Sección de Matemáticas
Universidad de La Laguna

Laura Macía Vázquez

El problema de la secretaria

The secretary problem

Trabajo Fin de Grado
Grado en Matemáticas
La Laguna, Julio de 2020

Dirigido por
Carlos M. González Alcón

Carlos M. González Alcón
Departamento de Matemáticas,
Estadística e Investigación
Operativa
Universidad de La Laguna
38200 La Laguna, Tenerife

Agradecimientos

A mis padres.

Laura Macía Vázquez
La Laguna, 7 de julio de 2020

Resumen · Abstract

Resumen

En este trabajo se presenta un problema de parada óptima, conocido por una de sus versiones como el problema de la secretaria. Consiste en seleccionar al mejor candidato para un puesto vacante entre un número conocido de solicitantes que acuden a ser entrevistados. Se estudia la solución aportada por la literatura y se muestran los resultados mediante simulaciones y gráficas. Se aporta código informático con funciones que implementan la solución, tanto haciendo uso de la recursividad como evitándola. También se estudia un problema similar propuesto por Cayley, en el que el número de entrevistas está acotado. En la última parte del trabajo se proponen y resuelven dos generalizaciones del problema inicial. En la primera se permite cualquier función objetivo. En la segunda, se resuelve el problema cuando las vacantes a cubrir son varias.

Palabras clave: *Parada óptima – Problema de la secretaria – Recursividad – Lenguaje de programación R – Simulación.*

Abstract

This work presents an optimal stopping problem, known by one of its versions as the secretary problem. It involves selecting the best candidate for a vacant position from a known number of applicants who come to be interviewed. The solution provided by the literature is studied and the results are shown using simulations and graphs. Computer code is provided with functions that implement the solution, both making use of recursion and avoiding it. A similar problem proposed by Cayley is also studied, in which the number of interviews is limited. In the last part of the work, two generalizations of the initial problem are proposed and solved. In the first, any objective function is allowed. In the second, the problem is solved when there are several vacancies to fill.

Keywords: *Optimal stopping – Secretary problem – Recursion – Programming language R – Simulation.*

Contenido

Introducción

Si escribimos en un buscador de internet *optimal stopping*, el primer enlace que nos aparece será la definición de problema de parada óptima de la conocida Wikipedia. El ejemplo que proponen es el problema de la secretaria que también se conoce como problema del matrimonio, juego de googol o problema de la dote del sultán.

El objetivo del problema es contratar al mejor de los candidatos presentados a un puesto vacante. Para ello se les va entrevistando de forma secuencial. Tras cada entrevista se contrata al candidato o se le descarta definitivamente. La meta es el diseño de estrategias óptimas que nos digan cuándo parar el proceso de selección.

En el primer capítulo se presenta el problema, sus antecedentes históricos y versiones que existen y trabajaremos en su solución recursiva. Esta solución primero será desarrollada para conocer bien cómo funciona la regla de detención óptima y a continuación se verificará realizando diversas simulaciones. Seguidamente planteamos una función recursiva en lenguaje R y se comprueba que los resultados son los mismos. El problema de la lentitud para valores grandes que muestra la recursividad nos lleva a dar una función que la evite basándonos en crear una matriz con los valores que necesitamos del anterior programa.

En el segundo capítulo se aborda el problema de Cayley, cuyo planteamiento es similar al de la secretaria. Se recoge la solución propia del autor junto con otras dos propuestas. De nuevo programamos una función que implementa la solución. Hablando en términos del problema de la secretaria, en el de Cayley el número de entrevistas no puede superar determinada cantidad y el pago es el valor del candidato que finalmente resulta contratado.

El último capítulo se destina a generalizar el problema de la secretaria. Nuestra primera idea ha sido considerar funciones objetivo generales, que de-

pendan del valor del candidato seleccionado y/o del número de entrevistas. De nuevo se programan funciones que implementan las soluciones y con los resultados que se obtienen podremos ver la política de decisión a seguir. La otra modificación consiste en tener que cubrir varios puestos vacantes en vez de uno, lo que será posible solucionar mediante simulación, y la política de decisión es definida previamente.

1

El problema de la secretaria

Vamos a presentar la versión tradicional del problema de la secretaria, a contar sus antecedentes históricos y a trabajar en su solución.

El planteamiento clásico de este problema que encontramos en [3] es el siguiente:

Se ha quedado un puesto libre de secretaria en una oficina y el jefe quiere contratar a alguien para cubrir ese puesto y publica un anuncio para hacer entrevistas y contratar al mejor de los candidatos.

Los postulantes pueden ordenarse estrictamente de manera que hay uno que es el mejor de todos, pero este orden es desconocido para la persona que realiza las entrevistas. Las entrevistas se hacen en orden aleatorio. Se entrevista sucesivamente a los candidatos.

Tras cada entrevista ha de decidirse si contratar a ese o rechazarlo; si se rechaza ya no podrá ser contratado. Si al llegar a la última entrevista todavía no ha escogido, se quedará con este último candidato. El objetivo es maximizar la probabilidad de contratar al mejor de los candidatos.

Hay dos principales formas de plantear este problema, la forma clásica que es una forma discreta donde hay un número finito de candidatos, y una forma continua donde hay un número infinito numerable o número desconocido de postulantes.

En este trabajo nos centraremos únicamente en esta primera forma en la que conocemos cuántos son los candidatos posibles.

1.1. Antecedentes históricos

El problema de la secretaria apareció en febrero de 1960 en la columna de Martin Gardner en la revista *Scientific American* donde fue atribuido a J.H. Fox y L.G. Marnie.

En marzo de ese mismo año y en la misma revista, se publicó una primera solución del problema por L. Moser y J.R. Pounder.

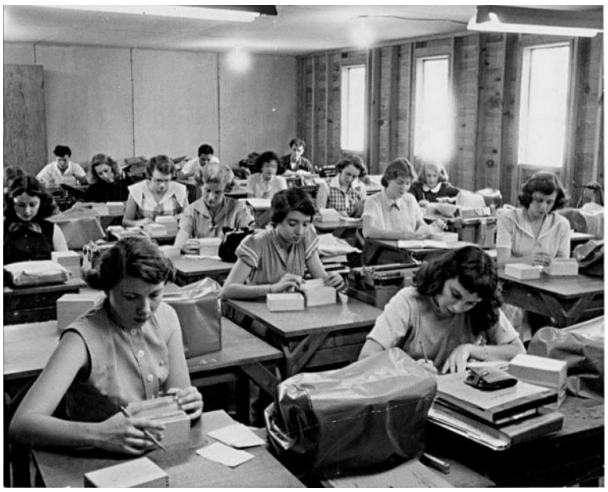


Figura 1.1. Candidatos al puesto vacante.

Tres años después, en 1963, en *The American Mathematical Monthly* apareció el problema atribuido a Bissinger y Siegel, quienes lo propusieron con un caso particular de $n = 1000$ candidatos, y la solución fue propuesta por Bosh en 1964.

Otro matemático, Mosteller, dijo que había conocido este problema en 1955 por Andre Gleason quien a su vez dijo haberlo escuchado de otros.

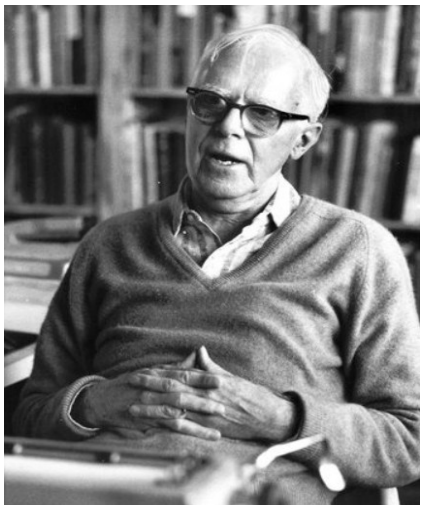
En resumen, cuando aparece el problema publicado por primera vez ya muchos habían escuchado hablar de él. Pero fue Lindley en 1961 quien resolvió el problema en una revista científica. Su solución se basa en el rango del candidato seleccionado y considera el problema de minimizar ese rango, siendo 1 el mejor.

Pero esta manera tenía dificultades cuando n era grande. Dynkin propuso en 1963 otra solución basándose en la teoría de los tiempos de detención de Markov. No fue hasta 1966, cuando Gilbert y Mosteller formularon una solución que presagió la cantidad de generalizaciones que impactarían en este área desde 1972 y que continúa a día de hoy.

Para descubrir quién ha sido la primera persona que ha planteado el problema de la secretaria, hay que remitirse a documentos históricos.

Johannes Kepler (1571–1630), famoso astrónomo alemán, perdió a su esposa en 1611 debido al cólera, y comenzó a buscar una nueva compañera de vida utilizando las mismas estrategias metódicas y con el mismo detalle y cuidado con el que encontró la órbita de Marte.

En una carta que envía en 1613 a Strahlendorf tras haber seleccionado ya a una mujer, explica con detalle los problemas que durante el proceso de selección ha tenido, y el porqué de cada decisión que había tomado. Otro matemático,



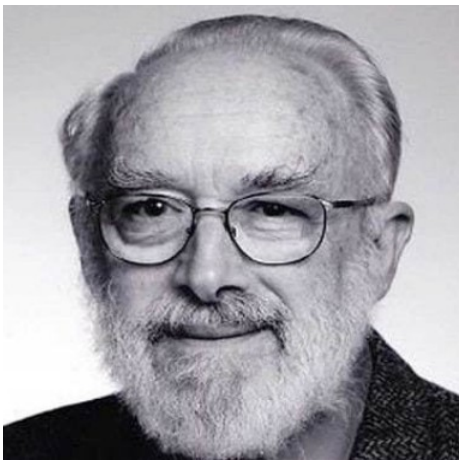


Figura 1.2. Martin Gardner (1914–2010) y Dennis Victor Lindley (1923–2013).

Arthur Cayley (1821–1895) propuso un problema similar al de la secretaria pero con tres bolas de lotería, del que más adelante hablaremos.

Por último y como ya he citado anteriormente, en 1960 en *Scientific American* apareció el problema como vemos la propia revista [2], con el nombre juego de 'googol'.

Este juego que aparece explicado consistía en escoger de un montón de papelitos en los que en cada uno había escrito un número que iba de pequeñas fracciones de 1 hasta un googol (10^{100}) o incluso mayores. Se barajan dados la vuelta y después se van levantando y se para el juego cuando se crea que se ha levantado el de mayor valor. Así que este juego también tiene el mismo planteamiento que el de la secretaria.

