

Hybrid Strategies: Brute Force paired with Greedy

There are some problems where neither a straight brute force, nor straight greedy strategy will suffice. Sometimes, people will call this type of hybrid strategy, "Meet in the Middle", but for the two problems presented in this lecture, they don't necessarily fit into that category exactly.

Welcome Party

We'll see this problem again when we cover Network Flow (which handles larger problem instances) but the bounds on this problem specifically allow for a nice solution which pairs both a brute force and greedy strategy.

The problem is as follows:

You have upto 300 people at a conference. Each person has a first and last name. Here is an example of people at a conference:

Richard Hamming
Marvin Minsky
John McCarthy
Edsger Dijkstra
Donald Knuth
Michael Rabin
John Backus
Robert Floyd
Tony Hoare

You want to arrange for some tents for the people at the conference to gather. Each tent will either be a "first name tent" or a "last name tent" and each tent will be assigned a letter. All first names will start with letters in between 'A' – 'Z' and all last names will start with letters in between 'A' – 'R'. You must arrange for tents so that everyone at the conference has at least one tent they can go to.

In the example above, one possible solution would be

First Name Tents: R, M, J, E, D (first eight people have names that start with these letters)

Last Name Tents: H (last person can go to this tent)

The goal of the problem is to minimize the number of tents. It turns out that the solution above can not be beat.

A fairly terrible brute force solution would be to try assigning each person to one of two tents and then count how many tents there are. Since there are 300 people, the number of steps this would take is $2^{300} \times 300$, roughly since there are 2^{300} ways to assign 300 people to tents (each person could go to a first name tent or a last name tent), and then for each assignment, you have to go through and count how many tents there are which means iterating through each name.

A better brute force solution would be to try each subset of 2^{26+18} tents and just go through and see if every person has a tent to which they can go. This would take roughly $2^{44} \times 300$ simple steps. (Recall that the +18 is because the only valid letters for the last name are 'A' – 'R'.)

Now, let's think a little bit about this weird restriction on the last name...there are only 2^{18} possible subsets of tents we could create for last names. This number is small enough (a bit more than 250,000), that we could manage iterating through that many subsets fairly quickly. To simplify matters further, let's say that we've picked in advance that the only last name tents we'll have are M, H and R (using our original example):

Richard Hamming
Marvin Minsky
John McCarthy
Edsger Dijkstra
Donald Knuth
Michael Rabin
John Backus
Robert Floyd
Tony Hoare

If that were the case, then the five individuals in yellow would have a tent to go to, so we only need to worry about the four others. Since we've already decided that we won't make any more last name tents, we're forced (greedy selection) to make the following first name tents:

E, D, J, R

Notice that once we "fixed" our last name tents, our selection for the rest of the tents could easily be done greedily since that was the only option left for those who aren't covered. This greedy algorithm takes roughly 300 steps, which isn't so bad. So if the last name tents were forced to be M, H and R, then the best we could do is use 7 tents. Now, to get the final algorithm, just recognize that we just have to try each possible set of tents for the last names.

So, here's the hybrid algorithm:

1. Set our initial answer = 18 (a tent for every last name letter)
2. For each subset of last name starting letters:
 - a. For each name in the list
 - a1. If this person's last name starts with a letter in the subset, we're good.
 - a2. Otherwise, add the letter this person's first name starts with to a set.
 - b. Compute the number of last name tents plus the number of first name tents created in a
 - c. Update answer to equal the answer from (b) if (b) is smaller than the previous answer.
3. The answer stored is the fewest number of tents possible.

This is a common solution idea. If we simplify one portion of the problem by "fixing" it, then solving that easier version of the problem might become easier. Then, we can brute force all possible choices of the "fixed" portion of our solution!

Walls

Here is a link to the full problem description:

<https://www.cs.ucf.edu/~dmarino/ucf/cop3503/extraprogs/Hybrid-Walls/Walls.pdf>

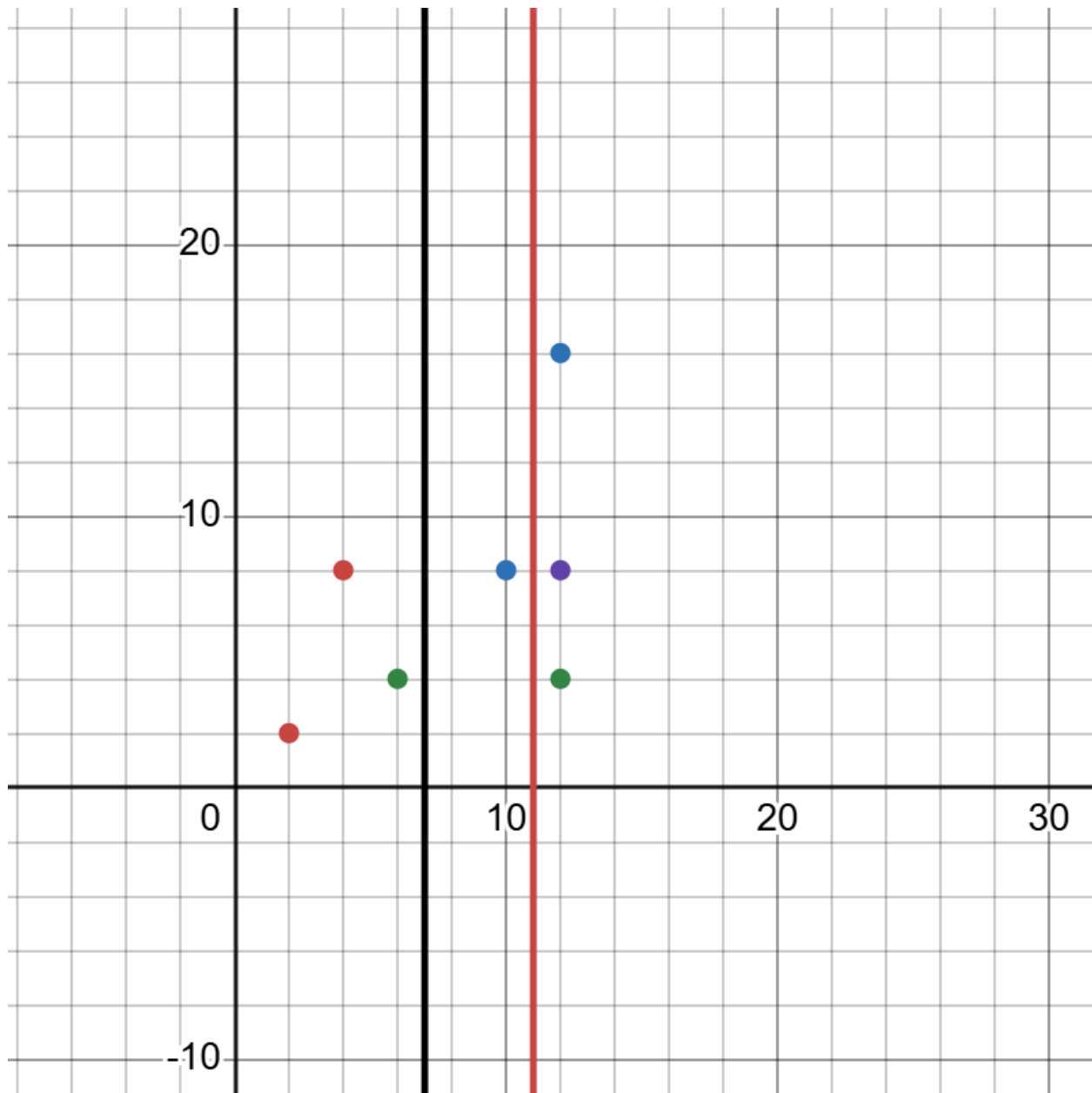
Again, notice the interesting bounds. You can draw at most 18 horizontal walls ($y=1, y=3, \dots, y=35$) and at most 18 vertical walls ($x=1, x=3, \dots, x=35$). If we tried all 2^{36} combinations of walls, we could then test if each point was in its own compartment, but this run time would be prohibitive, at least $2^{36} \times 100$ simple operations.

But, just like the previous problem, imagine trying each combination of vertical walls. There are 2^{18} of these. Once we do this, we can immediately look at each rectangular region (up and down) separated by these vertical walls, and our goal would be to draw in each required horizontal line...don't draw one until you're forced to (this is the greedy part).

