Forums

Post your approach | Feedback: (+10/-0) | [+] [-] | Reply — Thu, Dec 24, 2020 at 12:55 AM BDT

jdmetz
2338 posts

This was a fun problem to work on. Thanks for posing it!

There seem to be three main parts to this problem:

1) Computing the probability that each covered cell is a mine.
2) Choosing the next cell to guess, balancing the probability it is a mine with the expected information gain.
3) Deciding when to stop guessing.

My approach is to keep a list of sets of cells along with the number of each set that must be a mine. Every new non-mine cell I uncover adds a new set. I remove duplicate sets and use set intersection to reduce the number of cells in sets. I flag mines as soon as I am certain of them, and I keep a list of safe cells.

Once the list of safe cells is exhausted, I try to compute probabilities of each cell being a mine. My approach here is pretty simple - the board starts off with the average probability, and I then iterate over all sets (including the full set of all covered cells) multiple times, multiplying the probabilities of cells in a set by a factor to make them sum to the total number of mines in the set.

Finally, for the first 10% of the board I always guess the cell with lowest probability. For the rest of the board I penalize cells that are not in any set (since I need them to be a 0 to prevent another guess), and estimate my expected score at the next time I think I will need to guess (based on frequency of past guessing). If the cell with the highest expected score is lower than the current score, I stop guessing. Otherwise I guess it.

My provisional score is 93.475 in 11th place.

Re: Post your approach (response to post by jdmetz) | Feedback: (+6/-0) | [+] [-] | Reply — Thu, Dec 24, 2020 at 1:43 AM BDT

My solution has three algorithms of increasing complexity to pick the next cell. The first to give an answer, other than STOP, returns it.

First is the simple algorithm of looking at a cell with a number (a sum), and seeing if the number of unallocated mines equals 0 or the number of free

murrayr
195 posts

spaces. Then it knowns all free spaces are either clear or mines. This appears to be a common solution since I had the same score, ~36, as a number of others at the time.

The second algorithm is a DFS search of collections of related free cells. The collections of cells were built from the free cells of a sum, merging in cells from other sums that overlapped, particularly focusing on those sums which had a low (free choose mines) value. The search was constrained by the sums covering the set of cells. The search started from each cell in turn, trying to satisfy the constraints with it set to clear or mine, and recording the satisfaction state for all cells when found. This way it tends to take just a small number of searches to discover any cell that cannot be satisfied with both clear and mine, in which case you know what it must be. This scored ~52.

At this point I added simple probability based guesses, taking the score to ~56. I also started to notice that in local testing, over 100 cases, I was getting an absolute score of around 50, so a submit score of 56 implied either a) my code was failing lots on the servers, or b) the top competitors were getting perfect scores on nearly every case, which seemed incredible at the time. It turns out my code wasn't failing, and I needed a much better way of calculating probabilities.

The third algorithm too two days of coding before I could test and run it. Here my boss would preach TDD to me, but hey, who has the time in a marathon match? Each sum covers a set of cells and has a set of possible mine/clear patterns that can fit in those cells. I have a class, PlaceBitSet, which can hold up to 64 cells and uses up to 1000 64-bit numbers to represent the possible patterns. These PlaceBitSets can be merged together to form larger groups of cells and the patterns that can appear in them, only compatible patterns are kept, which sometimes means that larger sets have fewer patterns. When they are too big to merge, they can still be intersected to limit the numbers of possible patterns. Each time a guess is required, all active sums have any existing PlaceBitSets restricted to cover the current free cells, then these are merged or intersected with their neighbours up to a box-distance of 4. Now each free cell is counted for the number of times it appears as a mine or clear, and this probability is used to make a guess. The guess tries to forcast the value of the cell based on its clear probability, size of free area it's in, and the history of success of guesses so far.

I have some big timeout issues with both the second and third algorthms, which can both timeout and report STOP if too close to the time limit. This was appearing in a couple of the example test cases. But the curious thing is that my provisional tests were not affected much by any changes to timeout handling code, which makes me wonder if the provisional tester is checking for timeout, and penalising it at all?

This was an absolute beast of a problem. It looked simple, but pushed me very far. I have learned a lot doing this. Thanks @dimkadimon

Re: Post your approach (response to post by **jdmetz**) | Feedback: (+2/-0) | [+] [-] | Reply

Thu, Dec 24, 2020 at 3:15 AM BDT

tuff
37 posts

Two questions:

```
AAA..BBB
AAA..BBB
AAx..yBB
AAA..BBB
AAA..BBB
```

I notice in cases like the above where D=8, As are known,, "."s are known or unknown, Bs are known or unknown and x and y are known counts; that the number of Bs that are mines can be calculated despite the unknown "."s. Did people code for these cases explicitly or were they recognized by more general methods? What sort of more general methods?