1.1.1. Distintas versiones del problema de la secretaria

Enumeremos los elementos clave del problema clásico de la secretaria como aparecen en [3], de los cuales modificando alguno de ellos se obtendrán distintas versiones:

1. Hay un puesto vacante de secretaria.
2. El número n de candidatos al puesto es conocido.
3. Los candidatos son entrevistados secuencialmente en orden aleatorio y cada orden es igual de probable.
4. Se puede clasificar a los candidatos del mejor al peor. La decisión de escoger o rechazar depende del rango de cada uno.
5. Una vez se rechaza a un candidato, no puede volver a ser llamado.

6. Sólo se conforma si se escoge al mejor de todos, aquel cuyo rango sea 1.

Sin embargo a lo largo del tiempo han aparecido distintas maneras de plantear el problema o de resolverlo, vamos a citar los principales:

1. El problema de la secretaria con información completa (*Full-information problem*).
   Con las mismas condiciones que el problema tradicional, se conoce la distribución de probabilidad que siguen estos enteros aleatorios así como los parámetros de la distribución.
2. El problema de la secretaria con información parcial (*Partial-information problem*).
   El mismo planteamiento que el anterior y con conocimiento de la distribución de probabilidad pero no se conocen todos sus parámetros.
3. El problema de la secretaria con memoria finita (*Finite memory problem*).
   En las mismas hipótesis pero ahora el jefe no puede acordarse de todos los candidatos que ha entrevistado, sino de un número finito de ellos.
   Después de cada entrevista tiene que decidir si contratar al candidato, rechazarlo, o rechazarlo guardándolo en la memoria.
4. El problema de la secretaria de múltiple elección (*Multiple-choice problem*).
   Aquí el jefe escoge a un conjunto de candidatos.

También se puede plantear el problema cambiando la función objetivo. Hasta el momento, las versiones del problema que hemos citado, tienen la misma función objetivo, maximizar la probabilidad de encontrar al mejor de los candidatos.

Pero podemos plantearlo teniendo en cuenta otros detalles:

1. Minimizar el rango del candidato: costes de entrevista.
   Se supone que a cada candidato se le asigna un valor (positivo) y que cada entrevista tiene un costo (negativo) fijo.
   El objetivo sería minimizar el costo total de la serie de entrevistas, es decir, el valor del candidato quitando los costes de las entrevistas realizadas.
2. Optimización multiobjetivo.
   El problema de la secretaria también se podría plantear de manera que existan varias funciones objetivo. Por ejemplo si tenemos en cuenta la bondad del candidato y los costes de la entrevista. La función objetivo sería minimizar esos costes y también el orden del candidato en paralelo.

1.2. Solución recursiva del problema de la secretaria

El estado en el que nos encontramos en el proceso de la resolución del problema de la secretaria puede ser descrito por un par de números (r, s) como

encontramos en [3], donde r es el número de candidatos presentados hasta el momento y s es el rango aparente del último candidato presentado, el r-ésimo.

Para entender cómo funciona este par (r, s) que también se explica en [6], pongamos un ejemplo en el cual se presentan a la entrevista para el puesto vacante $n = 5$ candidatos.

A cada candidato se le asigna su bondad, es decir, un número entero positivo que significa lo bueno que es para el puesto, siendo 1 el mejor de todos.

A la entrevista, como ya he mencionado anteriormente, entran en un orden aleatorio. Imaginemos el siguiente orden de entrada y las siguientes bondades: $7, 2, 5, 8, 3$.

$(3, 2)$: aquí estaríamos expresando que hemos entrevistado a tres candidatos, y este último es el segundo mejor de ellos.
$(4, 4)$: ahora si nos quedamos con el cuarto candidato entrevistado, observamos que su rango aparente es el peor hasta el momento.
$(5, 2)$: escogiendo al último postulante, éste es el segundo mejor de todos. Por lo tanto no nos estaríamos quedando con el mejor.

Con esto, podemos entender que si $s \neq 1$, no estamos escogiendo al mejor de los candidatos. En cambio, si $s = 1$, es un candidato a aceptar y la probabilidad de que sea el mejor es $\frac{r}{n}$, que se obtiene de la probabilidad condicionada $P(A|B)$:

A = el r–ésimo candidato es el mejor de todos.

B = el r–ésimo candidato es el mejor de los r primeros.

$P(A \mid B) = \frac{P(A)}{P(B)} = \frac{\frac{1}{n}}{\frac{1}{r}} = \frac{r}{n}$.

Dado que estamos en (r, s), si entrevistamos a un nuevo candidato, pasaremos a un estado $(r + 1, s')$ donde s' puede tomar cualquier valor entre 1 y $r + 1$.

Definimos entonces la función $M(r, s)$ que denota el máximo de la probabilidad esperada de encontrar al mejor candidato.

Veamos cómo se construye.

Explicaremos esta función de recurrencia empezando por el caso $s = 1$ para un r arbitrario: $M(r, 1)$.

En este caso de la figura 1.3, el candidato r entrevistado tiene de rango $s = 1$, y habría dos posibilidades (las dos ramas de la imagen): lo contratamos y su probabilidad de ser el mejor es $\frac{r}{n}$; o entrevistamos a los siguientes postulantes.

En esta segunda rama estaríamos en el estado $M(r+1, s')$. Con cada valor de s', la probabilidad de que sean el mejor sería la misma, $\frac{1}{r+1}$.

De manera que si decidimos entrevistar a un nuevo candidato, el valor esperado será:

$$\frac{1}{r+1}[M(r+1,1) + \cdots + M(r+1, r+1)] = \frac{1}{r+1}\sum_{s'=1}^{r+1} M(r+1, s').$$

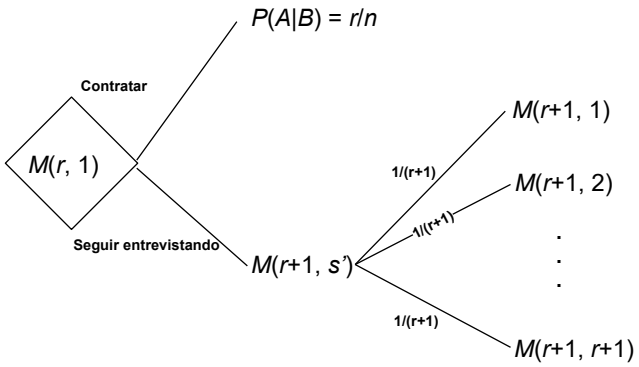


Figura 1.3. Función de recurrencia $M(r, 1)$.

Por lo tanto, nos decidiremos por la opción que tenga mayor valor esperado entre contratar al candidato o seguir entrevistando, esto es:

$$M(r,1) = \text{máx}\left\{\frac{r}{n}, \frac{1}{r+1}\sum_{s'=1}^{r+1} M(r+1,s')\right\} \tag{1.1}$$

donde

$$M(r,s) = \frac{1}{r+1}\sum_{s'=1}^{r+1} M(r+1,s'). \tag{1.2}$$

Cuando el candidato entrevistado es el último, estamos en el estado $M(n, s)$. Si $s = 1$, la probabilidad de que ese sea el mejor es 1; si $s \neq 1$, sería 0.

Pongamos ahora el ejemplo de antes, $n = 5$ candidatos.

Si estamos en el estado $M(4, 3)$ de la figura 1.4, no contrataríamos a ese candidato ya que su rango es 3 y por lo tanto la probabilidad de ser el mejor es 0. Seguiríamos realizando entrevistas, es decir, pasaríamos al estado $M(5, s')$ donde nos quedaríamos con el último candidato.

Hallando la esperanza de todos estos valores, concluiríamos que $M(5, s') = \frac{1}{5}$ y por lo tanto $M(4, 3) = \frac{1}{5}$.

Esto ocurriría siempre para $M(4, s \neq 1)$, la probabilidad nos daría $\frac{1}{5}$.

Pero, ¿cuánto valdría $M(4, 1)$?

Si contratamos al cuarto candidato, la probabilidad de que sea el mejor sería $\frac{4}{5}$. Si sigo con las entrevistas, pasaríamos al estado $M(5, s')$ donde s' toma valores de 2 a $r + 1$ y todos con probabilidad $\frac{1}{5}$.

Llegados a este punto en la figura 1.5, tengo que elegir si quedarme con el primer candidato o seguir.

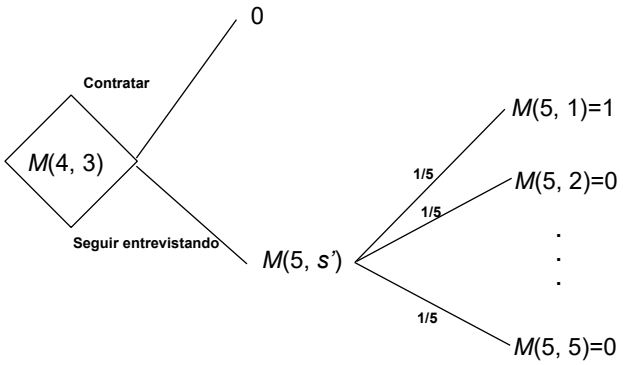


Figura 1.4. Función de recurrencia $M(4,3)$.

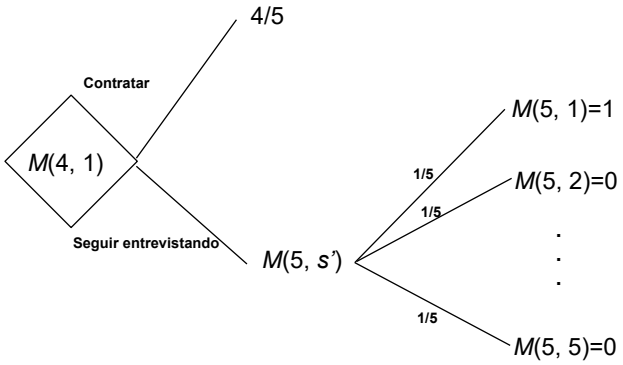


Figura 1.5. Función de recurrencia $M(4,1)$.

Lo que sería $M(4,1) = \text{máx}\{\frac{4}{5}, \frac{1}{5}\}$ por tanto me quedo con el último candidato entrevistado.

1.3. Programando el problema de la secretaria.

En esta sección comenzaremos a programar la versión clásica del problema de la secretaria.

