Greedy Approach for Solving Interval-Covering Problems

Salvian Reynaldi 13511007
Computer Science / Informatics
School of Electrical Engineering and Informatics
Institut Teknologi Bandung, Ganesha 10 Bandung 40132, Indonesia
salvianreynaldi@students.itb.ac.id

***Abstract* — an algorithm is a number of structured and well-ordered steps that can be used to solve problems. In the Computer Science subject area, the problems often involves computing. Those algorithms can be classified into some categories. By its design paradigm, algorithms are classified into the Brute Force Algorithm, the Divide and Conquer Algorithm, the Greedy Algorithm, and the Dynamic Programming Algorithm. On harder problems, it is usually not enough to use just one of the paradigms to solve the problems. Usually combining these techniques are required in order to achieve the desired solution or to get a more efficient solution. However, Greedy algorithm paradigm has been used to solve some famous problems in Computer Science. Some of the examples are the fractional knapsack problems, the load-balancing problems, Huffman-code compression algorithm, the Shortest Path problems, and the interval-covering problems.**

***Index Terms* — algorithm, Computer Science, greedy, interval covering.**

I. Introduction

One of the well-known problems in Computer Science subject area is the Interval-covering Problem. The core directive in the Interval-covering Problem is to cover as many areas as possible with as few covers as possible. A cover here is defined as anything that can extend over the area or has a covering radius/distance over some area. Actually, the application of interval-covering problems can be easily found in real-life. Many of these applications involve “installation problem”, such as the tower installation problem, the lighting installation problem, etc. An example of the interval-covering problem application that does not involve installation is the garden watering problem.

On the other hand, there are many algorithms that are already invented. By its design paradigm/methodology, some of the algorithm classifications are Brute Force, Divide and Conquer, Greedy, and Dynamic Programming algorithms. Brute Force (a.k.a. Complete Search) Algorithms is a method for solving a problem by traversing the entire part search space to obtain the required solution [1]. In some smarter variants, usually brute force algorithm involve some pruning (deliberate act of not exploring) parts of the search space so that the algorithm can be more efficient / run faster. Pruning is done if it is clear that some parts have no possibility of becoming the desired solution. Divide and Conquer is a method in which a problem is made simpler by ‘dividing’ it into smaller parts and then conquering each part. Dynamic Programming also make problems simpler by breaking them into smaller sub problems (as in Divide and Conquer), but usually the problems has unique characteristics, i.e. they usually have both optimal sub structure and overlapping sub problems. Greedy algorithm paradigm will be described in the next section.

II. Greedy Paradigm

Greedy algorithm is usually used for optimization problems. There are many optimization problems that can be solved by the Dynamic Programming technique, but sometimes using Dynamic Programming to solve those problems is overkill. Greedy algorithm usually is not only a simpler but also a more efficient option for solving those problems. It is usually easier both to think and to implement Greedy algorithm, plus it typically run fast. According to [1], an algorithm is said to be greedy if it makes the locally optimal choice at each step with the hope of eventually reaching the globally optimal solution. In another words, Greedy algorithm always makes the choice that looks best now [2]. Agreeing with [1] and [3], problems that can be solved by Greedy Algorithm Paradigm usually have special characteristics, explicitly they usually have:

- Optimal sub structures.
  It means that for every greedy problems, the optimal solution to the problem embraces optimal solutions to its sub problems. In other words, the solution to a problem can be determined from the optimal solutions to its sub problems.
- Greedy choice property.
  It means that by choosing the best-at-the-moment choice, the optimal solution can eventually be reached without having to look back / reconsider previous choices that have been made (making greedy choice at every step will generate the desired optimal solution). This characteristic distinguishes the Greedy paradigm from the Dynamic Programming paradigm. In the Dynamic Programming technique, we often reconsider the previous choices that have been made when we are about to make a present choice. That characteristic is usually called overlapping sub problems.