I considered using the following method when all else fails. Select a set of adjacent unknown cells bordering on known cells and examine all

consistent bomb/no-bomb assignments. If any cells were always bomb or always no-bomb over all consistent assignments then mark them as such. I ended up not implementing this as I estimated it would not help much. Would it have helped much? Did people do this?

Re: Post your approach (response to post by **tuff**) | Feedback: (+3/-0) | [+] [-] | Reply — Thu, Dec 24, 2020 at 3:40 AM BDT

jdmetz
2338 posts

This is what the set intersections I mentioned do. In this case since the As are known, x's set of unknowns includes the 10 '.'s. y's set of unknowns contains the 10 '.'s and the 14 'B's. If x is 5 and y is 7, then you know can subtract y's set from x's and the result is 2 mines among the 14 'B's. This helps the most in cases where the remaining Bs would have to all be mines or all clear after doing the subtraction.

Re: Post your approach (response to post by **jdmetz**) | Feedback: (+6/-0) | [+] [-] | Reply — 1 edit | Thu, Dec 24, 2020 at 4:24 AM BDT

maniek
463 posts

I did backtracking to prove that there is no bomb. That is, try placing a bomb and (by recursively placing bombs) try resolving all connected squares. If there is no solution, it means there is provably no bomb. This solves most tests with score 1.0, and gives score of 0.916 with no guessing for seed 2, which seems to be optimal (if we don't risk guessing unnecessarily).

It's sometimes too slow (but my implementation can probably be sped up by a factor of ~5), so I cut the search depth if the time remaining is nearing exhaustion.

Then I try guessing. I try the squares with lowest probability of bomb, but I prefer squares where there are many uncovered squares nearby.

I think there is some meta-optimization of guessing to do, where you fine-tune some parameters by trying lots of cases. I did not get around to doing that.

Score 94.68334, 8th place.

Re: Post your approach (response to post by **maniek**) | Feedback: (+5/-0) | [+] [-] | Reply — Fri, Dec 25, 2020 at 2:03 AM BDT

sullyper
279 posts

I did not have much time exploring what was the best way to guess, so my solution is pretty much:
- Find obvious safe moves
- Find not as obvious safe moves (writing a set of equation x1 + x2 + .. + xn = V, it gives me a matrix, with some gaussian pivot elimination you can sometimes find other easy solution. Especially if you keep track of the min/max possible value (given each x is binary), if the target value is equal to the min, then all positive are free cells, and all negative entry are mines, and the opposite if the target == max.
- If there is not too many unknowns, I do a bruteforce using my matrix for early exit. What I do is try to set the value for a given cell and see if I can find a solution, usually it returns quickly a solution, or quickly figure out that there is no solution. I confirm that for seed 2, 0.916 is optimal with no guess. And guessing would be too risky

FInally, if there is no safe move, I tried locally quickly few random heuristic, I don't even remember which one ended up scoring better, but I pick a random move if my score is below some threshold.

mugurelionut
299 posts

My solution consists of several approaches which I initially implemented separately and which I eventually managed to stitch into a single solution in the last few hours of the contest (giving me the extra score boost required to reach the provisional 1st place). At each step I am running all of the following approaches, skipping to the next iteration as soon as I find out the value of a new cell (whether that's an empty cell or a mine).

1) Try out simple safe moves (cells around a known free cell with a remaining count of zero - the version with all 1s is covered by another case)

2) I have the board split into so-called groups. A group is a set of unknown cells which have the same set of uncovered empty cells in their neighborhood. For these groups I maintain a list of up to X solutions consistent with the information available so far (a solution consists of the mine count in each group). For each solution I also compute its weight as the product of combinations. The biggest part is probably the term corresponding to C(num cells outside of groups, num remaining mines outside of the current solution). Using long double in C++ was enough to get a reasonable precision (I don't need exact numbers here). So I only keep the top X solutions in terms of weight.

If the list of solutions contains all the possible solutions, then I choose the groups which have all zeroes or all ones in all the solutions.

3) I generate an equation for each non-zero uncovered empty cell and propagate upper and lower bounds of the number of mines to subsets of groups. I start from the main set S of each equation and consider all of its subsets S1. If the lower and upper bounds of S and S1 are fixed, then this can lead to new lower/upper bounds for S2 = S \ S1. I only tried propagation towards subsets, not from subsets towards supersets (it should be possible to also update lower/upper bounds for S given fixed lower/upper bounds for S1 and S2, where S = S1 U S2).