Haremos distintas simulaciones con n número de candidatos en orden aleatorio, y veremos cual es el candidato r elegido con mayor probabilidad de ser el mejor.

```
s <-  #número de simulaciones
num.candidatos <-

r.optimo.prob.elegir.al.mejor <- numeric(length(num.candidatos))

for (n in num.candidatos){
    #almacenanamos los resultados de las simulaciones
    #sumando el resultado de cada simulación
    n.entrevistas.total <- numeric(n-1)
    elegido.el.mejor.total <- numeric(n-1)
    b.elegido.total <- numeric(n-1)
    for (i in 1:s){
        candidatos <- sample(1:n)
        el.mejor <- min(candidatos)
        rmejor <- cummin(candidatos)
        donde.el.mejor <- which(candidatos == el.mejor)
        #para los valores de r que están más allá del mejor el resultado
        #es entrevistar a todos los candidatos y quedarse con el último
        for (r in 1:min(donde.el.mejor,n-1)){
            n.entrevistas <- r + which(candidatos[-(1:r)] < rmejor[r])[1]
            elegido.el.mejor <- FALSE
            if (is.na(n.entrevistas)) n.entrevistas <- n
            b.elegido <- candidatos[n.entrevistas]
            if (b.elegido == el.mejor) elegido.el.mejor <- TRUE
            print(c(r, n.entrevistas, b.elegido, elegido.el.mejor))
            n.entrevistas.total[r] <-
                n.entrevistas.total[r] + n.entrevistas
            elegido.el.mejor.total[r] <-
                elegido.el.mejor.total[r] + elegido.el.mejor
            b.elegido.total[r] <-
                b.elegido.total[r] + b.elegido
        }
        n.entrevistas.total[-(1:donde.el.mejor)] <-
                        n.entrevistas.total[-(1:donde.el.mejor)] + n
        elegido.el.mejor.total[-(1:donde.el.mejor)] <-
                        elegido.el.mejor.total[-(1:donde.el.mejor)] + 0
        b.elegido.total[-(1:donde.el.mejor)] <-
                b.elegido.total[-(1:donde.el.mejor)] + candidatos[n]
    }
    r.optimo.prob.elegir.al.mejor[which(num.candidatos==n)] <-
                                which.max(elegido.el.mejor.total/s)
}
```

1. Para $n = 15$ candidatos
   Realizando simulaciones diferentes, obtenemos que el mejor candidato para ser contratado es el que corresponde al quinto entrevistado $r = 5$, con una probabilidad que en todas las simulaciones se aproxima a 0.39. La figura 1.6 lo muestra.
2. Para $n = 40$ y para $n = 100$ candidatos
   Realizando las mismas simulaciones que en el caso anterior, obtenemos que el mejor candidato para ser contratado es el que corresponde al $r = 15$; como

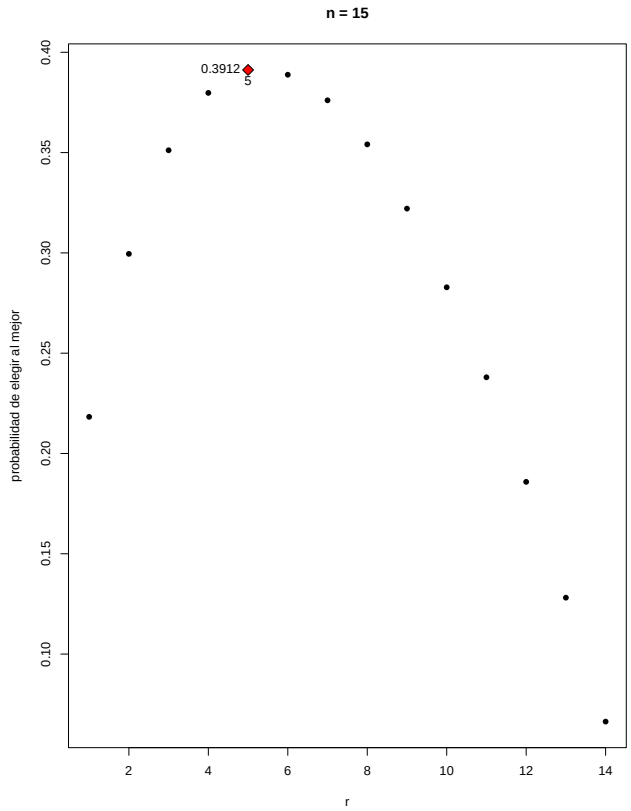


Figura 1.6. Mejor candidato r de los $n = 15$.

se aprecia en la figura 1.7. Aquí la probabilidad de que el elegido sea el mejor de todos es 0.375.
En el caso de 100 candidatos, es necesario hacer muchas simulaciones para que no haya ruido en la gráfica; obteniendo que el mejor candidato para ser contratado es el que corresponde al $r = 37$, con una probabilidad de 0.371506. Véase la figura 1.7

Ahora, veremos el número de entrevistas que hay que realizar para escoger al mejor candidato r: En este caso y sea cual sea el n número de candidatos, a mayor r, mayor número de entrevistas que hay que realizar como se puede ver en la figuras 1.8.

Lo único interesante en este ejemplo, es apreciar los óptimos de pareto, que son aquellos puntos correspondientes entre el primer candidato y el r óptimo estudiado anteriormente para cada caso, donde se realiza un menor número de entrevistas.

Veamos ahora como podemos, programando la función recursiva $maxM$, calcular la probabilidad de elegir al mejor de los n candidatos presentados cuando se han entrevistado a r de ellos y el último presenta rango aparente s:

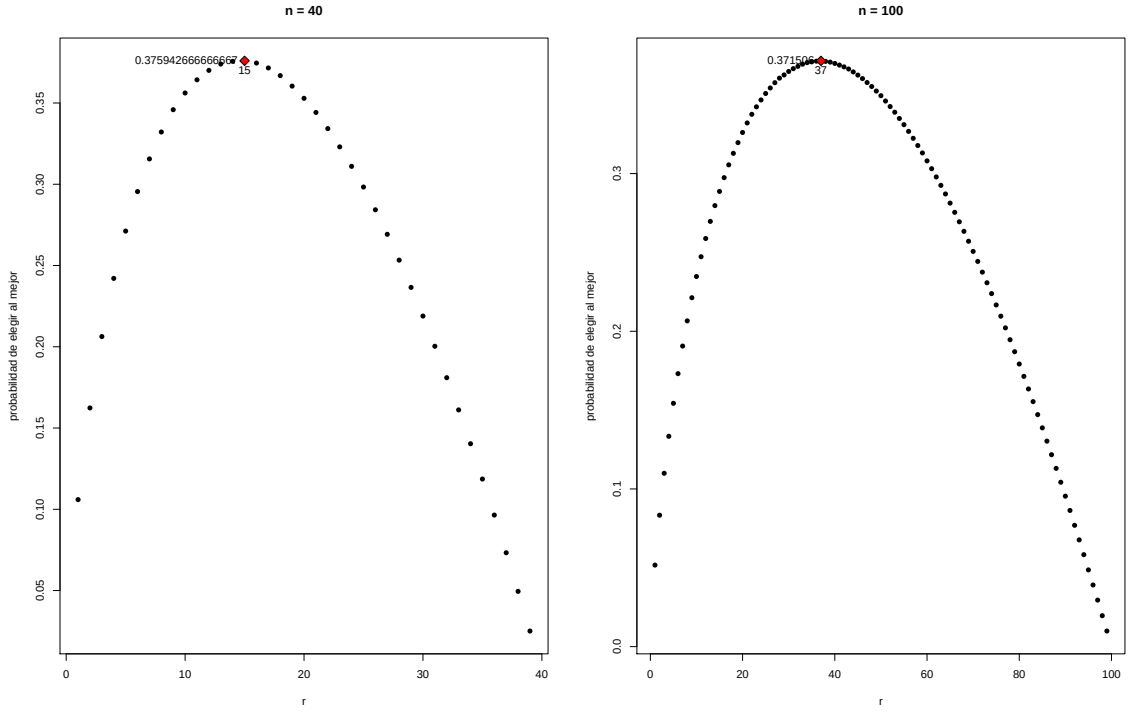


Figura 1.7. Mejor candidato r de los $n = 40$ y $n = 100$.

```
Función recursiva
Probabilidad de elegir al mejor
maxM <- function(r, s, n){
  if (r < s) return(0)
  if (r == n)
    if (s == 1)  return(1)  else  return(0)
  else{
    esperanza <- 0
    esperanza <- maxM(r+1, 1, n) + r*maxM(r+1, 2, n)
    if (s == 1)
      return (max(r/n, esperanza/(r+1)))
    else
      return (esperanza/(r+1))
  }
}
```

Esta función recursiva $maxM$ es muy lenta y gastará muchos recursos, ya que si por ejemplo queremos calcular $maxM(1, 1, 24)$, tardará casi 18 segundos de ejecución.

```
#tiempo de jercución de la función recursiva maxM con n=24
t <- proc.time()
maxM(1,1,24)
proc.time() - t
   user  system elapsed
```

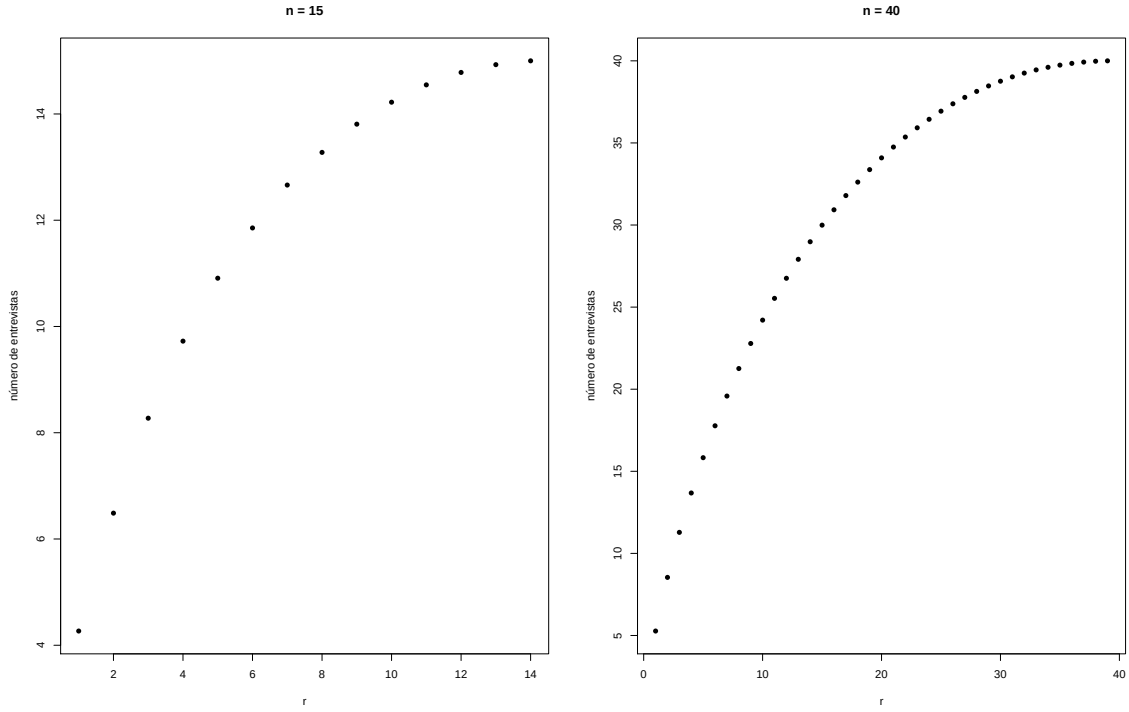


Figura 1.8. Número de entrevistas $n = 15$ y $n = 40$.

```
  17.36    0.00   17.49
```

