The Hiring Problem

Lilah Kelly

June 9, 2021

Abstract

This report addresses The Hiring Problem. The Hiring Problem is a relatively simple concept based on continuously hiring the best person for a job. In order to fully understand it, two simple examples are given. Because of The Hiring Problem's simplicity, it can be implemented with just a few lines of pseudo code. However, the theory behind the concept is a bit more in depth than one may think. In order to cover all aspects of this problem, a detailed description of The Hiring Problem will be given, along with an extensive proof. Additionally, the cost considerations of this algorithm will be touched upon and discussed. Finally, reflections on the algorithm will be made.

1 Introduction

The Hiring Problem has a fairly unconventional computational problem. The input is a list n of potential candidates, and the output is then several permanent decisions of which candidates to hire, and a final result of the best out of all n hired and categorized as the *best*.

The algorithm is thus responsible for picking the best possible candidate for some open job position. Expanding on the unconventional outputs, the algorithm hires a candidate if they are the best so far. All remaining candidates are required to be compared to this hired candidate. If they are better suited for the job, they will replace the previously hired candidate, and so on. There are also specified costs for both interviewing and hiring each candidate. The issue is then how to properly estimate the total cost of hiring the best possible person for the job out of all possible candidates.

1.1 Context

In terms of the rest of CSC 250, The Hiring Problem can be best related to several different concepts. The most obvious correlation to make is with sorting algorithms, as The Hiring Problem is essentially a sorting algorithm. More specifically, it is most similar in concept to the algorithm developed in Problem Set 5 based on Problem 4-5 in the textbook. The relation is the fact that the two algorithms share the purpose of sorting a set number of items into 'good' and 'bad' categories. For Problem Set 5, the concern was deciding which chips were functional (the good) or nonfunctional (the bad), and for The Hiring Problem the concern was finding the person most suited for the job (the good) out of the rest of the interviewed candidates (the bad).

It can also be argued that The Hiring Problem is slightly similar to Quicksort, as in they both have a sort of partition. For The Hiring Problem, the partition is set by the best candidate so far, with the other previous candidates which are less suited for the job placed behind said partition.

For Quicksort, the partition is the divide between the items less than on the right of the pivot, and the items greater than the pivot on the left. Thus, the partition for The Hiring Problem can be imagined as having all previous candidates to the left, and nothing to the right.

2 Examples

Below we will explore two examples of The Hiring Problem being implemented. The first will analyze the worst-case cost, and the second will analyze an average-case cost (See Total Cost Section). For simplicity, both examples will share the prices for interviewing and hiring. For interviewing, the cost will be $c_i = 50$ and the cost for hiring will be $c_h = 100$.

Let's assume we are given 6 candidates. Their qualification for the job, which in this case is intelligence level, is indicated in square brackets from 1 to 6, with 6 being for the most qualified. For the following examples, our list of candidates will be

$$n = [Janet[6], Tahani[3], Chidi[4], Michael[5], Eleanor[2], Jason[1]]$$

2.1 Example 1

Given our n, we now feed it into Random. As indicated by the name of the function, each time n is fed into it, a new random permutation will be returned. Our list of candidates is now

$$n = [Jason[1], Eleanor[2], Tahani[3], Chidi[4], Michael[5], Janet[6]]$$

. As the ranks indicate, the order happened to get randomized so that the candidates will be presented in increasing order of competency for the job.

We now continue to assign *best* as 0. Our first candidate is $n[1]$, or $Jason[1]$. We thus add his interview cost to the total cost, as well as his hiring cost, as he is better than the current *best* candidate. Our total cost is now $100 + 50 = 150$. Continuing on, we have $n[2]$, or $Eleanor[2]$. $Elenaor[2]$ is interviewed, and is smarter than $Jason[1]$, so we must add her interview and hiring costs to the total cost. We now have $150 + 100 + 50 = 300$. Moving on, we now have $Tahani[3]$, who is once again smarter than our current *best* candidate, $Eleanor[2]$, so our cost increases by $300 + 100 + 50 = 450$. Next is $Chidi[4]$, who is smarter than $Tahani[3]$, so the cost becomes $450 + 100 + 50 = 600$. $Michael[5]$ is then next, who is interviewed and then compared to $Chidi[4]$. He is smarter than $Chidi[4]$, so his interview and hiring costs are added to the total cost, giving us a total cost of $600 + 100 + 50 = 750$. Finally, we have $Janet[6]$, who is an all-knowing being, meaning they are then the best possible candidate out of all previous candidates. Therefore, their interview and hiring costs have to be added to the total cost, which gives us the overall total cost of $750 + 100 + 50 = 900$. In the next example we will explore how this is not the most cost effective version of hiring someone for our job opening.