4) I consider all the single element groups which have all zeroes or all ones in the list of solutions (if any). Let's assume there is such a cell C which is all Xs (X=0,1) in my (partial) list of solutions. Then I try to find a solution with its value fixed to 1-X. If I can't find one, then the cell's value must necessarily be X. If I find one, I add it to my partial list of solutions. If the new solution invalidates some all zero/ones groups from earlier, then I don't consider them later on. I use backtracking in order to find a solution with a cell's value being set. I also set a time limit for each individual backtracking, so it's possible for this to finish without finding a solution because of the time limit - in this case nothing can be inferred for the starting cell.

5) Finally, in this step I compute the probability of each cell of each group to be a zero. For this I use the partial list of weighted solutions. I also approximate the probability of a cell outside of the current groups to be a zero as simply the number of remaining (still hidden) empty cells divided by (the number of remaining (still hidden) empty cells + the number of remaining mines). I pick the group (or the outer group) with the highest probability of being a zero and then I ask about it only if I uncovered less than 50% of the empty cells so far (so independent of the number of exploded mines, expected score, actual probability, etc.). I tried other formulas (including an optimistic expected score formula), but they didn't seem to work better.

Example scores:
Seed 1: 1.0
Seed 2: 0.861714
Seed 3: 1.0
Seed 4: 1.0
Seed 5: 1.0
Seed 6: 0.996861
Seed 7: 0.454545 (1 mine exploded)
Seed 8: 0.5 (1 mine exploded)
Seed 9: 1.0
Seed 10: 1.0

For seed 2 I confirm the score of 0.916... without guesses mentioned elsewhere. I also got it at some point. I guess in the end my solution reaches the step requiring guesses earlier because of the time limit imposed on the individual backtracking runs (even if they would eventually confirm no solution, they don't do it within the chosen time limit).

What didn't work: I was hoping to be able to replace a part of the backtracking runs from step 4 by linear programming (ILP would have been too slow, I think). Basically, if a LP is infeasible then for sure the corresponding ILP has no solution either. I spent multiple hours to get the LP logic to work (I started from an incomplete version which didn't handle many cases) - and once I got it to work, it didn't help (it basically found real number solutions every time, even when integer solutions didn't exist).

During the contest I used seeds 1-1000 to check solution improvements. My average (absolute) score on these 1k seeds is 0.894497372.

Re: Post your approach (response to post by **jdmetz**) | Feedback: (+1/-0) | [+] [-] | Reply

Wed, Jan 6, 2021 at 12:08 PM BDT

dimkadimon
4499 posts

I am posting this on behalf of **Daiver19**, because he had problems posting it himself.

As everyone else, I'm creating the equation systems from the connected components of numbers with intersecting areas. I solve them by intersecting the binary masks satisfying the conditions (which is why I had to only use the equations with up to 20 variables, however that didn't make much difference). When I can't solve the system, I also add some randomly generated equations with different variable sets by adding/subtracting equations - this helps a bit for D=8+ (helps more than increasing the variable limit).

I've spent quite some time tweaking the guessing/stopping, especially for D=1, which seemed to have a lot of room for improvement (from the scores it seems like it was worth it). I ended up with the following approach:

At first, for small D we can easily generate all the possible solution sets, which allows us to be more precise about mine probability (this has been described by someone else, e.g. for 3 cells 101 has much lower prior probability than 010). So I always pick the lowest mine probability cell and have the size of the connected component of empty cells as a tiebreaker. I do that even when the probability of hitting mine is higher, than the probability of hitting mine by selecting a cell about which we have no information (it seems like information gain is worth the higher risk, but this could probably be optimized).

Now we need to decide when to stop. I look at how many guesses did I need to open/mark all the cells up to this point and compute the average number of cells per guess. Then I assign this number of cells-per-guess to all the future guesses, including the current one. Then I can compute the expected score for new guesses
`G=1..cells_remaining/cells_per_guess`
. For a given G the expected score is
`(cells_opened + G * cells_per_guess * empty_cell_fraction) * E[1 / (1 + mines_hit)]`
. The right part can be computed for a given G as
`sum from 0 to G of P(i_mines_hit) / (1 + mines_already_hit + i)`
, where P(i_mines_hit) is simply a probability of hitting i mines out of G guesses (binomial distribution). Obviously, we don't know the mine probability for the future guesses, so I just use the current mine probability, as hopefully it doesn't change much from guess to guess. In the end, I also tune down the expected score a bit (by 0.95). This is just a value which worked the best for me. If the maximum tuned expected score out of all G is greater than the current one, then I take a guess.