# Some Random Small (?) Problems

Quentin F. Stout
qstout@umich.edu

June, 2022

Here are some random problems where I know partial answers to some of the variants, and in some cases don't even know the most interesting questions. Feel free to contact me if you've got partial solutions and want to collaborate, or just publish your solutions and send me a pointer to the solution. However, I don't have any funding for people working on these problems so please don't request any. The problems and models are also slightly quirky so I'm not sure which conferences would accept solutions If relevant solutions have been published please let me know about them, I've circulated some of these questions for decades.

## 1 Stationary Resources and Growing Demand

This is a problem I long ago called the *well problem*, but it could just have easily been called the firehouse or school building problem. In the simplest version, suppose everything is occurring on the unit interval (0,1). People will be arriving, no one leaves, and they share resources equally, so that if there are $k$ people then the interval is broken into $k$ pieces of equal size, $(0, \frac{1}{k})$, $(\frac{1}{k}, \frac{2}{k})$, $\ldots$ $(\frac{k-1}{k}, 1)$ and each person gets one interval. They're open intervals so that there aren't any arguments about who owns the endpoint. However, everyone needs a well, and wells don't move. When there are $k$ people you want there to be exactly $k$ wells, where each person has a well on their interval. As each person arrives where should you drill the new well so that this is always true?

For example, suppose the first person arrives and digs a well at 0.65. Then a second person comes and a new well is needed in $(0, \frac{1}{2})$, so it is dug at 0.4. Then a third person arrives. Unfortunately since there are 2 wells in $(\frac{1}{3}, \frac{2}{3})$ there is no well in $(0, \frac{1}{3})$ nor in $(\frac{2}{3}, 1)$. Thus you have to drill 2 wells, not 1.

However, if the second well had been at $\pi/10$ then when the third person came drilling one at 0.75 would work. In fact, it is easy to extend this when the 4th person comes, drilling a new one at 0.15. However, when the 5th came then there is no well in $(\frac{2}{5}, \frac{3}{5})$ and none in $(\frac{4}{5}, 1)$. You can easily go back and change some of the earlier decisions to make everything work through the first 5 people, but you should have looked ahead and seen that this problem was coming.

Can you dig wells so that as every new person arrives only 1 more well is needed? If you put the wells at locations that are rational numbers then obviously it is false because any such well will eventually be on the dividing line between two intervals. You could fix that by saying the original interval, and all subintervals, are half-closed, containing the left endpoint. E.g., [0,1), then $[0, \frac{1}{2})$ and $[\frac{1}{2}, 1)$, etc. Does this make it possible? (Hint: no, but prove it.) What if you use locations at irrational numbers?

Suppose you weaken the objective and want to drill only 1 new well every time, and are satisfied if each person has an interval of size at least $\alpha/k$, for some positive real number $\alpha$. What is the largest possible $\alpha$? Or suppose you keep the constraint that each person gets an interval of size $1/k$: must there be infinitely many times when you have to drill 2 wells? Can you arrange it so you never have to drill 3 or more?

I first encountered this when listening to a talk about allocating parallel computing resources and realized this is what the speaker actually wanted. However, it comes up in more common settings, such as building fire stations and schools, knowing that the community will expand. In Ann Arbor, where I live, the first high school was built in the middle of the city, which was optimal assuming the population density was the same throughout the city (it isn't, but ignore that). As the population grew they needed a second high school (and then a third, ...). Where should it go? First, what should the criteria be? Perhaps all students go to the closest school and you want to keep the number of students at the schools the same (i.e., if the population is uniformly distributed then in the Voronoi diagram of the schools' locations all polygons have the same area). Or should you minimize the worst distance any student travels? Or ... ? I once had a fire chief in California contact me about where to locate firehouses.

Other possibilities to investigate are models where population growth isn't uniform throughout the region, or the region is higher dimensional or a graph.

## 2 Hungry Turtles with Pebbles

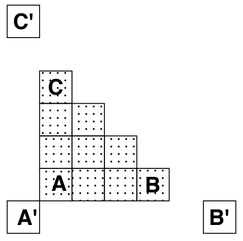
Here everything takes place on a $d$-dimensional grid, infinite in all directions, where each dimension is indexed by the integers There will be some turtle food at one of the grid locations and a turtle is trying to find it. However, the turtle has just a finite memory (i.e., is a finite automaton) and all the locations look the same, except for those that have food. Further, the turtle cannot leave marks at the locations to keep track of where it has been. Tthis would allow it to act like a Turing machine, and it could easily locate the food. However, it does have a little help: it has a few pebbles. It can leave them at locations, and when it returns to a location with pebbles it can pick some or all of them up and carry them. The question is: for a given dimension $d \geq 1$ what is the minimum number of pebbles necessary for the turtle to be guaranteed to find the food, and what is an algorithm it should use?

Note: one way to think about finite automata is that they are a computer with a fixed number of bits of memory. You can write a program for them, but it can't have things such as counters, stacks, queues, arrays, or recursion levels that are arbitrarily large.