2.2 Example 2

Given our n, we now feed it into Random. As indicated by the name of the function, each time n is fed into it, a new random permutation will be returned. Our list of candidates is now

$$n = [Jason[1], Chidi[4], Tahani[3], Michael[5], Janet[6], Eleanor[2]]$$

.

We now continue to assign *best* as 0. Our first candidate is $n[1]$, or $Jason[1]$. We thus add his interview cost to the total cost, as well as his hiring cost, as he is better than the current *best* candidate. Our total cost is now $100+50 = 150$. Continuing on, we have $n[2]$, or *Chidi*[4]. *Chidi*[4] is interviewed and is smarter than $Jason$, so we must add his interview and hiring costs to the total cost. We now have $150 + 100 + 50 = 300$. Moving on, we now have $Tahani[3]$, who is once again interviewed. However, she is not smarter than *Chidi*[4], so we only add her interview cost to the total cost, giving us $300 + 50 = 350$. Next is $Michael[5]$, who is smarter than *Chidi*[3], so we add both his interview and hiring costs to the total, which is now $350 + 100 + 50 = 500$. $Janet[6]$ is next. They are an all-knowing being, meaning they are smarter than $Michael[5]$, so their hiring and interview costs are added. The total cost is now $500 + 100 + 50 = 650$. Last is *Eleanor*[2]. *Eleanor*[2] is interviewed and compared against $Janet[6]$, meaning she does not get hired and we only have to pay the interviewing cost. Thus, we end the process with the total cost of $650 + 50 = 700$. Comparing this to the worst-case running time example above, we have saved the hypothetical company we work for 200 dollars (See Examples section 2.1).

3 The Algorithm

```
Hire-Assistant($n$)
1  RANDOM($n$) //randomizes order of $n$
2  $best = 0$
3  for $i = 1$ to $n$
4      interview candidate $i$
5      if candidate $i$ is better than candidate $best$
6          $best = i$
7          hire candidate $i$
```

[1, p.115 Section 5.1]

3.1 Description

The pseudo code HIRE-ASSISTANT first takes an input of n, and immediately randomizes its order. Next, *best* is assigned a value of 0, which the first candidate is then compared to in order to always make the first candidate the *best* so far. Once *best* has been initialized, the algorithm iterates through all n candidates and interviews each one. If the candidate i being interviewed has a higher rank than the current *best* candidate, then $best = i$. Additionally, the previous *best* candidate is fired and is replaced by hiring candidate i. At the end of the for loop, *best* should carry the highest ranked candidate from input n.

HIRE-ASSISTANT is known as a randomized algorithm. What makes it a randomized algorithm is line 1, which randomly permutes the order of n before the for loop. The purpose of randomizing n is to ensure that there is no real order of the candidates being presented. In other words, it is essential that n is not sorted in any way. The reasoning behind this is that the algorithm wants to avoid the worst-case running time (See Section Running Time4.1). The original version of this algorithm did not have line 1, and was therefore at risk of having the worst-case running time [1, Section 5.1]. Thus, there must be some control over the order of candidates being presented, which is why calling RANDOM in the first line was a necessary modification.

3.2 Proof

First, we will prove that randomizing n works by using probabilistic analysis. Probabilistic analysis is when probability is used to analyze a problem if and only if there is some control or knowledge of the input."the use of probability in the analysis of problems...we must use knowledge of, or make assumptions about, the distribution of the inputs" ([1, Section 5.1]. For The Hiring Problem, we will assume that n is permuted in some random order, and that all permutations are equally likely. We will use m as an indicator random variable, which is essentially a boolean variable, which will be 1 if the candidate i is hired, and 0 when the candidate i is not hired. Thus, we get that

$$E[m_i] = Pr[\text{candidate } i \text{ is hired}]$$

[1, Lemma 5.1 Section 5.2]