Para $n = 25$ se duplica el tiempo que tarda en ejecutar el anterior n. Por lo tanto podemos concluir que no compensa hacer estos cálculos de manera recursiva.

```
#tiempo de jercución de la función recursiva maxM con n=25
t <- proc.time()
maxM(1,1,25)
proc.time() - t
   user  system elapsed
  36.58    0.17   36.98
```

Podemos acumular todos los valores que obtenemos con la función $maxM$ en una matriz $3 \times n$ que llamaremos M sin necesidad de usar dicha función recursiva.

A las filas de M las llamaremos $s = 1, 2$ y a las columnas $r = 1, \ldots, n$. Comenzamos haciendo $M[1, n] = 0$ y $M[2, n] = 1$.

A continuación construimos la matriz de tal manera que la primera fila sea $\frac{1}{n}$, luego empezando por la última columna la segunda fila será $max\{M[1, r], M[3, r]\}$. Y por último la segunda fila se calcule como $(M[2, r+1]+r*M[3, r+1])/(r+1)$.

Solo necesitamos saber esas 3 primeras filas ya que lo único interesante es el $M[r, s = 1]$ y el $M[r, s = 2]$ y encontraremos el $r + 1$ cuando la diferencia de

ellos, es decir, $M[r, s = 1] - M[r, s = 2]$ sea 0. Por lo tanto el r óptimo será el anterior.

Todo este proceso se puede realizar en una función sencilla y muy rápida:

```
#Matriz M con tres filas y n columnas, una para cada valor de r
secretaria.prob.mejor <- function(n){
  M <- rbind((1:n)/n, (1:n)/n, (1:n)/n)
  M[3,n] <- 0
  for (r in (n-1):1){
    M[3,r] <- (M[2,r+1] + r*M[3,r+1])/(r+1)
    M[2,r] <- max(M[1,r], M[3,r])
  }
  list(r.optimo = which(M[2,] - M[1,] == 0)[1]-1, prob = M[2,1])
}
```

Esta función no recursiva es mucho más rápida y eficaz que la anterior, ya que por ejemplo, con $n = 1500$ candidatos, calcula el r óptimo y su probabilidad en un tiempo de 6 milésimas de segundo.

```
t <- proc.time()
secretaria.prob.mejor(1500)
proc.time() - t
user  system elapsed
   0.02    0.02    0.06
```

Comparando los 18 segundos que tardaba nuestra función recursiva $maxM$ solamente con $n = 24$ frente a las 6 milésimas de la función no recursiva con $n = 1500$, claramente vemos que es mucho más eficiente el uso de la no recursividad para la resolución de este problema.

2

El problema de Cayley

Arthur Cayley (1821–1895) fue un distinguido matemático inglés, muy conocido por su trabajo en la teoría de invariantes algebraicos. Sus trabajos están recogidos en unos 966 documentos. El documento 705 contiene cincuenta páginas con problemas y sus soluciones que Cayley envió a la revista *Educational Times* desde 1871 hasta 1894.

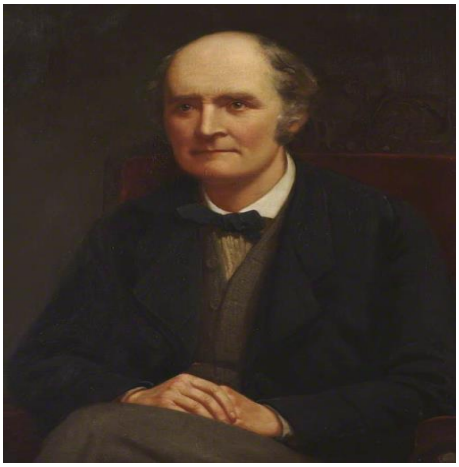


Figura 2.1. Arthur Cayley (1821–1895).

En 1895, publicó en esa revista un problema muy similar al problema de la secretaria que dice lo siguiente:

Una lotería funciona de la siguiente manera. Hay n tiques que representan $a, b, c, \ldots$ libras respectivamente. El jugador levanta un tique y mira su valor, y si quiere, levanta otro de los $n-1$ restantes; no superando levantar más de k veces. El jugador se queda con el valor del último tique que levante.

4528. (Proposed by Professor CAYLEY.)—A lottery is arranged as follows:—There are n tickets representing a, b, c pounds respectively. A person draws once; looks at his ticket; and if he pleases, draws again (out of the remaining $n-1$ tickets); looks at his ticket, and if he pleases draws again (out of the remaining $n-2$ tickets); and so on, drawing in all not more than k times; and he receives the value of the last drawn ticket. Supposing that he regulates his drawings in the manner most advantageous to him according to the theory of probabilities, what is the value of his expectation?

Figura 2.2. Problema de Cayley.

¿Cuál es el valor de la esperanza de levantar el tique de mayor valor ?

2.1. Solución del problema de Cayley con inducción hacia atrás

Cayley creyó que debe entenderse que los valores de k, n y $a, b, c, \ldots$ son conocidos. Él resolvió el problema con lo que hoy en día conocemos como inducción hacia atrás de la programación dinámica como podemos ver en [2]. Cayley propuso un ejemplo con $n = 4$ y $a, b, c, d = 1, 2, 3, 4$, y encuentra para $k = 1, 2, 3, 4$ la esperanza correspondiente.

Como aparece en la figura 2.3, es sencillo observar que al levantar un único tique, la esperanza es:

$E = 4\frac{1}{4} + 3\frac{1}{4} + 2\frac{1}{4} + 1\frac{1}{4} = \frac{10}{4}$

En el caso de poder levantar hasta un máximo de dos tiques, véase en la figura 2.4, la esperanza es:

$E = 4\frac{1}{4} + \text{máx}\{3, \frac{7}{3}\}\frac{1}{4} + \text{máx}\{2, 3\}\frac{1}{4} + \text{máx}\{1, \frac{8}{3}\}\frac{1}{4} = \frac{38}{12}$

Si ahora el jugador levanta hasta tres veces los tiques como se muestra en la figura 2.5, el estudio de la esperanza es:

$E = 4\frac{1}{4} + (4\frac{1}{4}\frac{1}{3} + \frac{1}{4}\frac{1}{3}\frac{5}{2} + 3\frac{1}{4}\frac{1}{3}) + (4\frac{1}{4}\frac{1}{3} + 3\frac{1}{4}\frac{1}{3} + \frac{1}{4}\frac{1}{3}\frac{7}{2}) + (4\frac{1}{4}\frac{1}{3} + 3\frac{1}{4}\frac{1}{3} + \frac{1}{4}\frac{1}{3}\frac{7}{2})$

Para el último caso donde el jugador llega hasta la cuarta extracción, es obvio que la esperanza es: $E = 4$

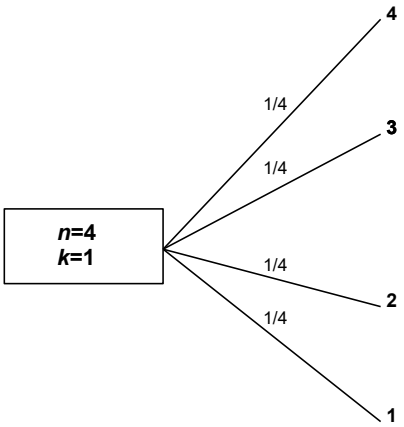


Figura 2.3. Caso $n = 4$ tiques, $k = 1$ extracción.

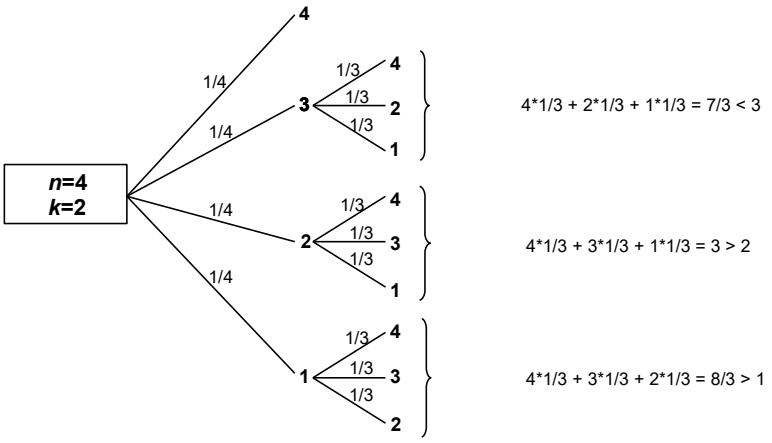


Figura 2.4. Caso $n = 4$ tiques, $k = 2$ extracciones.

Existen puntos en común entre ambos problemas pero hay una importante diferencia ya que en el problema de la secretaria original la recompensa es 0 si no contratamos al mejor de los candidatos y 1 si contratamos al mejor, y en el problema de Cayley la recompensa es el valor del tique con el que nos quedemos.