$\mathbf{d = 1}$ If the turtle has no pebbles then there are only 3 types of exploration behavior it can. Either there is some $a \leq b$ so that the turtle only visits all locations in $[a, b]$, perhaps visiting some infinitely often (at some point it may just halt); or there is an $a$ such that the turtle only visits all locations in $[a, \infty)$, visiting none of them infinitely often; or there is an $a$ such that the turtle only visits all locations in $(-\infty, a]$, visiting none of them infinitely often. That these are the only possible behaviors follows from the fact that the turtle has a fixed amount of memory and if it keeps seeing the same input (an empty location) it goes into a cycle. With a little more work one can show the same is true if it has 1 pebble. Since these behaviors don't visit all locations it is possible the turtle doesn't find the food.

However, with 2 pebbles it can explore the entire line, ultimately finding the food. Basically, if the turtle just dropped a pebble at $a$ and there is a pebble at $b$, $b > a$, the turtle starts going right until it encounters the pebble at $b$, picks it up, goes one more location to the right, drops it, and turns and starts going towards $a$, where it will follow similar behavior, moving that pebble one position to the left. If the food is $x$ locations away from where the turtle started it will take $\Theta(x^2)$ time for the turtle to ultimately find it.

$\mathbf{d = 2}$ With some work it can be shown that even 2 pebbles is not enough, but 3 suffices. A cycle of the turtle is shown below, where the turtle goes from pebble A to B to C to A .... Suppose the turtle has just gone from from C to A. It then moves pebble A one down and one to the left (location A'), then goes back to where A was and goes right until it encounters B. It moves B one down and two to the right (location B'), then goes back to where B was and zig-zags towards C, going left one, up one, left one, .... When it encounters pebble C it moves it up two and left one (location C'), then starts down towards pebble A (now at location A'), and the cycle repeats.



$\mathbf{d = 3}$ Given the algorithm for $d = 2$, showing how to use 4 pebbles for $d = 3$ isn't too difficult, but the interesting thing is that 3 suffice. I'll leave that up to you.

$\mathbf{d \geq 4}$ ??

What if there are simple obstacles, such as disjoint rectangular regions where each location has an obelisk and the turtle can't reach up to put a stone there?

Note that for any dimension, if the turtle didn't use the pebbles but instead just took a random walk, where at each step it is equally likely to go to one of the $2d$ neighbors, independent of all other moves, then almost surely (i.e., with probability 1) it will eventually find the food. However, this requires some way to obtain independent uniformly random numbers, something it can't do on its own.

## 3 Dimensionless Cellular Automata in High Dimensional Space

Cellular automata (CA) are copies of the same finite state automata, perhaps with different initial states, forming some sort of graph, where the next state of an automaton depends on its current state and the states of its neighbors. Usually they are connected as 1-, 2-, or 3-dimensional grids, either finite or infinite, but might be in a configuration such as a binary tree, or more exotic structures such as cube-connected cycles, where each vertex of a hypercube of dimension $d$ is replaced by a cycle of $d$ automata, each of which has 2 neighbors in the vertex's cycle, and 1 neighbor which represents an edge of a standard hypercube. Typically CA are synchronous. A good example is Conway's Game of Life, which is easy to find on the web.

The turtle problem mentions that you can think of a finite automaton or finite state machine as a program with a fixed number of bits of memory. For cellular automata the state of an automaton can depend on the states of its neighbors, which in the programming view just means that the automata have variables whose value can be read by their neighbors. For CA an important aspect is that the finite state automata does not change no matter how large the CA is, and hence for sufficiently large CA the automata don't have enough memory to store their coordinates, just as the turtle couldn't remember where it had been. However, sometimes they can work in groups to store coordinates (see my paper "Using clerks in parallel processing" *Proc. FOCS* 1982, pp. 272–279).

What if the automata are in a high-dimensional grid? Now it may not be reasonable to say an automaton's next state depends on the states of all of its neighbors, so suppose it depends on a small subset, such as the two adjacent ones in a fixed dimension, but the automaton can change which dimension it is using to define its neighbors, i.e., which of the adjacent automata in the grid it is reading values from. For example, in a 2-d mesh it might pick the 2 horizontal neighbors, or the 2 vertical neighbors (there may or may not be boundaries depending on whether the dimension is infinite in each direction). However, if the dimension is high then we assume no automaton can store the number of dimensions so we want an automaton that can work no matter how many dimensions there are.

