@S S Roy

Randomized Algorithms for the Hiring Problem

1. Problem Statement

The **Hiring Problem** models the process of hiring the best candidate for a job. The problem assumes that candidates arrive sequentially, and we must decide immediately whether to hire or reject a candidate, without recalling past candidates.

2. Naïve Deterministic Approach

In a deterministic setting, the best strategy is to hire the first best candidate encountered. However, without knowledge of future candidates, we might hire a suboptimal choice early on. A deterministic strategy does not work well if better candidates arrive later.

3. Randomized Algorithm for the Hiring Problem

A **randomized approach** improves the probability of selecting the best candidate. The idea is to divide the selection process into two phases:

- **Phase 1 (Learning Phase):** Observe the first `k` candidates but do not hire anyone.
- **Phase 2 (Selection Phase):** Hire the first candidate who is better than all the candidates observed in Phase 1.

This method ensures a better chance of selecting the best candidate.

---

4. The Randomized Algorithm

Algorithm Steps

1. **Choose a random integer** `k` (typically `k = n/e` where `e ≈ 2.718`).
2. **Observe** the first `k` candidates but do not hire.
3. **Select the first candidate** that is better than all observed candidates.
4. **If no such candidate is found**, stick with the last candidate.

---

5. Numerical Example

Problem Setup

Suppose there are **7 candidates**, each with a unique ranking (1 = best, 7 = worst):

Candidate	Rank (1 = best)
A	3
B	6
C	1
D	5
E	2
F	7
G	4

We assume candidates arrive in the order `A → B → C → D → E → F → G`.

Step-by-Step Execution

Step 1: Choose `k`

For `n = 7`, an optimal choice is `k = ⌊7/e⌋ = ⌊7/2.718⌋ ≈ 2`.
So, we observe the first **2** candidates (`A` and `B`) but do not hire anyone.

Step 2: Observe First `k` Candidates (Learning Phase)

- Candidate **A** arrives (Rank **3**). **Do not hire**.
- Candidate **B** arrives (Rank **6**). **Do not hire**.

At the end of this phase, the **best candidate seen so far is A (Rank 3)**.

Step 3: Selection Phase

- Candidate **C** arrives (Rank **1**). Since **1 < 3 (best seen so far)**, **hire C**.
- Stop hiring, even if better candidates arrive later.

Thus, **Candidate C is selected**, who is actually the best candidate!

---

6. Analysis of Probability of Success

Success Probability Calculation

To succeed, we must select the absolute best candidate. The probability of this happening follows the **Secretary Problem**:

$$P(\text{Success}) \approx \frac{1}{e} \approx 0.37$$

That means the randomized approach selects the best candidate **about 37% of the time**, which is significantly better than a naive deterministic method.

---

7. Advantages of Randomized Hiring

- **Higher success rate** (37%) compared to naive strategies.
- **Works without prior knowledge** of candidate distribution.
- **Efficient** since decisions are made in **O(n)** time.

8. Disadvantages

- Still **not guaranteed** to pick the best candidate.
- Requires **randomness**, which might not be practical in some real-world hiring processes.

---

9. Conclusion

Randomized algorithms provide an elegant solution to the hiring problem, significantly increasing the probability of selecting the best candidate compared to naive approaches. By using a **learning phase followed by an optimal stopping rule**, we can improve hiring outcomes.

Example-2

1. Problem Statement

The **Hiring Problem** models a scenario where we want to hire the best candidate for a job. However, we must make hiring decisions **immediately** without the ability to recall previously rejected candidates.

We assume:

- There are **n** candidates.
- Each candidate has a unique rank (1 = best, n = worst).
- Candidates arrive in a **random order**.
- Once a candidate is rejected, they cannot be recalled.

---

2. The Randomized Algorithm

The **randomized selection strategy** follows these steps:

1. **Choose a threshold** `k`, usually `k = n/e` (where `e ≈ 2.718`).
2. **Observe** the first `k` candidates without hiring.
3. **Hire the first candidate** who is better than all observed candidates.
4. **If no such candidate appears, hire the last one.**

Why Does This Work?

- The first `k` candidates **serve as a benchmark** (learning phase).
- The remaining candidates **have a chance to be compared against an established standard**.
- This approach increases the probability of hiring the best candidate.