X_i is our boolean variable for this equation. It will be 1 when candidates 1 to $i-1$ have been interviewed and are worse than candidate i, and 0 if any of the candidates 1 to $i-1$ are better. Since we started this proof with the assumption that all permutations are equally likely, we know that it is also equally likely for all candidates i to be the best candidate. Candidate i then has the probability of being hired 1 out of i times. Thus, by using Lemma 5.1, each candidate i has the probability of being

$$E[m_i] = 1/i$$

[1, Lemma 5.1 Section 5.2]

Thus, the summation of $E[m_i]$, which is $E[m]$, can then be calculated as follows:

$$E[m] = E[\sum_{i=1}^{n} m_i] = \sum_{i=1}^{n} E[m_i] = \sum_{i=1}^{n} 1/i = \ln n + O(1)$$

From left to right, we know that $E[m]$ is equal to the sum of all $E[X_i]$ where $i = 1$ to n. We can then shift E inside the summation via linearity of expectation. Next, we recognize that this can get translated into the summation of $1/i$ where $i = 1$ to n. Finally, we use a summation property, which says that

$$\sum_{K=1}^{n} 1/k = ln \cdot n + O(1)$$

[[1, A.7 Section A.1]

The final result gives us Lemma 5.2, which is used for our average cost (See Section Running Time 4.2) [1, Lemma 5.1 Section 5.2].

As for the rest of the algorithm, we will prove by induction.

At the beginning of the for loop from lines $3-7$, all candidates $[1..i-1]$ have been interviewed and compared against the *best* candidate out of $1..i-1$, with the *best* candidate out of $[1..i-1]$ hired.

Initialization: For this case, $i = 1$. Plugging this into the loop invariant, we get that all candidates [1..0], or [0] have been interviewed. Since [0] is an empty list, it is trivially true. Additionally, the *best* candidate is initialized to 0, or an empty candidate, this is also trivially true.

Maintenance: For this case, $1 \leq i < n$. We know that for each iteration of the for loop, i iterates by 1. We also know that in each iteration i, i is interviewed and then compared to the *best* candidate. Thus, for $i = 2$, we know that the for loop iterated through $i = 1$, and compared it to the *best* candidate, and so on for each increasing value of i until $i = n$. Thus, for all cases where $1 \leq i < n$ at the beginning of iteration i, all candidates $[1..i-1]$ have been interviewed and compared against the *best* candidate. We also know that for each iteration i that is better than the *best* candidate becomes the new *best* candidate, and replaces the previous by getting hired. Thus, at the beginning of iteration i, the best of all candidates $[1..i-1]$ has been hired.

Termination: For this case, $i = n+1$. Plugging this in, we get that all candidates $[1..n+1-1]$ $= [1..n]$ have been interviewed and compared against the *best* candidate. This means that all candidates in set n have been interviewed and compared against the *best* candidate, meaning that the for loop terminates with the *best* candidate out of all $[1..n+1-1]$ or $[1..n]$ candidates being hired.

4 Total Cost

Let's start by assuming that both interviewing and hiring have individual costs. Interviewing will have a cost of c_i dollars, and hiring will have a cost of c_h dollars. We can then say that each running time of The Hiring Problem will cost $\Theta(n * c_i + m * c_h)$. n is the number of candidates interviewed, and m is the number of candidates hired. As proved above, m is also the summation of each true value of i from 1 to n. We know from the for loop in our algorithm that all candidates are interviewed, so once we know the number of candidates presented to us, we get our constant cost for interviewing candidates (See Section The Algorithm). In other words, in all cases of running times, the interviewing cost remains the same. This means that we only need to focus on the value of m for our cases.

4.1 Worst-case Cost

For The Hiring Problem, we know that the worst total cost would be if every candidate presented to us were in increasing order of suitability for the job opening. This would mean that all possible candidates that were interviewed would also get hired, meaning that in this situation, $n = m$. Thus, the worst case running time for HIRE-ASSISTANT would be $\Theta(n * (c_i + c_h))$.