However, in Computer Science, greedy heuristic strategy is not (always) a good approach to solve optimization problems. If a problem does not exhibit one of the two characteristics mentioned above, the greedy algorithm may not work. The difficult part in using Greedy to solve problems is to prove the second characteristic mentioned before. Greedy algorithms are infamous for being tricky, that even missing a very small detail can be fatal[1]. Nevertheless, Greedy algorithm has been proven able to solve many optimization problems, such as activity-selection problems, fractional knapsack problems, load balancing problems, and interval-covering problems.

III. INTERVAL-COVERING - PROBLEMS

Let us start with a simple example. The problem here is titled Scarecrow; it is taken from [4], problem number 12405. Here, there is a man that has a very long field for farming that can be modeled as a $1 \ x \ N$ grid. Some parts of the field are fertile meanwhile the others are infertile. However, the area is full of crows, and the man fears that they might feed on most of the crops, so he decide to place some scarecrows on the field. When placed to a spot, a scarecrow can cover the cell to its immediate left and right as well as the cell it is on itself. The goal is to set the number of scarecrow that needs to be placed so that all the fertile sections of the field are covered. A real-life application is the lighting installation on a narrow road. To be as efficient as possible, of course we want to set up as few lamps as possible.

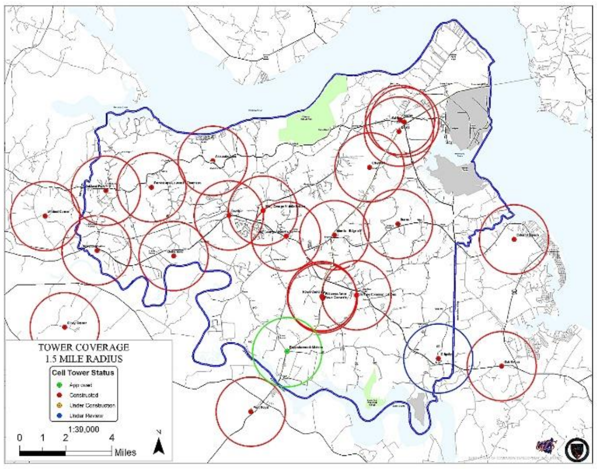
Scarecrows. Source: Wikipedia.

Another example of interval-covering problem is titled Watering Grass; it is taken from [4], problem number 10382. Here, there is an $l \ x \ w$ strip area of grass, and also some sprinklers. Each sprinklers have their own radius of operation, their own distance from the left end of the strip, and are placed in the middle of the strip. The operation radius of the sprinklers may overlap each other. The first goal is to decide whether it is possible to water the entire strip with the sprinklers. If it is possible, determine the minimum number of sprinklers needed to do so (this is both a decision and an optimization problem at once). This problem can actually be directly applied in real life. We want to be as efficient as possible (thus saving not only electricity power but also water) when we are watering the garden. In order to achieve that, we need to turn on as few sprinklers as possible (although in real life sprinklers may have same operation radius, so the problem may be not about which sprinklers to turn on, but about how to place sprinklers smartly).

Sprinklers. Source: http://www.temeculavalleyirrigationsystems.com/home/

One more example of the Interval-covering Problem is titled the radar installation problem. Here, we are given some number of small islands; each has its distinct location. There are also some radars with same covering radius. The radars are to be placed somewhere in the coast (it is assumed here that the coast is a straight line). We have to determine the minimum number of radars that needs to be set up so that all the islands are covered by those radars. Real-life application of this example is actually obvious, like the previous example, it can be directly applied. Assume a new mobile network operator (a.k.a. wireless service provider / wireless carrier) just started their service in Indonesia. The first goal they will want to achieve is usually to cover as many areas as possible / every single island in Indonesia, so that they can provide their wireless communications service to as many people as possible. However, with just “a few” budget (given that they have just been established), they’ll more likely to set up as few BTS (Base Transceiver Station) towers as possible while achieving the goal.