---

3. Step-by-Step Numerical Example

Let's consider **7 candidates** arriving in random order. Their **true rankings** are:

Candidate	Rank (1 = best)
A	4
B	2
C	6
D	1
E	5
F	3
G	7

Candidates arrive in this **random order**:
`C → A → B → G → F → E → D`

Step 1: Compute `k`

For `n = 7`, the optimal threshold is:

$$k = \lfloor \frac{7}{e} \rfloor = \lfloor \frac{7}{2.718} \rfloor = \lfloor 2.57 \rfloor = 2$$

This means we **observe** the first **2** candidates without hiring.

Step 2: Learning Phase (`k=2` Observations)

- Candidate **C** arrives (Rank **6**) → Observe but do not hire.
- Candidate **A** arrives (Rank **4**) → Observe but do not hire.

At the end of this phase, the **best candidate seen so far is A (Rank 4)**.

Step 3: Selection Phase (Hire the First Better Candidate)

Now, we evaluate the remaining candidates and **hire the first candidate better than Rank 4**.

- Candidate **B** arrives (Rank **2**)
  - Rank **2** < Rank **4** (Best seen so far) ✅ **Hire B**.
  - Stop the process (no further evaluation).

Final Decision: Hired Candidate = B (Rank 2)

---

4. Another Numerical Example

Let's analyze **another random arrival order** for the same set of candidates.

New Arrival Order:

`E → B → A → G → C → D → F`

Step 1: Compute `k`

(Same as before, `k = 2`)

Step 2: Learning Phase (`k=2` Observations)

- Candidate **E** arrives (Rank **5**) → Observe.
- Candidate **B** arrives (Rank **2**) → Observe.

At the end of this phase, the **best seen candidate = B (Rank 2)**.

Step 3: Selection Phase

- Candidate **A** arrives (Rank **4**)
  - Rank **4** > Rank **2** → **Do not hire**.
- Candidate **G** arrives (Rank **7**)
  - Rank **7** > Rank **2** → **Do not hire**.
- Candidate **C** arrives (Rank **6**)
  - Rank **6** > Rank **2** → **Do not hire**.
- Candidate **D** arrives (Rank **1**)
  - Rank **1** < Rank **2** ✅ **Hire D**.
  - Stop the process.

Final Decision: Hired Candidate = D (Rank 1, Best Overall)

---

5. Probability Analysis

What is the chance of selecting the best candidate?

The probability of success follows the **Secretary Problem** formula:

$$P(\text{Best Candidate Selected}) = \frac{1}{e} \approx 0.37$$

This means that the algorithm **selects the best candidate 37% of the time**, which is significantly better than **random selection (14%)**.

---

6. Why Does This Algorithm Work Well?

- **We learn from early candidates** (reducing risk).
- **We avoid hiring too early** (ensuring a better selection process).
- **We increase the probability of picking the best candidate**.

---

7. Worst-Case and Best-Case Scenarios

Worst Case

- The best candidate appears in the **first `k` candidates** (we never hire them).
- Example: If the best candidate (Rank 1) appears in the learning phase, they will be ignored.

Best Case

- The best candidate appears just **after the learning phase**, making them immediately hireable.

---

8. Summary & Key Takeaways

Factor	Naive Hiring	Randomized Algorithm
Probability of Best Candidate	Low (~14%)	~37%
Decision Time Complexity	**O(n)**	**O(n)**
Recall Allowed?	❌ No	❌ No
Strategy	Pick first best candidate	Observe **`k`**, then pick first better

Key Insights

- The randomized strategy **triples the probability** of hiring the best candidate compared to naive approaches.
- The best **value of `k` is `n/e`**, balancing learning and selection phases.
- Works well **in real-world hiring**, auctions, and decision-making scenarios.

---

9. Additional Variations

1. **Multiple Positions Hiring:** Extend the strategy for hiring `m` out of `n` candidates.
2. **Cost-Based Hiring:** Introduce costs for interviewing candidates to optimize selection.
3. **Relaxed Recall:** Some real-world settings allow recalling the last `m` candidates.

---

10. Conclusion

Randomized algorithms provide a **powerful solution** for hiring problems by **balancing observation and selection**. By carefully choosing **k = n/e**, we can maximize our chances of hiring the **best candidate** in an **efficient, probabilistic** manner.

@S S Roy, 23rd March, 2025