Enunciado 2 - Noche en el Casino

Alejandro Sánchez de Castro Fernández — alejandro.sanchezdecastro.fernandez@alumnos.upm.es
Joaquín Jiménez López de Castro — jo.jimenez@alumnos.upm.es
Ángel Fragua Baeza — angel.fragua@alumnos.upm.es

13 de diciembre de 2021

1. Introducción

En esta práctica se pretende aplicar métodos de simulación para estudiar los resultados de jugar a la ruleta europea bajo distintas restricciones. Una ruleta europea consiste en treinta y siete números del $0 - 36$ con la misma probabilidad de ser escogidos. 18 números tienen asignado el color rojo y otros 18 el color negro, el 0 tiene el color verde. Se pueden hacer siete tipos distintos de apuesta:

1. **Apuestas 1 a 1**: Estas apuestas engloban las apuestas a color, a paridad y a mitades. Tienen $(37 - 1)/2 = 18$ resultados favorables posibles ya que el 0 nunca es un resultado favorable. La probabilidad P de acertar es $P = 18/37$.
2. **Apuestas 2 a 1**: Estas apuestas engloban las apuestas a filas y a columnas. En todas se puede apostar a 3 conjuntos de números disjuntos entre sí de tamaño 12, donde el 0 no forma parte de ninguno de ellos. La probabilidad de acertar es $P = 12/37$
3. **Apuesta 5 a 1**: Es el análogo al anterior para 6 conjuntos de tamaño 6. Se tiene $P = 6/37$.
4. **Apuesta 8 a 1**: Análogo al anterior para 9 conjuntos de tamaño 4. Se tiene $P = 4/37$
5. **Apuesta 11 a 1**: Análogo al anterior para 12 conjuntos de tamaño 3. Se tiene $P = 3/37$.
6. **Apuestas 17 a 1**: Engloba apuestas de dos números consecutivos en una fila o columna. Es análogo al anterior para 18 conjuntos de tamaño 2 y se tiene $P = 2/37$
7. **Apuesta 35 a 1**: Se apuesta a un solo número, que puede ser el 0. Naturalmente, $P = 1/37$.

Los nombres de las apuestas describen cuál es el premio por acertar, de forma que en una apuesta 2 a 1 si se apuesta una ficha y se acierta, se ganarán dos, en caso contrario, se habrá perdido la ficha.

Supóngase un jugador que va cada día al casino a jugar a la ruleta a lo largo de un mes, con las siguientes restricciones:

1. El jugador solo apuesta una ficha por partida.
2. El jugador siempre comienza con treinta fichas cada día.
3. El jugador se retira hasta el día siguiente si llega a las cero fichas, supera o iguala las ciento cincuenta fichas, o juega un máximo de cincuenta partidas.
4. El jugador elige el tipo de apuesta de forma equiprobable.

En este trabajo se busca obtener, modelando el proceso como una cadena de Markov en tiempo discreto, y mediante simulación:

A) Proporción de noches a lo largo de un mes que se retira por quedarse sin dinero.

B) Distribución y número medio de fichas a lo largo del mes.

C) Número medio de partidas jugadas antes de retirarse por bancarrota.

D) Modificando las restricciones anteriores para jugar un solo día sin límite de partidas, obtener la probabilidad de retirarse por bancarrota, y la opuesta, es decir, la de retirarse por superar o igualar las ciento cincuenta fichas.

E) Manteniendo las restricciones originales, hallar la distribución de probabilidad óptima $P = \{p_1, p_2, ..., p_7\}$ para la selección de cada tipo de apuesta que maximiza el número medio de fichas a lo largo del mes, utilizando para ello, la metaheurística de recocido simulado.

En este documento se va a explicar la aproximación para resolver cada una de las preguntas planteados en la Sección 2 , se van a analizar los resultados de las simulaciones en la Sección 3 y finalmente se darán unas conclusiones en la Sección 4. Se ha añadido en el Anexo A los resultados a las preguntas A, B y C haciendo uso de un único tipo de apuesta. Finalmente, en el Anexo B se muestra como instalar y lanzar el código implementado.

2. Desarrollo

2.1. Simulación de una cadena de Markov en tiempo discreto

En este apartado se detallará la solución mediante simulación de cadenas de Markov en tiempo discreto de las cuestiones A-D. En primer lugar, hay que justificar que el problema se pueda modelar de esta forma, al menos, el transcurso de un día en el casino. Definiendo el estado actual en la cadena como el número de fichas que tiene el jugador, el número de fichas del estado siguiente depende únicamente de las que se dispone en este momento, por lo que se verifica la propiedad de Markov de que "el futuro es independiente del pasado dado el presente". Además, se modela el tiempo de forma discreta, pues no se tiene en cuenta la duración de las partidas.