[1] TC



Radar Coverage Illustration. Source: http://cdn.blogs.fredericksburg.com/kinggeorge/files/2011/06/Cell-Tower-Coverage2.jpg

IV. Interval-covering - Analysis

A. The Scarecrow Problem

Assume the garden layout is like this:

```
.#.....##
123456789
```

Here, a dot denotes the fertile part while a hash denotes the infertile part of the field. In addition, assume that the Scarecrow coverage is like this (three-unit long):

```
XXX
 S
```

A brute force approach for solving this problem is to test all the 2^N combinations of placing scarecrow. Of course, it is not very efficient. However, since the coverage scarecrows is three unit long, it is intuitive that the maximum number of scarecrows needed for the N unit long field is $\lceil N/3 \rceil$; That is, we cover each three unit from the left end to the right end with a scarecrow. Remember that the field has some infertile parts in it; thus, to cover the infertile parts of the field is actually unnecessary.

For the greedy algorithm to work, let us first reveal the optimal sub structure and greedy choice characteristic out of this problem. Actually, it is clear that this simple problem exhibits those characteristics. To emphasize it, let us start from a smaller example.

For $N = 1$, the optimal number of scarecrows to be placed in the field is 1. We place it on position 1.

```
S
1
```

For $N = 2$ or $N = 3$, the answer is still 1, because we can place the scarecrow on position 2; it can still cover position 1 (and position 3 as well).

```
XSX
123
```

For $N = 4, 5, 6$, it is just the same as if $N = 1, 2, 3$; we can independently place another scarecrow on position 4 (or 5), without caring where the previous scarecrow is placed. Thus, this problem has the greedy choice property. Assume there is another way to place a scarecrow on a three-unit long field (not on position 2), i.e. if the field has an infertile part:

```
#XS
123
```

Then the optimal solution for six unit long field will be like this:

```
#XSXSX
123456
```

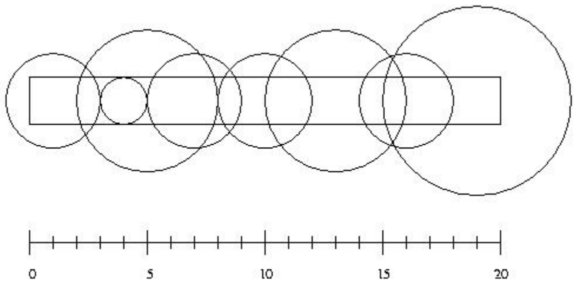
Or like this:

```
#XSXXS
123456
```

Since the solution for $N = 6$ field contains the solution of the $N = 3$ field, we can say that this problem exhibit the optimal sub structure characteristic. It will always contain optimal sub problems solutions, because if it did not, we would choose that more-optimal sub solution when we are making the global solution.

So, the greedy strategy for solving this problem is like this. From the left end position, see if that unit is fertile or not. The main step is: if the unit is fertile and not covered yet, place a scarecrow on the next-to-the-right position; if it is not, skip the current unit. After that, move to the next unit. Repeat the main step, until we reach the right end of the field.

B. The Watering Grass Problem.

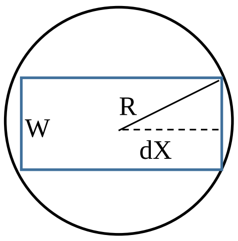


Problem illustration. Source: [4].

Assume there's a garden which length is 20, width is 2, and has eight sprinklers. The sprinklers are on this position (sorted by increasing x_i):

1) 1, its operation radius is 2,
2) 4, its operation radius is 1,
3) 5, its operation radius is 3,
4) 7, its operation radius is 2,
5) 10, its operation radius is 2,
6) 13, its operation radius is 3,
7) 16, its operation radius is 2, and
8) 19, its operation radius is 4.