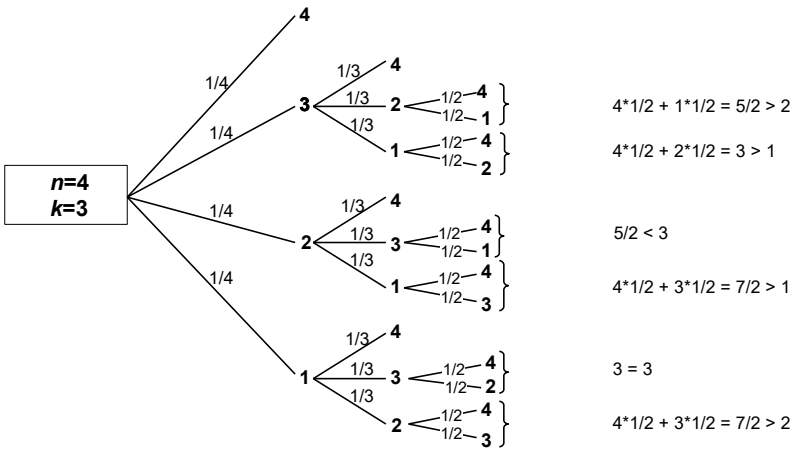


Figura 2.5. Caso $n = 4$ tiques, $k = 3$ extracciones.

2.2. Soluciones del problema de Cayley

2.2.1. Solución por Cayley

En el libro Mathematical Questions with their solutions from the Educational Times VOL. XXIII. aparece el problema de Cayley original y algunas soluciones.

La primera es la propuesta por el propio Cayley y la podemos consultar en [1].

Sean las cantidades $(a, b, c, \dots)$ y sea $M_1(a, b, c, \dots)$ la media de esas cantidades, es decir, su suma dividida entre el número de ellos.

Para fijar mejor las ideas, consideremos 5 cantidades a, b, c, d, e y escribimos:

$$M_1(a, b, c, d, e) = M_1(a, b, c, d, e)$$
$$M_2(a, b, c, d, e) = M_1(\text{máx}\{a, M_1(b, c, d, e)\}, \text{máx}\{b, M_1(a, c, d, e)\}, \dots, \text{máx}\{e, M_1(a, c, d)\})$$
$$M_3(a, b, c, d, e) = M_1(\text{máx}\{a, M_2(b, c, d, e)\}, \text{máx}\{b, M_2(a, c, d, e)\}, \dots, \text{máx}\{e, M_2(a, c, d)\})$$

y así sucesivamente.

El proceso es similar en el caso de cualquier número de cantidades $a, b, c, \dots$. Entonces el valor de la esperanza es $M_k(a, b, c, \dots)$.

Si $k = 1$, el valor es $M_1(a, b, c, \dots)$.

Si $k = 2$, si escoge a, el jugador estará satisfecho o escogerá de nuevo, esto es, la esperanza será $\text{máx}\{a, M_1(b, c, \dots)\}$.

Si el jugador escoge b, la esperanza será máx$\{b, M_1(a, c, \dots)\}$. Y así con los demás valores.

Por tanto la esperanza es:

$$M_2(a, b, c, \dots) = M_1(\text{máx}\{a, M_1(b, c, \dots)\}, \text{máx}\{b, M_1(a, c, \dots)\}, \dots)$$

y similar para k= 3, k= 4,...

El ejemplo visto anteriormente, ahora sería: $a, b, c, d = 1, 2, 3, 4$.

Entonces ,

$$\begin{aligned}
M_1(1,2,3,4) &= \tfrac{10}{4} \\
M_2(1,2,3,4) &= M_1(\text{máx}\{1, \tfrac{9}{3}\}, \text{máx}\{2, \tfrac{8}{3}\}, \text{máx}\{3, \tfrac{7}{3}\}, \text{máx}\{4, \tfrac{6}{3}\}) \\
&= M_1(\tfrac{9}{3}, \tfrac{8}{3}, 3, 4) = \tfrac{38}{12} \\
M_2(2,3,4) &= M_1(\text{máx}\{2, \tfrac{7}{2}\}, \text{máx}\{3, \tfrac{6}{2}\}, \text{máx}\{4, \tfrac{5}{2}\}) \\
&= M_1(\tfrac{7}{2}, 3, 4) = \tfrac{21}{6} \\
M_2(1,3,4) &= M_1(\text{máx}\{1, \tfrac{7}{2}\}, \text{máx}\{3, \tfrac{5}{2}\}, \text{máx}\{4, \tfrac{4}{2}\}) \\
&= M_1(\tfrac{7}{2}, 3, 4) = \tfrac{21}{6} \\
M_2(1,2,4) &= M_1(\text{máx}\{1, \tfrac{6}{2}\}, \text{máx}\{2, \tfrac{5}{2}\}, \text{máx}\{4, \tfrac{3}{2}\}) \\
&= M_1(3, \tfrac{5}{2}, 4) = \tfrac{15}{8} \\
M_3(1,2,3,4) &= M_1(\text{máx}\{1, \tfrac{21}{6}\}, \text{máx}\{2, \tfrac{21}{6}\}, \text{máx}\{3, \tfrac{19}{6}\}, \text{máx}\{4, \tfrac{15}{6}\}) \\
&= M_1(\tfrac{21}{6}, \tfrac{21}{6}, \tfrac{19}{6}, 4) = \tfrac{85}{24} \\
M_3(1,2,3) &= \dots \\
M_4(1,2,3,4) &= M_1(\text{máx}\{1,4\}, \text{máx}\{2,4\}, \text{máx}\{3,4\}, \text{máx}\{4,3)\} \\
&= M_1(4,4,4,4) = 4
\end{aligned}$$

Finalmente,

$$M_1, M_2, M_3, M_4 = \frac{10}{4}, \frac{38}{12}, \frac{85}{24}, 4 = \frac{60}{24}, \frac{76}{24}, \frac{85}{24}, \frac{694}{24}$$

2.2.2. Solución por Moser

Moser en el año 1956 [9] plantea el siguiente problema. Observamos secuencialmente una serie de variables aleatorias $X_1, \dots, X_k$ independientes e idénticamente distribuidas, según una distribución uniforme $(0, 1)$. Si nos detenemos tras observar X_j, recibimos ese valor X_j como premio.

Moser propuso que lo óptimo sería detenerse cuando quedan m observaciones si el valor de la presente observación es mayor que E_m, donde los E_m se definen recursivamente por $E_0 = 0$ y $E_{m+1} = \frac{1+E_m^2}{2}$.

La explicación de esta solución la vemos de la siguiente manera:

Tenemos el intervalo $[0, 1]$, si sólo podemos realizar una observación, la media será $\frac{1}{2}$. Si podemos realizar hasta dos observaciones, si el valor de la primera observación está ente 1 y $\frac{1}{2}$ nos plantamos y si está entre 0 y $\frac{1}{2}$ volvemos a observar.

Entonces el pago medio que vamos a obtener es la probabilidad de obtener un número superior a la media que es $\frac{1}{2}$, por la esperanza del pago cuando es mayor que la media, más la probabilidad de observar un número entre 0 y $\frac{1}{2}$ y multiplicado por la esperanza de otra observación $\frac{1}{2}$.

Entonces,

$$E_2 = \frac{1}{2}\frac{3}{4} + \left(1 - \frac{1}{2}\right)\frac{1}{2}.$$

Generalizando podemos ver que

$$E_{m+1} = (1 - E_m)\frac{1 - E_m}{2} + E_m^2 = \frac{1 + E_m^2}{2}.$$

Este problema de Moser sirve como una reformulación del problema de Cayley cuando se considera un n grande y los valores de los tiques $a, b, c, \ldots$ son $1, 2, 3, \ldots, n$.

2.2.3. Solución por S. Tebay

En [1] se ofrece también una solución propuesta por S. Tebay.

La notación es bastante ambigua y según reconoce el mismo autor no es exacta y en ocasiones estima la esperanza por abajo.

Lo que podemos sacar en claro de esta propuesta es: si hacemos la media de los valores de los tiques que hay en el juego, cuando levantamos un tique, si su valor está por debajo de esa media debemos levantar otro tique. Si por el contrario el valor del tique es superior a esa media, debemos quedarnos con él.

2.3. Programando el problema de Cayley

Podemos programar la solución al problema de Cayley mediante una función $maxE$ recursiva que tiene como argumentos k que es el número de extracciones que realiza el jugador, y *tique* que es el vector de los valores de cada tique. Esta función recursiva, como la programada en el problema de la secretaria, tiene un mal tiempo de ejecución cuantos más tiques hay en el juego y también cuantas más extracciones hagamos.

```
#problema de Cayley
#maxE tiene como argumentos
#k: cantidad de extracciones posibles
#tique: vector con el valor de cada tique
maxE <- function(k, tique){
```

```
    if (k==1) mean(tique)
    else{
        total <- 0
        for (i in 1:length(tique)){
            total <- total + max(tique[i], maxE(k-1, tique[-i]) )
        }
        total/length(tique)
    }
}
```

Podemos calcular el ejemplo visto antes, con $n = 4$ tiques y comprobar que obtenemos los mismos resultados para cada extracción.

```
> maxE(1,1:4)
[1] 2.5
> maxE(2,1:4)
[1] 3.167
> maxE(3,1:4)
[1] 3.541667
> maxE(4,1:4)
[1] 4
```

Veamos ahora con $n = 10$ como quedan las probabilidades:

```
> maxE(1, 1:10)
[1] 5.5
> maxE(2, 1:10)
[1] 6.889
> maxE(3, 1:10)
[1] 7.681
> maxE(4, 1:10)
[1] 8.223
> maxE(5, 1:10)
[1] 8.613
> maxE(6, 1:10)
[1] 8.936
> maxE(7, 1:10)
[1] 9.199
> maxE(8, 1:10)
[1] 9.393
> maxE(9, 1:10)
[1] 9.615
> maxE(10, 1:10)
[1] 10
```

3

Generalizaciones del problema de la secretaria

En este capítulo trataremos algunas posibles generalizaciones del problema de la secretaria y las programaremos para evaluar los resultados. Para ello, primero retomaremos las seis características del problema.

1. Hay un puesto vacante de secretaria.
2. El número n de candidatos al puesto es conocido.
3. Los candidatos son entrevistados secuencialmente en orden aleatorio y cada orden es igual de probable.
4. Se puede clasificar a los candidatos del mejor al peor. La decisión de escoger o rechazar depende del rango de cada uno.
5. Una vez se rechaza a un candidato, no puede volver a ser llamado.
6. Sólo se conforma si se escoge al mejor de todos, aquel cuyo rango sea 1.

Nuestro interés es rebajar alguna de estas características y comenzaremos relajando la función objetivo, expresada en el último punto, esta idea la hemos sacado de [7].

3.1. Función objetivo general

En el problema clásico, tal como se indica en el punto 6, nuestro objetivo es contratar al mejor de los candidatos y por ello maximizamos la probabilidad de contratarlo.