4.2 Average-case Cost

As addressed before, we know that line one of our algorithm addresses the worst-case running time and minimizes the likelihood of the candidates being presented in increasing order of suitability for the job opening. Thus, we know that majority of the cases for running the algorithm HIRE-ASSISTANT will not be running at the same time as the worst-case running time. As proved above, we know that the average running time when n is randomized is actually $\Theta(c_h \ln n)$ plus the interview cost $\Theta(c_i \cdot n)$, which simplifies to $\Theta(c_h \ln n + c_i \cdot n)$ [1, Lemma 5.2, Section 5.2].

5 Reflection

After researching The Hiring Problem thoroughly, I gained knowledge in how probabilistic analysis is helpful for optimizing algorithms. However, the authors of the three sources I consulted seemed to agree that randomizing n would minimize the likelihood of the worst-case running time occurring. However, as seen above, it is still entirely possible to encounter a worst-case running time when randomizing n (See Examples section 2.1). Thus, it is not a direct disagreement, but I think there must be some better way of modifying this algorithm in order to avoid the worst-case running time 100 percent of the time. As the problem of finding a more efficient approach than randomization is currently unsolved, randomization does seem to be reasonable. After researching real-world examples of this topic, I concluded that there are very few if any real-world applications. However, when prompted to research its predecessor, The Secretary Problem, I was able to find an example of an online advertising system [**?**]. This system essentially sells banner ads to companies, with new options for places to put banner ads updating all the time. Additionally, there is a buyback where you have to pay to give up a previously purchased banner ad if a better opportunity came. This concept of consumerism, paired with the Simpsons example from one of my sources, inspired me to create examples with pop culture characters [2].

I feel the concept of probabilistic analysis could have been explored more. What I mean by this is that from the one example I looked at, I believe I understand probabilistic analysis to a reasonable degree. However, I am unsure in my capabilities of using it myself. Thus, I know enough to follow someone else's work, but I don't think I know enough to properly apply it.

In conclusion, there are two aspects of The Hiring Problem that I would further research if given the time. The short-term research would be exploring probabilistic analysis further in order to feel more comfortable conveying it to my peers. The long-term research would be looking into real world applications of The Hiring Problem, as my short-term research has yielded no results. Perhaps consulting with the professor of this course would be beneficial in figuring out a way to research this, as well as consulting academic databases such as JSTOR.

In terms of the sources I consulted, I appreciated the CLRS source the most. This is because it was clear that it was used as a source when the other two sources were created. Additionally, the methods in which new concepts were communicated were clear. I did have to read ahead in certain instances to get the full picture (i.e. proving The Hiring Problem with probabilistic analysis) [1, Section 5.2]. Also, as I've commented before via Perusall, I personally prefer to learn with some sort of analogy. CLRS tends to not use analogies for concepts, which is understandable, as not all students need parallels drawn in order to grasp new ideas. However, from a personal perspective, it did make it a bit difficult to fully understand the material. The second source provided a clear organization of how probabilistic analysis is applied to The Hiring Problem, as well as necessary background information that helped me understand it better. The third source was my personal favorite. This is because it discussed The Hiring Problem in detail, as well as providing clear visuals to illustrate how it worked. Additionally, it used Simpsons characters to represent different candidates, which I appreciated. Not only did it help me understand the material better, but it inspired me to create my own example for The Hiring Problem using my favorite television show, The Good Place.

References

[1] Cormen, Thomas H. and Leiserson, Charles E. and Rivest, Ronald L. and Stein, Clifford *Introduction to Algorithms, Third Edition*, The MIT Press, 3rd edition, 2009.

[2] Vassilvitskii, Segei and Broder, Andrei and Kirsch, Adam and Kumar, Ravi and Mitzenmacher, Michael, and Upfal, Eli *The Hiring Problem: Going Beyond Secretaries*

`https://www.slideshare.net/tinoubao/the-hiring-problem`

[3] Pollett, Chris, *The Hiring Problem, Probability Review, Indicator Random Variables* 2006. http://www.cs.sjsu.edu/faculty/pollett/255.1.06s/Lec30012006.pdf

[4] Babaioff, Moshe and Hartline, Jason D. and Kleinberg, Robert D. 2012. *Selling Banner Ads: Online Algorithms with Buyback* https://www.researchgate.net/publication/228707002_Selling_banner_ads_Online_algorithms_with_buyback/link/0c96052cdc4447a680000000/download