Brute Force strategy, that tries all possible subsets of sprinklers to be turned on, is definetely infeasible to try, because there will be 2^N possible subsets of sprinklers. If we observe this problem further, actually, it is just like the previous example (Scarecrow), but this time the coverage of each scarecrow is different, and the coverage area is a circle, not a rectangle. Let us try transforming this problem into a normal interval-covering problem.



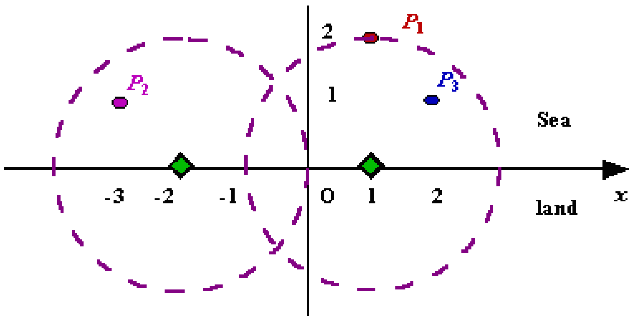
If we transform the sprinkler coverage from a circle into a rectangle (that has the same width W as the grass strip), the effective rectangle coverage area is shown as in this figure on the left. It is a rectangle which length is$2dX = 2\sqrt{R^2 - \left(\frac{w}{2}\right)^2}$, while the rest $(R - dX)$ area will not be completely covered by the circle.

After that, let us transform the entire sprinkler operation radius into rectangles, so that the coverage area of each sprinkler will be$[xi - dx\,..\,xi + dx]$. Now it becomes a normal interval-covering problem. Interval covering problem has been proven to have the optimal sub structure (the optimal solution for $l * w$ garden contains the optimal solution for $(l-1) * w$, $(l-2) * w$, … garden) and the greedy choice property (to choose whether a sprinkler should be turned on or not is independent one another) characteristic.

To solve this kind of interval-covering problem, first we sort the sprinklers by increasing left-end-coverage and by decreasing right-end-coverage if ties arise. After that, from the left to the right we continuously choose a sprinkler that covers "as far right as possible" and can still cover the current uncovered position. For example, if we use the problem illustration case above, our greedy strategy will choose the first sprinkler, skip the second sprinkler while choosing the third sprinkler (because the second sprinkler dX is 0 $\rightarrow$ it is too small to cover the grass strip). Later it will choose the 4th, 5th, and 6th sprinkler, and skipping the 7th sprinkler while choosing the 8th sprinkler (because the 8th sprinkler is more far right than the 7th semester).

(1)

C. The Radar Installation Problem



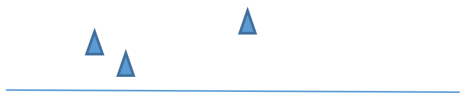
Because the coast line is an infinite straight line, let us assume that the coast line is the x-axis, and the islands are located above the coastline. Firstly, soon we will realize that the radars should be installed exactly on the coastline, not below the coastline, because it will be not very useful if we install it below the line; the islands are above the line, so in order for the radar to reach as far top as possible, installing the radars "as top as possible". So that, we will be more likely to cover more islands.

Secondly, let us try the Brute Force Approach for this problem. The naïve brute force approach that tries to set up radars in every possible places will not work, because in this case the possible places are not only integer-numbered positions, but also real-numbered positions. Another smarter brute force approach is to try placing radars on every possible subsets of islands. There will be 2^M possible subsets, so does the time complexity of the solution.

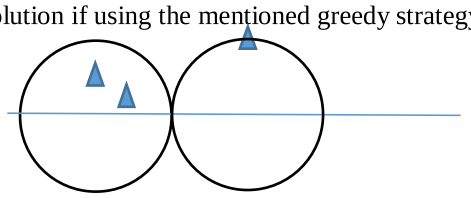
Now we know already that the interval-covering problem is a greedy problem. This radar installation problem is actually an interval-covering problem, too. It tries to cover as many islands as possible with as few radars as possible. So let us try a greedy approach to solve this interval-covering problem.