Ahora, el rango del candidato finalmente contratado es indiferente si no es el 1.

Crearemos una función pago como vemos en [8] que dependerá de tres variables: w será el rango absoluto del candidato contratado, e el número de candidatos entrevistados, y n el número total de candidatos.

```
#c: bondad del ultimo candidato entrevistado
#e: numero de entrevistados
#n: total de candidatos
pago <- function(e, w, n){
```

```
}
```

Podemos implementar el pago en una función M que sirve para poner una función objetivo genérica pero manteniendo todas las características citadas excepto la número 6.

La función M nos devolverá la esperanza del pago en caso de que actuemos de forma óptima.

Evaluada en el punto (r, s, n) indica que de los n posibles candidatos, llevamos r entrevistados y el últimos de ellos tiene un rango relativo s.

Para ello debemos comparar el pago medio que obtendremos si entrevistamos a un nuevo candidato con el pago esperado si contratamos al último candidato entrevistado.

```
#funcion general (sin devolver accion)
#r: numero de entrevistados hasta el momento
#s: rango relativo del ultimo candidato entrevistado
#n: total de candidatos
M <- function(r, s, n){
  if (r < s) return(NA)
  if (r == n)
    return( pago(n,s,n) )
  else{  #s < r
    esperanza.si.sigo <- 0
    for (i in 1:(r+1))
        esperanza.si.sigo <- esperanza.si.sigo + M(r+1,i,n)/(r+1)

    esperanza.si.planto <- 0
    for (k in 0:(n-r))
      esperanza.si.planto <- esperanza.si.planto +
      pago(r, s+k, n)*choose(n-s-k, r-s)*choose(s+k-1, s-1)
    esperanza.si.planto <- esperanza.si.planto/choose(n,r)

    return(max(esperanza.si.planto, esperanza.si.sigo ))
  }
}
```

Vamos a imitar el problema original a efectos de ver cómo funciona en los ejemplos el programa.

```
pago <- function(e, w, n){
   if (w == 1) return(1) else return(0)
}
}
```

Por ejemplo, si tenemos $n = 4$ candidatos para ser entrevistados y hemos entrevistado a 3 de ellos y este último tiene de rango relativo 2; suponiendo que a partir de este candidato el jefe o entrevistador se comporta de la mejor manera, el pago esperado será de 0.25.

```
> M(3,2,4)
[1] 0.25
```

Podemos comprobar que obtenemos el mismo resultado si usamos la función recursiva $maxM$ que vimos en el capítulo 1.

```
> maxM(3,2,4)
[1] 0.25
```

Podemos modificar la función M de tal manera que además de devolver la esperanza, nos devuelva la acción a tomar con cada candidato entrevistado.

Ahora nos devuelve una lista con la esperanza y esa acción, siendo 0 quedarnos con ese último candidato entrevistado y 1 seguir realizando entrevistas.

```
#funcion general que indica accion a tomar
#devuelve una lista (esperanza, accion)
#donde 0: parar, 1: seguir entrevistando
#s: rango relativo del ultimo candidato entrevistado
#r: numero de entrevistados hasta el momento
#n: total de candidatos
M <- function(r, s, n){
  if (r < s) return(NA)
  if (r == n)
    return( pago(n,s,n) )
  else{  #s < r
    esperanza.si.sigo <- 0
    for (i in 1:(r+1)) esperanza.si.sigo <-
                                  esperanza.si.sigo + M(r+1,i,n)[[1]]
    esperanza.si.sigo <- esperanza.si.sigo/(r+1)

    esperanza.si.planto <- 0
    for (k in 0:(n-r))
      esperanza.si.planto <- esperanza.si.planto +
      pago(r, s+k, n)*choose(n-s-k, r-s)*choose(s+k-1, s-1)
    esperanza.si.planto <- esperanza.si.planto/choose(n,r)

    if (esperanza.si.planto < esperanza.si.sigo)
      return(list(pago.esperado=esperanza.si.sigo, accion=1))
    else
      return(list(pago.esperado=esperanza.si.planto, accion=0))
  }
}
```

Ejecutamos la función M en el mismo punto que antes con $n = 4$ candidatos y nos devuelve que en este caso es mejor seguir entrevistando.

```
> M(3,2,4)
$pago.esperado
[1] 0.25

$accion
[1] 1
```

El problema es que esta función M es recursiva, y por lo tanto para valores de n grandes, tarda mucho tiempo en ejecutarse.

Esto podemos verlo calculando la media de los tiempos en milésimas de segundo que tarda en ejecutarse $M(1, 1, n)$ con n candidatos.

```
k <- 10
npruebas <- 10
tiempos <- numeric(k)
tt <- numeric(npruebas)
for(n in 1:k){
  for(i in 1:npruebas){
    t <- proc.time()
    M(1,1,n)
    tt[i] <- (proc.time() - t)[[2]]
  }
  tiempos[n] <- mean(tt)
}
plot(1:k, tiempos, type= "b", xlab="numero de candidatos",
 ylab="tiempo (ms)", main="Tiempo de ejecucion de M(1, 1, n)")
```

En la gráfica 3.1 podemos apreciar el crecimiento exponencial que es prácticamente inherente en la recursividad. Realizando 10 pruebas con $n = 10$ candidatos vemos ese crecimiento a medida que vamos entrevistando.

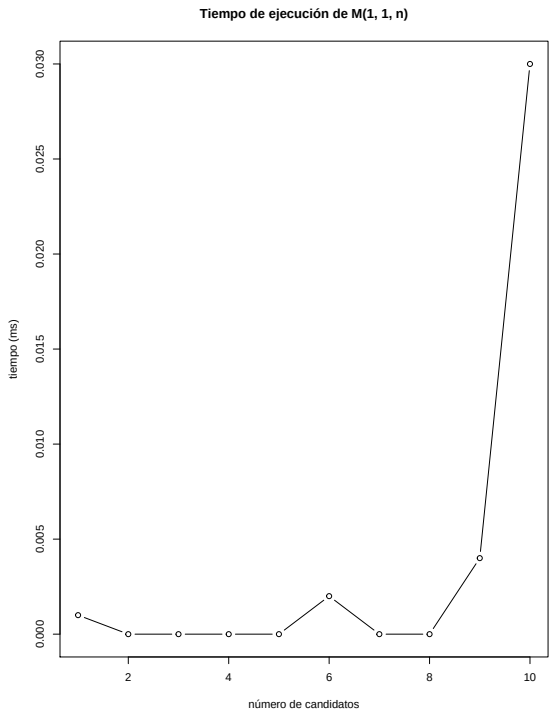


Figura 3.1. Tiempo de ejecucion de $M(1, 1, n)$

Para ejecutar $M(1, 1, 12)$ se tarda más de una hora,por lo que tenemos realmente un problema ya que los tiempos se disparan.

Lo solucionaremos creando un programa que devuelva dos matrices $n \times n$ donde r el número de candidatos entrevistados hasta el momento son las columnas, y s el rango relativo del último candidato entrevistado son las filas.

La primera matriz de pagos esperados $PE[s, r]$, devuelve el pago medio que se consigue cuando se ha entrevistado a r candidatos y ese último tiene rango relativo s.

La matriz de acción a tomar $A[s, r]$ guarda la acción a tomar, siendo 0 que debemos quedarnos con ese candidato y 1 que debemos seguir entrevistando.

```
#matriz de nxn donde el elemento PE[s,r] me da el pago medio
#que puedo conseguir si llevo r candidatos entrevistados y el ultimo
#tiene de rango relativo s
#En la matriz A guardo al accion a tomar;
#0: plantarme, 1: entrevistar a uno mas
n <-
PE <- matrix(NA, nrow=n, ncol=n)
A <- matrix(NA, nrow=n, ncol=n)
#Ã°ltima columna
for (s in 1:n){
    PE[s,n] <- pago(n,s,n)
    A[s,n] <- 0  #no hay nada que hacer
}
#las demÃ¡s columnas
for (r in (n-1):1 )
    for (s in 1:r){
        esperanza.si.sigo <- sum(PE[1:(r+1), r+1])/(r+1)
        esperanza.si.planto <- 0
        for (k in 0:(n-r))
            esperanza.si.planto <- esperanza.si.planto +
                   pago(r, s+k, n)*choose(n-s-k, r-s)*choose(s+k-1, s-1)
        esperanza.si.planto <- esperanza.si.planto/choose(n,r)

        if (esperanza.si.planto < esperanza.si.sigo){
            PE[s,r] <- esperanza.si.sigo
            A[s,r] <- 1}
        else{
            PE[s,r] <- esperanza.si.planto
            A[s,r] <- 0}
    }
```

Volvamos al ejemplo de los $n = 4$ candidatos, y veamos ambas matrices.

```
> PE
      [,1]  [,2]  [,3]  [,4]
[1,] 0.458   0.5  0.75     1
[2,]    NA 0.417  0.25    NA
[3,]    NA    NA  0.25    NA
[4,]    NA    NA    NA    NA
```

```
> A
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]   NA    1    1    0
```

```
[3,]   NA   NA    1    0
[4,]   NA   NA   NA    0
```

Podemos observar en la matriz PE que el pago esperado si nos comportamos inteligentemente cuando llevamos un único candidato entrevistado es de 0.458. Por otro lado, en la matriz A obtenemos que es mejor seguir entrevistando ya que $A[1,1] = 1$.

Si entra el segundo candidato y es el mejor hasta el momento, deberíamos quedarnos con él porque $A[1,2] = 0$; mientras que si es el segundo mejor, tenemos que seguir entrevistando.

Lo mismo pasa cuando son tres los candidatos entrevistados: si el último es el mejor, nos quedaríamos con él; mientras que si es el segundo o tercero mejor, pasaríamos a entrevistar al último candidato. Al llegar al último, nos tenemos que quedar con él.

Hemos imitado la función pago del problema inicial de la secretaria, pero nuestro objetivo era poder definir funciones generales.

Tal vez nuestro interés no sea solo contratar al mejor candidato, puede bastarnos con contratar a uno bueno.

Definamos una función de pago como lo bueno que es ese candidato, siendo $n-1$ el pago por contratar al mejor candidato, $n-2$ por contratar al segundo mejor... y un pago de 0 para el peor de los candidatos disponibles.

```
pago <- function(e, w, n){
  n - w
}
```

Con esta función, las decisiones óptimas y el estudio es un poco más complejo.