An extreme model is that it reads the state of neighbors in one dimension and then for the next step can switch to the next or previous dimension, e.g., if it is using the states of its neighbors in dimension 17 then in the next step it can then read the states of the two neighbors in dimension 16, or in 17, or in 18. This is in the same spirit as cellular automata only being able to use the states of their neighbors at any one step, and only after multiple steps can they obtain information from far way. Further, in this model assume at most 2 neighbors can read its state (they may not be the neighbors it is reading from). If more than 2 try then all but 2 fail and which 2 succeed is arbitrary, not under the control of any automaton. Or you could be more draconian and say if more than 2 try then all fail (these variants are similar to different versions of concurrent read for PRAMs), but that might make some algorithms far more difficult. Note that the dimension is not part of the state of the automaton, just as the position of a Turing Machine's read/write head is not part of the state of the TM (it too is a finite state machine) and the head can move left or right (or perhaps up and down as well, if it is on a 2-dimensional tape) from wherever it is (unless it is at an end of the tape). Thus the automata I'm describing don't have an intrinsic dimension, so I'll call them *dimensionless*, even though as a group they form a multidimensional grid. Initially every automaton starts using the states of its neighbors in dimension 1.

Note that the cube-connected cycles mentioned above was a different attempt to address high-dimensional connectivity.

I haven't yet tried to do any interesting algorithms yet, and there are many additional variations. For example, you might want use a slight variant where when an automaton is reading the states of neighbors then it knows if they are reading its state. Another variant is that its state depends on a neighbor in dimension $x$ and a neighbor in dimension $y$.

These dimensionless automata are independent of dimension in one sense, but can control going through dimensions. One type of problem that may be reasonable to solve with them is one where solutions are known for $d$-dimensional automata for all $d$, where the changes in the algorithm from one dimension to another aren't very dramatic. For example, sorting fits this bill. Suppose the strings to be sorted each start and end with the special symbol #, and they are stored in a $d$-dimensional cellular automaton, for some $d \geq 2$, of dimensions $n \times n \ldots \times n$, where $d \ll n$. Thompson and Kung, "Sorting on a mesh-connected computer", *Comm. ACM* 20, 1977, pp. 263–271, showed that if the strings were words and each processor had enough memory to store a few words, along with $d$, the processor's coordinates, and a few other variables, then the strings could be sorted in $\Theta(n)$ time, where the implied constants depend on $d$ (this also assumes word comparisons can be done in unit time). If all the strings are the same length it should be possible to do the same on a dimensionless CA, though perhaps with a time dependence on the strings' length as well.

There are many problems where a basic algorithmic approach gives an optimal solution for all dimensions when the dimension was known by the processors, so perhaps most of these can be extended to dimensionless CA. In *Parallel Algorithms for Regular Architectures: Meshes and Pyramids*, MIT Press, 1996, the chapters on meshes have many such problems.

There are other problems where the input might be just a part of a much larger grid. For example, suppose you have an $n \times n$ 2-d CA, where each automaton is initially white or black. Beyer (1969) and Levialdi (1972) independently discovered that if this is interpreted as a maze where the black squares are obstacles, and if one white square is marked "start" and another marked "end", in $\Theta(n)$ time the CA can decide if there is a path from the start to the end (references and the algorithm can be found in section 3.3.1 of the book mentioned above). The algorithm is very simple and elegant, but has an unusual aspect: it proves there is a path, but doesn't find any path. In fact, the shortest path might be a swirl of length $\Theta(n^2)$, and a breadth-first search can be used in a regular 2-d CA automaton to find a shortest path in this time. Further, no faster algorithm is known, nor is there a proof that no such algorithm can exist. However, if the maze is the corner of a slightly larger 2-dimensional grid where each side is at least $n\lceil\sqrt{\log_2 n}\rceil$ then a shortest path can be found in $\Theta(n\sqrt{\log n})$ time. If it is in a 3-d CA of size $n \times n \times \lceil\log_2 n\rceil$ then it should be possible to find a shortest path in $O(n \log n)$ time, and perhaps in $\Theta(n(\log n)^{1/3})$.

For mazes in dimensions $\geq 3$ the approach used by Beyer and Levialdi to decide if there is any path cannot be used. The clerks paper mentioned above shows a different, far more complex, approach that can be used, and presumably a dimensionless CA version can be created. However, it too does not find any path. If the maze is a $d$-dimensional block where each size has length $n$, stored in a corner of a CA, then a dimensionless CA should be able find a shortest path in $O(n^{d-1} \log n)$ time if it is a $c$-dimensional grid where $c \geq d$, each dimension has extent at least $n$, and the product of all of the extents is at least $n^{2d-2} \log n$. I haven't checked all the details, so perhaps I'm wrong, and the algorithm would definitely be complex. Probably one of the first things the dimensionless CA needs to do is determine the dimensions of the maze and the grid.

You may think of high dimensional space as being absurd, but my math thesis was about infinite dimensional Hilbert space so it makes sense to me. In classical physics most state spaces have a very high dimension, sometimes treated as being infinite, and in quantum physics dimension and distance aren't nearly as important as they are in classical physics. As I mentioned at the start of this collection of problems, some of this is quite quirky and it may take some effort to get conferences to accept papers on dimensionless CA. If the model is extended to include some randomness there might be some interesting properties in terms of expected behavior.