op1}

f3 - Function of collecting uncovered cells in play
f3.t1 - If this cell is uncovered and is in play
f3.op1 - Append to list

Examining the runtime analogy *f2*, we see that it begins with a test *f2.t1*. This test limits its execution to the set of newly uncovered cells. Each newly uncovered cell can only have 3, 5 or 8 adjoining cells, based on their location and most will have 8. So the upper runtime limit of this algorithm is the sum of all the cells that are not bombs multiplied by 8. We also need to add the runtime of *f2.t2{f2.op3}*, which is the set of newly uncovered cells that are *nullified* multiplied by 8. The set of newly uncovered cells that are *nullified* will fluctuate between sum upper and lower bounds based on the laws of combinatorial probability. So *f2s* runtime complexity is not connected to the number of turns necessary to solve the game; allowing us to isolate its runtime complexity from that of *f1*.

Examining the runtime analogy *f3*, we see that it begins with a test *f3.t1*. This test limits its' execution to the set of uncovered cells still in play. The number of uncovered cells in play and the number of turns, vary between some predictable upper and lower limit based on the laws of combinatorial probability. We see evidence supporting this claim; watching the runtime characteristics of *Neutralized Cells* to *Plays* displayed on the game form during the course of play.

Examining the runtime analogy *f1*, we see that its runtime varies directly with the number of turns needed to solve the game. We will see that *f1* is also subject to the law of combinatorial probability similar to that of *f3* and has some predictable upper and lower bounds too.

5.4 Bomb detection

Using the list of uncovered cells, we look for bombs that have become apparent at this time. Bombs become apparent when the number of covered adjoining cells are equal to the cells *potential value*. When we find a cell where this condition is true, we know that all the adjoining covered cells are bombs, and can mark them as so. We will also check to see if we have found the last remaining bomb in play. In the event we have found the last bomb, we are free to uncover all the remaining cells that are not bombs, thus winning the game. We can also win the game by uncovering the last safe cell. The runtime profile of bomb-detection is dependent on the set of uncovered cells still in play, which is dependent on the laws of combinatorial probability.

f4(s1) = [s1](t1{op1}+t2{op2})

f4 - Function of bomb detection
f4.s1 - Set of uncovered cells in play
f4.t1 - If the number of adjoining covered cells are equal to current potential
f4.op1 - Set all adjoining covered cells state to bomb and Append to list of bombs
f4.t2 - If all bombs are accounted for
f4.op2 - Append all covered cells to the list of safe cells

5.5 Nullified cell detection

Using the set of cells that are known to be bombs, we look for adjacent cells that have been *nullified* at this time as a direct result of identifying the bombs. *Nullified* cells become apparent when all the adjoining bombs have been found. When a cell has been *nullified* all of its adjoining cells that are not bombs are added to the list of safe cells, these cells are then uncovered ending this turn. The runtime profile associated to this functionality can be reduced even further by limiting *f5.s1* to the set of newly discovered bombs for this turn because we will only discover each bomb once and the number of bombs are a finite set. Therefore the runtime profile of this function is based on the product of the bombs and not the number of turns necessary to solve the game.

f5(s1) = [s1]([s2](t1{[s3](t2{op1})}))

f5 - Function of neutralized cell detection
f5.s1 - Set of bombs
f5.s2 - Set of adjoining cells
f5.t1 - If this cell is uncovered, in play, *Not Nullified* and has a potential of zero
f5.s3 - Set of adjoining cells
f5.t2 - If this cell is covered and not a bomb
f5.op1 - Append it to list of safe cells

5.6 Guessing

If there are no safe cells to uncover, we must make an educated guess. Guessing is more of a complex problem involving the ratios of potential threats that the cell has of being a bomb. Covered cells still in play have a direct or an indirect potential of being a bomb. A direct potential is when the adjacent uncovered cell has a value other then zero. Uncovered cells having a value other then zero are referred to as *centric values*. An *indirect potential* is when there are no directly adjacent uncovered cells. In the case of the first move all the cells have an *indirect potential* threat equal to the number of bombs divided by the number of cells that are not bombs.