Retomando el caso de $n = 4$ candidatos al puesto de secretaria, ahora obtendríamos lo siguiente:

En la matriz PE, el pago esperado ahora va desde 0 hasta 3, siendo 3 que nos quedamos con el mejor candidato.

```
> PE
      [,1]   [,2]   [,3]   [,4]
[1,] 2.125  2.333  2.75    3
[2,] NA     1.917  1.50    2
[3,] NA     NA     1.50    1
[4,] NA     NA     NA      NA
```

La matriz de decisiones, es similar a la anterior. La diferencia está en el tercer candidato entrevistado, ya que ahora si es el mejor o el segundo mejor debemos contratarlo, y solo si es el peor de los tres entrevistados hasta el momento, entrevistamos al siguiente.

```
> A
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
```

```
[2,]   NA    1    0    0
[3,]   NA   NA    1    0
[4,]   NA   NA   NA    0
```

Por lo tanto, con una función de pago en general, el procedimiento para la elección no es el de entrevistar a un número inicial de candidatos y a partir de ahí contratar al primero que supere a los demás. Ahora el procedimiento puede ser algo más rico.

Poniendo un ejemplo con $n = 10$ vemos mejor la política óptima a seguir:

```
> round(PE,3)
       [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8] [,9] [,10]
 [1,] 7.442 7.442 7.442 7.800 8.167 8.429 8.625 8.778  8.9     9
 [2,]    NA 7.442 7.442 7.323 7.112 6.857 7.250 7.556  7.8     8
 [3,]    NA    NA 7.442 7.323 7.112 6.846 6.410 6.333  6.7     7
 [4,]    NA    NA    NA 7.323 7.112 6.846 6.410 5.722  5.6     6
 [5,]    NA    NA    NA    NA 7.112 6.846 6.410 5.722  4.5     5
 [6,]    NA    NA    NA    NA    NA 6.846 6.410 5.722  4.5     4
 [7,]    NA    NA    NA    NA    NA    NA 6.410 5.722  4.5     3
 [8,]    NA    NA    NA    NA    NA    NA    NA 5.722  4.5     2
 [9,]    NA    NA    NA    NA    NA    NA    NA    NA  4.5     1
[10,]    NA    NA    NA    NA    NA    NA    NA    NA   NA     0
> A
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    1    1    1    0    0    0    0    0    0     0
 [2,]   NA    1    1    1    1    0    0    0    0     0
 [3,]   NA   NA    1    1    1    1    1    0    0     0
 [4,]   NA   NA   NA    1    1    1    1    1    0     0
 [5,]   NA   NA   NA   NA    1    1    1    1    0     0
 [6,]   NA   NA   NA   NA   NA    1    1    1    1     0
 [7,]   NA   NA   NA   NA   NA   NA    1    1    1     0
 [8,]   NA   NA   NA   NA   NA   NA   NA    1    1     0
 [9,]   NA   NA   NA   NA   NA   NA   NA   NA    1     0
[10,]   NA   NA   NA   NA   NA   NA   NA   NA   NA     0
```

Con la función de pago tipo el problema original, veríamos siempre que para n candidatos, y tras entrevistar a r que es elóptimo, si el candidato entrevistado era el mejor lo contrataríamos, pero si no era el mejor, seguíamos entrevistando.

Con la función de pago que definimos como lo bueno que es cada candidato, vemos que si entrevistamos a un candidato y es el mejor, lo contratamos, sin embargo, desde $A[1, 6]$ vemos que si es el segundo mejor, también lo contratamos. Ya en $A[3, 8]$ observamos que si es el tercero mejor, lo contratamos, y en $A[5, 9]$ contratamos al candidato incluso si es el quinto mejor.

El r óptimo no varía sea cual sea la función de pago mientras que la decisión óptima si.

3.2. Cubrir varias vacantes

Otra generalización posible de este problema que se nos ha ocurrido leyendo [5], es modificar o relajar el punto 1. Ahora, el número de puestos vacantes será v

y tendremos que seleccionarlos de un total de n candidatos. Definir una función recursiva que asuma esta generalización no ha sido posible debido a su dificultad, y por tanto lo vamos a ver haciendo simulaciones.

En esta simulación nos da igual cuál sea la función pago, es más, podemos definir varias.

La función pago que definiremos tendrá como argumentos: n número de candidatos, v número de vacantes a cubrir, e total de entrevistas realizadas y w que es el vector de rangos absolutos de los candidatos contratados.

Queremos encontrar el r óptimo y para ello: entrevistamos a un número de candidatos r y a partir de ese momento nos quedamos con los v candidatos que son mejores que los r primeros.

El código funciona de la siguiente manera: simula un orden de los candidatos y con este orden evalúa cuál va a ser el candidato final con la función de pago. Para cada permutación de los candidatos generada, se evalúa lo bueno que es cada valor de r posible.

Es decir, para cada r número de candidatos que entrevisto sin contratar, evaluamos el máximo de ellos y sigo con las entrevistas contratando a los v que superen ese máximo. En caso de llegar al final de las entrevistas, contrataríamos a los candidatos necesarios del final.

```
#num.candidatos: total de candidatos
#num.simulaciones: numero de simulaciones
#v: candidatos a contratar

sim.varias.vacantes <- function(num.candidatos, v, num.simulaciones){
  mr <- num.candidatos - v
  n.entrevistas.total <- numeric(mr)
  pago.total <- numeric(mr)

  for (i in 1:num.simulaciones){
    candidatos <- sample(1:num.candidatos)
    for (r in 1:mr){
      m <- min(candidatos[1:r])
      w <- candidatos[candidatos < m]
      num.mejores <- length(w)
      if(num.mejores >= v){
        w <- w[1:v]
        num.entrevistas <- which(candidatos == w[v])
      } else {
        faltan <- v - num.mejores
        w <- c(w,rev(candidatos[candidatos > m])[1:faltan])
        num.entrevistas <- num.candidatos
      }
      n.entrevistas.total[r] <-
                  n.entrevistas.total[r] + num.entrevistas
      pago.total[r] <-
                 pago.total[r] + pago(num.entrevistas, w, num.candidatos)
    }
  }
  n.entrevistas.total <- n.entrevistas.total/num.simulaciones
```

```
  pago.total <- pago.total/num.simulaciones

  list(num.entrevistas = n.entrevistas.total,
        pagos.medios = pago.total, r.optimo =  which.max(pago.total))
}
```

Vamos a definir la función de pago tipo el problema original. Contrataremos a los v mejores candidatos y el pago nos devolverá 1 por cada candidato que está entre ellos. Entonces el pago nos devolverá cuántos candidatos estoy contratando de esos v mejores.

```
pago <- function(e, w, n){
  sum(w <= v)
}
```

Poniendo por ejemplo que queremos contratar a $v = 4$ candidatos y de un total de $n = 50$ que se presentan a la entrevista, obtenemos que una vez entrevistemos a $r = 8$ contrataríamos en media a 1.88 candidatos de esos v mejores y tendríamos que realizar 40 entrevistas.

```
> num.candidatos <- 50
> v <- 4
> num.simulaciones <- 10000
> sim.varias.vacantes(num.candidatos, v, num.simulaciones)
$num.entrevistas
 [1] 14.6476 21.2830 26.3120 30.5969 33.8033 36.4313 38.4812 40.3168
 41.8212 43.1132 44.2903 45.1623 45.9834 46.6335 47.2040 47.6817
 48.0690 48.4110 48.7227
 [20] 48.9648 49.1555 49.3119 49.4824 49.5738 49.6651 49.7292 49.8017
 49.8442 49.8784 49.9062 49.9320 49.9537 49.9683 49.9795 49.9850
 49.9885 49.9922 49.9968
 [39] 49.9984 49.9990 49.9996 50.0000 50.0000 50.0000 50.0000 50.0000

$pagos.medios
 [1] 0.9770 1.3398 1.5666 1.7130 1.8001 1.8569 1.8737 1.8812 1.8685
 1.8428 1.8089 1.7703 1.7232 1.6724 1.6132 1.5472 1.4880 1.4309
 1.3730 1.3150 1.2567 1.2005
 [23] 1.1424 1.0896 1.0392 0.9921 0.9419 0.8946 0.8458 0.8012 0.7570
 0.7169 0.6821 0.6430 0.6093 0.5788 0.5511 0.5200 0.4932 0.4668
 0.4413 0.4133 0.3891 0.3693
 [45] 0.3453 0.3257

$r.optimo
[1] 8
```

Ahora podemos definir una función de pago que tenga en cuenta lo buenos que son los candidatos que contratamos y a su vez que tenga en cuenta un cierto costo por cada entrevista realizada.

```
#n: total de candidatos
#w: vector de bondades de los candidatos contratados
#e: numero de entrevistados
pago <- function(e, w, n){
```

```
    sum(n - w) - 0.1*e
}
```

Entonces con el ejemplo puesto antes, obtendremos de pago medio 157.58 si entrevisto a $r = 4$ candidatos y a partir de entonces me quedo con los $v = 4$ que superen al mejor de los entrevistados, haciendo de media 30 entrevistas.

```
> num.candidatos <- 50
> v <- 4
> num.simulaciones <- 10000
> sim.varias.vacantes(num.candidatos, v, num.simulaciones)
$num.entrevistas
 [1] 14.4025 21.3255 26.4522 30.4040 33.4911 35.9371 38.0671 39.8768
 41.3910 42.7295 43.8273 44.8052 45.6916 46.4021 47.0146 47.4916
 47.9216 48.2826 48.5714
 [20] 48.8188 49.0350 49.1885 49.3618 49.4934 49.6050 49.6891 49.7499
 49.7985 49.8422 49.8823 49.9138 49.9363 49.9572 49.9709 49.9801
 49.9866 49.9923 49.9956
 [39] 49.9975 49.9991 49.9997 49.9998 49.9998 50.0000 50.0000 50.0000

$pagos.medios
 [1] 142.10955 154.10065 157.09738 157.57770 156.69209 155.04109
 152.92039 150.47472 147.77710 145.33005 142.62567 139.93008
 137.23424 134.81809 132.43174
 [16] 130.36484 128.21124 126.11814 124.14866 122.39112 120.54800
 118.97445 117.17492 115.41676 113.71260 112.07379 110.67451
 109.43345 108.16078 106.88027
 [31] 105.72622 104.55397 103.47278 102.42431 101.39719 100.39704
 99.41927  98.62264  97.88365  97.21229  96.53093  95.87892  95.26242
 94.52700  94.00330
 [46]  93.43990

$r.optimo
[1] 4
```