Let us try three distinct greedy strategies. The first greedy strategy is to set up radars from the left to the right so that a radar is assigned to each island, also from the left to the right, if the island is still uncovered. In addition, that radar is actually placed on position Xi (the same position as the island's absis coordinate). See the figure below.

Islands' location:

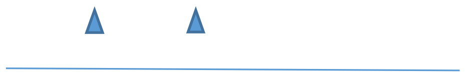


Solution if using the mentioned greedy strategy:

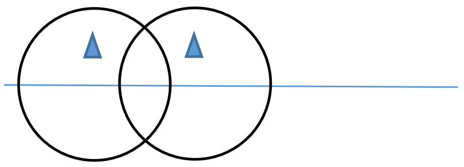


However, this greedy strategy is not always optimal. Let us observe this case:

Islands' location:

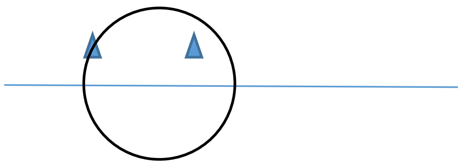


Solution if using the mentioned greedy strategy:



It uses two radars to cover two islands. However, it is actually possible to set up just 1 radar to cover them all, like this:

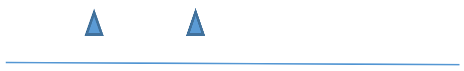
The optimal solution:



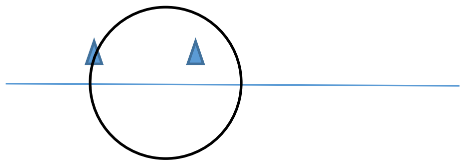
So this greedy strategy is proven to be not optimal.

The second greedy strategy is to scan from the left to the right, assigning a radar to any island, as long as the island is uncovered, so that the island will be on the edge of the coverage radius of the radar. Let us see the figure below.

Islands' location:

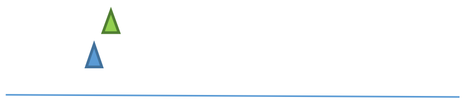


Solution if using the second greedy strategy:

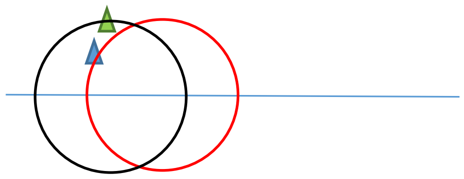


Of course it is optimal. It was the example that we used to prove that the first greedy strategy is not an optimal one. However, this (the second) greedy strategy is not the best, either. See the figure below.

Islands' location:

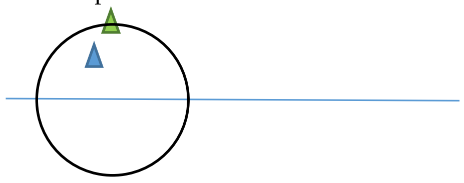


Solution if using the second greedy strategy:

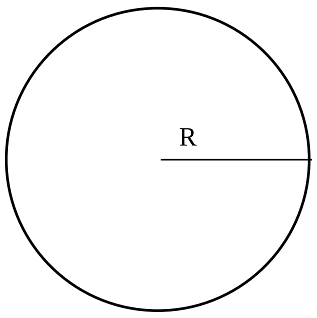


We force the island to be on the edge of the radar's coverage radius and we scan from the left to the right. Because of that, we will begin with the blue island. We assign one radar to cover this island, the red radar. However, the red radar is not able cover the next island, the green island. So we assign another radar, the black radar, to cover this island. Actually, the optimal solution is to just use one radar to cover those islands.