The *indirect potential* becomes more complex once we have cells with a *direct potential*. The reason is that the bombs are now reflected as a part of the *direct potential* and can not be part of the *indirect potential*. So we need to calculate the number of bombs that are part of the *direct potential*. We then reduce the number of bombs remaining by this value giving us the ratio of *indirect cells* to *indirect bombs*. We keep a list of all the covered cells that have a *direct potential*. For each cell in the list we keep the lowest potential threat and the reference to the uncovered cell that gave us this value.

Then we group the covered cells by their *centric value* ending up with either a complete set or an incomplete set. We know the number of bombs for a complete set is a given. It is the sets of incomplete cells that are the are the variable in the mix. We approximate their potential by multiplying the number of covered cells by its own base ratio. We then sum up these potentials getting the approximate number of bombs that represent the direct threat. Subtracting this value from the total number of bombs we get the approximate number of *indirect bombs*. We then find the sets of cells with the lowest potential threat of being a bomb. We then randomly select one cell from this set of cells.

Guessing has one of the longest and most complex runtime analogies of any in this solution. The number of guesses during any given game seem to vary between some predictable upper and lower limits based on some laws of combinatory probability associated with the game. Please note that the runtime analogy below does not account for statements relating to composite gates or debugging statements.

f6(s1) = op0+[s1](t1{t2{op1+[s2](t3{t4{op2}+op3+t5{op4}}!t3{op5+t6{op6}}}!t2{t7{op7})op8}})+t8{op9+[s3](op10+t9{op11}!t9{op12})}+[s4](op13+t10{op14}!t10{op15})+op16+t10{op17}+t11{op18}+op19+t12{op20}+t13{op21+[s5](op22)+t14(op23)}}+op24+[s6](t15{op25}!t15{op26})+op27

f6 - Function to venturing a guess
f6.s1 - All the cells in play
f6.s2 - Set of adjoining cells
f6.s3 - Set of cells adjoining a centric value
f6.s4 - Set of cells being a centric value
f6.s5 - Set of cells with an indirect potential
f6.s6 - Set of cells being a centric potential
f6.t1 - If the cell is in play
f6.t2 - If the cell is uncovered
f6.t3 - If the cell is still in play and is covered
f6.t4 - If the key is in the sorted list
f6.t5 - If the key is not in the sorted list
f6.t6 - If this is a lower value currently recorded in list
f6.t7 - If cell has no direct potential
f6.t8 - If there are any cells with an indirect threat
f6.t9 - If this is a new centric value
f6.t10 - If the calculated value is less than zero
f6.t11 - If the calculated value is greater than upper limit
f6.t12 - If there are any direct threats present
f6.t13 - If the indirect threat is lower than the direct
f6.t14 - If there are any indirect threats present
f6.t15 - If centric potential is the same as the previous
f6.op0 - Create 7 object + increment a value
f6.op1 - Assignment of a division + value to string + object creation
f6.op2 - Insert into list
f6.op3 - Append to list
f6.op4 - Simple object creation + Append list + Insert into list + Insert into list
f6.op5 - Append to list
f6.op6 - Replace in list + Replace in list
f6.op7 - Append to list
f6.op8 - Insert into list
f6.op9 - Two variable initializations and object

creation
f6.op10 - Lookup by index
f6.op11 - Create object + Append list + Insert list
f6.op12 - Lookup by value + Append list
f6.op13 - Variable initialization + Reference by index
f6.op14 - Multiplication + Addition
f6.op15 - Indexed value + Multiplication + Addition
f6.op16 - Assignment of a subtraction + Assignment of a subtraction and a division
f6.op17 - Assign to zero
f6.op18 - Assign to upper limit
f6.op19 - Assignment of division and two multiplications + Assignment
f6.op20 - Assignment of a division and a comparison
f6.op21 - Assignment of object creation
f6.op22 - Assignment of object creation + Append list
f6.op23 - Assignment of string conversion and concatenation
f6.op24 - Three assignments + one division + list [0]
f6.op25 - Increment
f6.op26 - End loop (break)
f6.op27 - Two calls to function rand + two list[n] + assignment