Otro ejemplo de función de pago podría ser el que nos devuelve si entre los candidatos que contratamos está el mejor o no.

```
#n: total de candidatos
#w: vector de bondades de los candidatos contratados
#e: numero de entrevistados
pago <- function(e, w, n){
  if (sum(w == 1)==1) return(1) else return(0)
}
```

De nuevo, con el ejemplo anterior, obtendremos que el pago medio será de 0.67 candidatos una vez entrevistemos a $r = 8$ y que tendremos que realizar en media también 44 entrevistas.

```
> num.candidatos <- 50
> v <- 4
> num.simulaciones <- 10000
> sim.varias.vacantes(num.candidatos, v, num.simulaciones)
$num.entrevistas
```

```
 [1] 14.5146 21.3003 26.4262 30.4618 33.6434 36.2716 38.4213 40.2908
 41.9370 43.2458 44.2792 45.1412 45.9892 46.6586 47.2140 47.6654
 48.0822 48.4239 48.7019
 [20] 48.9544 49.1540 49.3158 49.4662 49.5850 49.6840 49.7451
 49.7929 49.8508 49.8827 49.9065 49.9282 49.9533 49.9677 49.9731
 49.9807 49.9879 49.9923 49.9952
 [39] 49.9966 49.9981 49.9983 49.9998 50.0000 50.0000
 50.0000 50.0000

$pagos.medios
 [1] 0.2705 0.3908 0.4766 0.5383 0.5786 0.6108 0.6365 0.6528 0.6602
 0.6677 0.6746 0.6716 0.6674 0.6605 0.6519 0.6407 0.6256 0.6119
 0.5965 0.5824 0.5625 0.5440
 [23] 0.5265 0.5106 0.4912 0.4709 0.4526 0.4332 0.4127 0.3945 0.3790
 0.3622 0.3425 0.3219 0.3029 0.2818 0.2610 0.2395 0.2173 0.1974
 0.1778 0.1585 0.1380 0.1172
 [45] 0.0958 0.0790

$r.optimo
[1] 11
```

Conclusiones

En este trabajo hemos tratado un problema de parada óptima que se ha venido denominando el problema de la secretaria. Nuestro objetivo ha sido dar un procedimiento óptimo de parada para maximizar la probabilidad de contratar al candidato más idóneo. Hemos repasado las soluciones ofrecidas en la literatura científica y lo hemos implementado informáticamente.

1. Propusimos el cálculo del r óptimo así como del número total de entrevistas total que hay que realizar mediante simulaciones. Después programamos una función recursiva que nos proporcionaba la esperanza de la probabilidad de escoger al mejor candidato aunque por ser recursiva, era lenta para valores de n grandes.
2. Programamos finalmente una función no recursiva, basándonos en los valores que obteníamos con la función recursiva. Esto nos permite calcular el r óptimo y su probabilidad para valores de n grandes con muy poco tiempo de ejecución.
3. Consideramos también el problema de Cayley por su similitud con el problema de la secretaria. Ambos se diferencian en la función objetivo; en la recompensa que se obtiene, siendo 0 o 1 en el problema de la secretaria mientras que en el de Cayley es el valor del tique con el que el jugador decide quedarse, y en que hay un número k de extracciones.
4. En el último capítulo planteamos algunas generalizaciones del problema de la secretaria. Nos fijamos en los puntos clave del problema y decidimos modificar dos de ellos.

   Por un lado planteamos considerar una función objetivo general:
   - Modificando la función objetivo general, definimos funciones de pago distintas y las implementamos en una función recursiva que nos permite encontrar el pago esperado y también la acción a tomar, contratarlo o seguir entrevistando.
   - Nuevamente, la recursividad impedía un buen tiempo de ejecución para valores grandes y pudimos solucionarlo creando un programa que nos de-

vuelve dos matrices, una con los pagos esperados y la otra con la acción a tomar.
- Lo interesante ha sido ver como la política de decisión varía según la función de pago que escojamos.

Por otro lado:
- Creamos una solución del problema cuando hay varios puestos de secretaria para ser cubiertos, frente al único puesto vacante en el problema original.
- Recurrimos a las simulaciones para la programación de esta solución del problema.

Bibliografía

[1] Miller, W.J.C (Editor) *Mathematical questions with their solutions from the "Educational Times"* Vol. XXIII. C.F. Hodgson & Son, London 1875. Disponible en: https://babel.hathitrust.org/cgi/pt?id=chi.63389902.

[2] Ferguson, T.S. Who solved the Secretary Problem? *Statistical Science*,1989, Vol. 4, No. 3, pp. 282–296. Disponible en: https://www.jstor.org/stable/2245639.

[3] Freeman, P.R. The Secretary Problem and its extensions: A Review. *International Statistical Review*, 1982,Vol. 51, No. 2, pp. 189–206. Disponible en: https://www.jstor.org/stable/1402748.

[4] Gardner, Martin. Mathematical Games. *Scientific American*, February 1960, Vol. 202, No. 2, pp. 150–156. Disponible en: https://www.jstor.org/stable/24941266.

[5] Bartoszyński, R. and Govindarajulu, Z. The Secretary Problem with Interview Cost. *The Indian Journal of Statistics*, 1978, Vol. 40, No. 1/2, pp. 11–28. Disponible en: https://www.jstor.org/stable/25052096.

[6] Lindley, D.V. Dynamic Programming and Decision Theory. *Journal of the Royal Statistical Society*, 1960, Vol. 10, No. 1, pp. 39–51. Disponible en: https://www.jstor.org/stable/2985407.

[7] Lorenzen, T.J. Generalizing the Secretary Problem. *Advances in Applied Probability*, 1979, Vol. 11, No. 2, pp. 384–396. Disponible en: https://www.jstor.org/stable/1426845.

[8] Lorenzen, T.J. Optimal Stopping with Sampling Cost: The Secretary Problem. *The Annals of Probability*, 1981, Vol. 9, No. 1, pp. 167–172. Disponible en: https://www.jstor.org/stable/2243190.

[9] Moser, L. On a problem of Cayley. *Scripta Mathematica*, 1956, Vol. 22, pp. 289–292.

[10] R Core Team. *R: A language and environment for statistical computing*. 2019. R Foundation for Statistical Computing, Vienna, Austria.

https://www.R-project.org/.

Todos los códigos de este trabajo se encuentran en los siguientes notebooks de Jupyter:

[11] *El problema de la secretaria.* 2020.
Disponible en: https://colab.research.google.com/drive/1JBEkg_StnIh9DWNDVPTJWt1SebVWRT5y?usp=sharing

[12] *El problema de Cayley.* 2020.
Disponible en: https://colab.research.google.com/drive/1vcoqR7SaozB-s4_6kTxmJ0Xlyir1MczO?usp=sharing

[13] *Generalizaciones del problema de la secretaria.* 2020.
Disponible en: https://colab.research.google.com/drive/1b8U2XiUFTK6sJtpI4gudwrclrqLrPfL_?usp=sharing

The secretary problem

Sección de Matemáticas
Universidad de La Laguna

Laura Macía Vázquez
Facultad de Ciencias · Sección de Matemáticas
Universidad de La Laguna
alu0101001753@ull.edu.es

Abstract

Optimal stopping problems are those in which you have to choose a moment to take a certain action to maximize an expected reward. We will study in particular the secretary problem. We will start by understanding its solution and later we will program it through simulations, using recursion, and also creating generalizations of it.

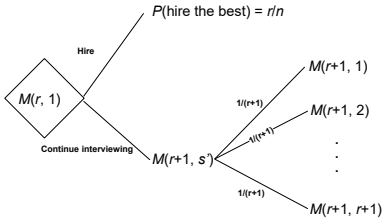
1. How to marry the right person

In 1611 the astronomer and mathematician Johannes Kepler lost his wife to illness. They had children and a house to take care of, so he decided to find a new wife to help him. Kepler began interviewing different women. He came to interview 11 women, taking notes of the entire process, and his search was not satisfactory. He only liked the fifth woman he met during the process but he still wanted to continue meeting some more women in case there was anyone better. Once all the meetings where finished he ended up alone. Honestly it is a pity that in those years the rule that guarantee the successful choice was not known. Anyway, in this work we wil explain it, so if someone has several candidates to get married, it is recommended to pay attention to the following.

2. The optimal stopping rule

In 1960, in Martin Gardner's column in the magazine *Scientific American*, the secretary problem appeared, and a little later, its solution. The problem consists of the following: we have a vacant position in an office and there are n candidates for the interview. Candidates can be ranked from best to worst, and they are interviewed in random order. Once you reject a candidate in the interview, they cannot be hired. In case you reach the end of the interviews, you will have to keep the last one candidate. The goal is to hire the best of all candidates. The optimal stopping rule is to interview r candidates out of the total n, and then hire the candidate who is better than all you have interviewed before.

A recursive function is defined that we will call $M(r, s)$, which denotes the maximum of the expected probability of finding the best candidate. Where s represents how good the candidate is, $s = 1 \dots n$. The probability that a candidate is the best is $\frac{r}{n}$. We can see the situation of the process when we interview candidate number r and it has a relative rank $s = 1$



Which leads to:

$$M(r,1) = \max\left\{\frac{r}{n}, \frac{1}{r+1}\sum_{s'=1}^{r+1} M(r+1,s')\right\} \tag{1}$$

$$M(r,s) = \frac{1}{r+1}\sum_{s'=1}^{r+1} M(r+1,s'). \tag{2}$$

With this, we can already calculate the expected probability of hiring each candidate, we can see if it is better to hire him, or continue interviewing.

3. Recursion

Recursion is a programming technique that is used to make a call to a function from itself, hence its name. The scope of recursion lies in problems whose solution can be found by solving the same problem. Most programming languages support recursion by allowing a function to call itself from within the program text. The advantage of its management is that it allows us to provide solutions to complex problems, but its problem is generally the execution time, which is very slow when we take large values.

References

[1] Ferguson,Who solved the Secretary Problem? *Statistical Science*,1989, Vol. 4, No. 3, pp. 282–296.

[2] Freeman,The Secretary Problem and its extensions. *International Statistical Review*, 1982,Vol. 51, pp. 189–206.

[3] Gardner,Mathematical Games. *Scientific American*, February 1960, Vol. 202, No. 2, pp. 150–156.

[4] Lindley,Dynamic Programming and Decision Theory. *Journal of the Royal Statistical Society*, 1960, Vol. 10, pp. 39–51.