The optimal solution:

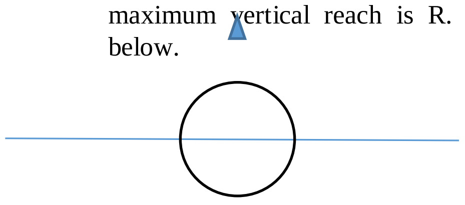


So, again we prove that this greedy strategy is not an optimal one, either. Many other greedy strategies can be tried to solve this problem, but let us go directly to the optimal solution by doing some deeper analysis.

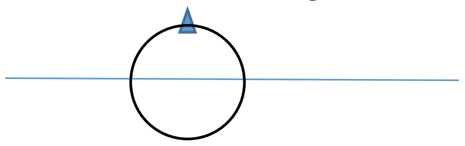


In this problem, the radius coverage of the radars are the same. Let us say that the radars' radius coverage is R units, and the islands' location coordinates are denoted with Xi, Yi. Since the coast is described as an infinite straight line (the x-axis), soon we will notice that there are three possible cases of Yi.

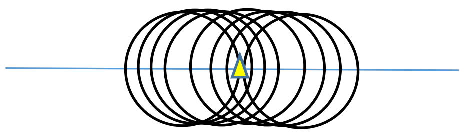
- Yi is greater than R. There will be no radar that is able to cover that island, because the radars' maximum vertical reach is R. See the figure below.



- Yi is exactly R. There should be a radar that is placed *at* Xi, so that this particular islands can be covered. See the figure below.



- Yi is less than R. There are some possible spots to set a radar up. Specifically, a radar can be installed on position $[Xi - dx \,..\, Xi + dx]$ in order to cover this island. See the figure below.



What is dX? Simillar to the previous example (the Watering Grass Problem), $dx = \sqrt{R^2 - Yi^2}$. To solve this problem, first transform all the islands' coordinate from Xi, Yi to Ai, Bi, where $Ai = Xi + dx, Bi = Xi + dx$. After that, sort the islands by increasing Bi, or by increasing Ai if ties arise.

Next thing to do is to set up radars from the left to the right. From the first island, we put a radar as far right as possible but it should be still able to be cover the current position.

Look at the pseudo-code below.

```
i:=1;ans:=0;
while (i<=n) do
  begin
    j:=1;ans:=ans+1;
    while (j<=n) and (x[j]<=y[i]) do
      j:=j+1;
    i:=j;
  end;
```

V. Conclusion

For some Computer Science optimization problems, solving them with the Dynamic Programming technique is usually too much. In addition to be a bit slower, a Dynamic Programming solution also tends to be harder / far more complex to implement than the Greedy one. However, Greedy Algorithm requires a problem to have two main characteristics in order to work; the optimal sub structures and the greedy choice property. That is actually the difficult part of implementing a greedy solution: to prove that the problem has those characteristics; without doing this proving, using greedy paradigm to solve problem is actually risky.

VI. Acknowledgment

I want to thank

- God, for giving me strength and blessing so I am able to finish this paper.
- Dr. Ir. Rinaldi Munir, the lecturer of the IF2211 course, for making algorithms more interesting by his inspiring lectures so that we students enjoy attending the class every Monday and Wednesday.
- My IF2211 AY2013/2014 friends, for further explanations about greedy paradigm so I can understand this algorithm paradigm better.

References

[1] Halim, Steven and. Felix Halim. *Competitive Programming 3: The new Lower Bound of Programming Contests*.

[2] Cormen, Thomas H.; Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford. Introduction to Algorithms (3rd ed.). MIT Press and McGraw-Hill, 2008.

[3] TopCoder Community Forum: http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=greedyAlg accessed on December 19, 2013, at 7.37 a.m.

[4] University of Valladolid (uVa) Online Judge: http://uva.onlinejudge.org, accessed on December 17, 2013, at 8.40 a.m.

STATEMENT

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 20 Desember 2013

[illegible]

Salvian Reynaldi
13511007