5.7 Uncovering cells

This is the simplest operation, where we send each cell to be uncovered to the game. We can also see that the number of cells that can be uncovered will be limited to the number of cells that are not bombs. So the runtime of this function will not depend on the number of turns necessary to solve the game. Each time we uncover a cell we receive a status of *playing*, *won* or *lost* back. It is this status that drives the game play.

f7(s1) = [s1](op1+t1{op2}!t1{op3})

f7 - Function of uncovering cells for the list of neutralized cells
f7.s1 - Set of cells to be uncovered
f7.op1 - Sending the cell to be uncovered
f7.t1 - If the game has ended and we won
f7.op2 - End of game
f7.op3 - Throws an exception

5.8 While playing

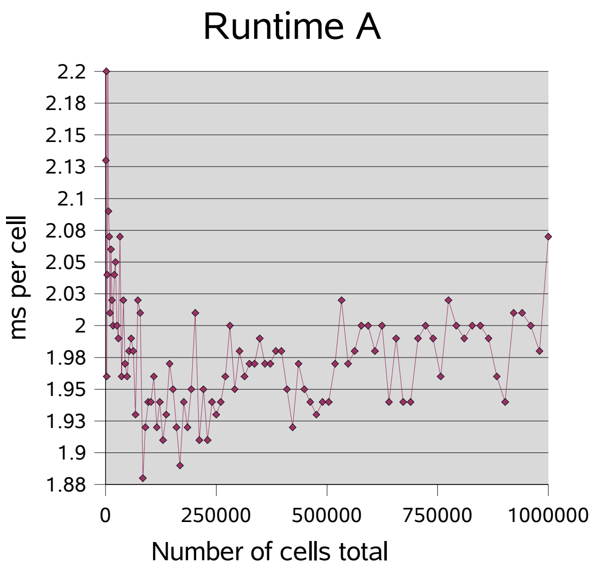
We continue to repeat the five states of play until we win or lose the game.

f0(s1) = [t1](op1+op2+t2{op3}+t3{op4}!t3{op5})

f0 - Function playing the game
f0.t1 - If still playing
f0.op1 - Setting the new display state *(f2)*
f0.op2 - Bomb detection *(f4)*
f0.t2 - If we have not won
f0.op3 - Neutralized cell detection *(f5)*
f0.t3 - If there are cells to uncover
f0.op4 - Action of uncovering cells *(f7)*
f0.op5 - Action of guessing *(f6)*

6. Results

The graph labeled *Runtime A* plots the average runtime per cell of winning games. The field of play starts at 100 cells and progresses to 1,000,000 cells, increasing by a factor of ten squared for each successful win. The number of bombs is ten percent of the field of play. This graph shows the solution works and works extremely well. The graph shows a flat rate of growth, varying between some upper and lower limit that is connected to the number of passes necessary to win the game.



Processor AMD 3200 with1GB DDR 400 memory

7 Beyond The Basic Solution

During the course of testing and refining of the solution we noticed a series of covered cells which enabled us to make an intelligent guess about where the bombs were. After a great deal of study the logic to exploit this discovery was found and is aptly named a *composite-gate*. Its efficiency might be improved by keeping track of the series of cells known not to produce a positive result. So we could skip them in repeated checks; however, we still get a better rate of success just as it is now. Note that this is a secondary measure that only improves the odds of winning.

You will find this section of code imbedded in the *(5.6) Guessing* section of code and can be enabled or disabled for testing purposes.