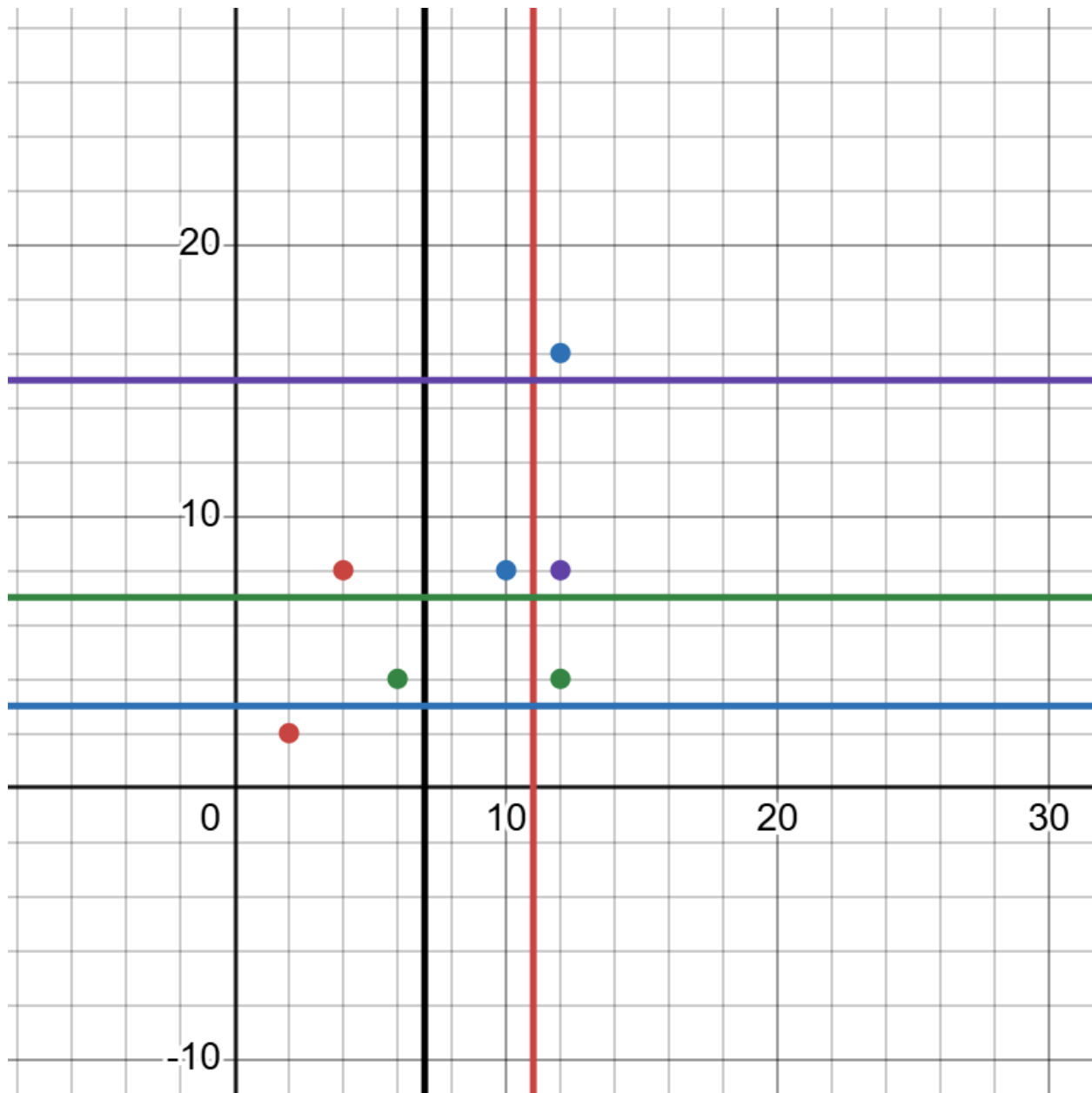
Consider the following example:

```
2 2
4 8
6 4
10 8
12 4
12 8
12 16
```

Now, let's imagine that we've decided to draw the vertical lines $x = 7$ and $x = 11$. Here is the current state of affairs:



Now, we start at $y = 1$, and keep advancing until we get to a point where 2 points are in the same region. This occurs at $y = 5$, meaning that we must add in the line $y = 3$. This will then ensure that $(2, 2)$ and $(6, 4)$ are in different regions. The next line we'll be forced to draw is $y = 7$ because without it $(6, 4)$ and $(4, 8)$ would be in the same region AND $(12, 4)$ and $(12, 8)$ would be in the same region. The last line we'd be forced to draw is $y = 15$, so that $(12, 8)$ and $(12, 16)$ aren't in the same region. Here is our final picture of the optimal set up when we draw the lines $x = 7$ and $x = 11$:



Thus, for this set of vertical lines, the best answer is 5. For this particular input set, there are only 4 meaningful vertical lines to draw since there are only 5 unique x values amongst all of the points. Thus, if we tried all possible subsets of these 4 lines, and for each subset greedily draw the horizontal lines, we can just take the best answer of each of these possibilities. Note that for some subset of vertical lines, no answer might occur. For example, if we only drew $x = 7$ as the only vertical line, it would be impossible to separate $(10, 8)$ and $(12, 8)$. In fact, the ONLY way to separate these points is to have $x = 11$, since no horizontal line can separate those two points.

To ease implementation, the posted solution just automatically tries all 2^{18} subsets of all vertical lines, even if some of those subsets would never produce the optimal answer.