7.1 Composite gate

Unraveling the logic behind the composite gate begins by identifying a series of covered cells either vertically or horizontally aligned. This series of covered cells need to be tightly coupled to the gates. What is meant by this is that the gates are the adjacent uncovered cells which share a pair of inputs between them. Inputs are the covered cells. After we match up the gates and the inputs we can use some simple rules to determine where the bombs are. The rules are: *NOT OR, NOT AND,* and *Compound ANDS*. *NOT OR* occurs where one *OR* gate is a subset of another *OR* gate. In this case the inputs not contained in the union can not be part of the *OR* gate and must have the value of zero or off. *NOT ANDS* occurs at the union of an *AND* and *OR* gate. In this case we can reduce the *AND* gate leaving only the *OR* gate. *Compound ANDS* are based on the union of the tightly coupled *ANDS*. The first thing is that our *AND* gates are broken up into three possible combinations of sub *ANDS* as in Fig. 7.1. These Sub Ands form unions when tightly coupled that allow us to solve certain conditions. In the case where a *NOT OR* meets an *AND* gate we know the state of inputs 1 is off, which only leaves *ANDS* A2 which gives us A2 U B1 tells us that Input 0, 2 and 3 have the value of one or on. The code is a little more involved but that's the gist of it.

Fig 7.1 Decomposition of an And Gate

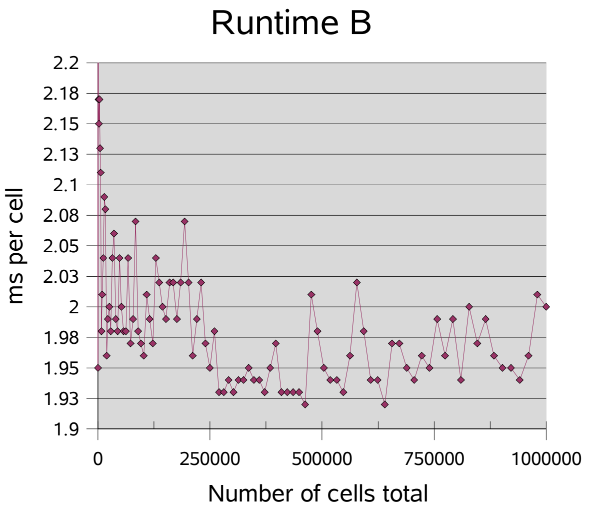
Sub Ands	**Input 0**	**Input 1**	**Input 2**
0	X	X	
1		X	X
2	X		X

Fig 7.2 Unions of Compound Ands

Ands	**Input 0**	**Input 1**	**Input 2**	**Input 3**	**Unions of Ands**
A0	X	X			A0 U B2
A1		X	X		A1 U B0
A2	X		X		A2 U B1
B0		X	X		B0 U A1
B1			X	X	B1 U A2
B2		X		X	B2 U A0

8. Additional results

The graph labeled *Runtime B* shows the performance of the solution incorporating the logic of *composite-gates*. We see a very similar runtime performance to that of *Runtime A*. There is one significant difference between them being the ratio of wins to losses. We need to win 100 games to complete the test using the range of 100 to 1,000,000 cells. In our first case it took a total of 795 games to complete the test. In our second case using *compound-gates* it only took 321 games to complete the test. This evidence demonstrates an increased performance as a result of incorporating this logic into the solution.



Processor AMD 3200 with1GB DDR 400 memory

9. Conclusions

The proof of this solution is a program that is able to play the game of Minesweeper. The evidence is presented in the form of runtime statistics that conclusively supports the realization that the program is able to play the game.

The question remains; do all these types of problems have similar solutions? Whether or not this solution provides the key to solving the fundamental P vs. NP question remains to be seen.

Addendum

Additional materials include source code, project file, and additional materials are provided at www.some_url.com.

Appendix

Runtime notations used:
1. [s*n*]() Series
2. [t*n*]() Conditional iteration
2. t*n*{} Inclusive test
3. !t*n*{} Exclusive test
4. op*n* Operation
n - numeric identifier

References

1. R. Kaye, "Minesweeper is NP-complete," Mathematical Intelligencer volume 22 number 4, 9-15 (Spring 2000).

2. L. Stewart, "Million-Dollar Minsweeper," Scientific America, 94-95 (October 2000).