2.1.1. Diseño

La simulación de un mes en el casino se puede expresar con el pseudocódigo 1:

Algoritmo 1: `Simular_Mes_Casino`

Entrada: (P, f_i, M_p, M_f, B, G), Probabilidades de seleccionar cada jugada, fichas iniciales, máximo de partidas, máximo de fichas, beneficio de cada jugada, probabilidad de ganar de cada jugada
Resultado: (D, N_b, P_b), Distribución de fichas al final de cada día, promedio de días en bancarrota, promedio de partidas antes de bancarrota

inicio
  $(D, P_b, N_b) \leftarrow (\{0\}, 0, 0)$;
  para cada $dia \in [1, 30]$ **hacer**
    $(n, p) =$ `Simular_Dia_Casino`(P, f_i, M_p, M_f, B, G) ;
    $D_n \leftarrow D_n + 1$;
    si $n = 0$ **entonces**
      $P_b \leftarrow P_b + p$;
      $N_b \leftarrow N_b + 1$
    fin
  fin
  devolver $(D/30, N_b/30, P_b/N_b)$
fin
Function `Simular_Dia_Casino`$((P, f_i, M_p, M_f, B, G))$**:**
  Resultado: (n, p), Número de fichas, total de partidas jugadas
  $n \leftarrow f_i, p \leftarrow 0$;
  mientras $n \in [1, M_f]$ **y** $p < M_p$ **hacer**
    $j \leftarrow$ `Seleccionar_Jugada`(P) ;
    $u \sim U(0, 1)$;
    si $u < G_j$ **entonces**
      $n \leftarrow n + B_j$;
    en otro caso
      $n \leftarrow n - 1$;
    fin
    $p \leftarrow p + 1$;
  fin
  devolver *(n, p)*
Function `Seleccionar_Jugada`(P)**:**
  acc $\leftarrow 0$;
  $u \sim U(0, 1)$;
  para cada $p_i \in P$ **hacer**
    acc $\leftarrow$ acc$+p_i$;
    si $u <$ *acc* **entonces**
      devolver i
    fin
  fin

Haciendo muchas llamadas y promediando los resultados del algoritmo 1, se pueden obtener las respuestas a las cuestiones A-C. En cuanto a la cuestión D, se puede resolver con el mismo algoritmo, pero estableciendo el número de días a 1, y el número máximo de partidas a $M_p = \infty$.

Vale la pena mencionar que los apartados A, B y D se pueden resolver de forma exacta y en mucho menos tiempo utilizando la matriz de transiciones asociada a la cadena de Markov, como se describe en el pseudocódigo 2:

Algoritmo 2: `Simular_Casino_MatrizTransiciones`

Entrada: (P, M_p, M_f, B, G), Probabilidades de seleccionar cada jugada, máximo de partidas, máximo de fichas, beneficio de cada jugada, probabilidad de ganar de cada jugada
Resultado: (D, N_b), Distribución de fichas al final de cada día, promedio de días en bancarrota

inicio
- $M \leftarrow MatrizTransiciones(P, B, G, M_f)$;
- $S^{(0)} \leftarrow \{0\}$;
- $S^{(0)}_{30} \leftarrow 1$ // Estado inicial con 30 fichas ;
- $S^{(M_p)} \leftarrow (S^{(0)} \cdot M^{M_p})$ // Distribución tras M_p partidas;
- $D \leftarrow S^{(M_p)} \cdot 30$ // Normalizar a un mes;
- $N_b \leftarrow S^{(M_p)}_0$;
- **devolver** (D, N_b)

fin

La obtención de la matriz de transiciones es trivial, simplemente, se debe tener cuidado de que el estado asociado a la bancarrota y cualquiera en el que se tenga 150 fichas o más, solamente pueda transitar a sí mismo. El algoritmo funciona porque cada multiplicación de la distribución de probabilidades de cada estado inicial $S^{(0)}$ por la matriz de transiciones M, nos da la distribución de probabilidades $S^{(1)}$, de manera que $S^{(1)} = S^{(0)} \cdot M$ y por inducción, $S^{(n)} = S^{(0)} \cdot M^n$. De esta forma, se puede obtener la probabilidad de cada número de fichas al finalizar el día, que debería ser similar al promedio de los resultados obtenidos de simular un mes varias veces, por la ley de los grandes números.

Como el objetivo de esta práctica es utilizar simulación, y además, en principio este método no sirve para el apartado C, solamente se utilizará este método para contrastar resultados ya dados por la simulación.

2.1.2. Implementación

Para la implementación de este apartado se ha utilizado *Python*, con ayuda de la librería externa *numpy* para cálculos y *Jupyter, matplotlib* para ejecución de código y visualización de resultados. El primer detalle relevante de la implementación es el generador de números aleatorios utilizado. En este caso, se ha empleado el de *numpy*, *Permuted Congruential Generator (64-bit, PCG64)* de O'Neill (2014). Este generador destaca por su amplio periodo de 2^{128}, rapidez, y aunque no es el mejor para criptografía, se considera que tiene una calidad estadística excelente, de acuerdo con esta fuente. Por estas cualidades, se considera que es lo suficientemente bueno para utilizar en simulación.

Otra consideración es el tiempo de ejecución. Cuando hay que hacer muchas llamadas a *Simular_Mes_Casino*, el programa puede tardar mucho en finalizar, por ello, se han hecho algunas optimizaciones. En primer lugar, se paralelizan las llamadas a *Simular_Mes_Casino*. Esto es sencillo, pues son independientes entre sí. Para ello, se ha utilizado la librería *multiprocessing* de *Python*.

En segundo lugar, *Seleccionar_Jugada* involucra una selección sobre una distribución categórica. Una forma de optimizar este proceso es ordenar el vector de probabilidades P de mayor a menor antes de las llamadas a *Simular_Mes_Casino*, y mantener una referencia de las posiciones originales. De esta forma, es más probable acabar antes en el bucle. De hecho, haciendo este cambio, se observan tiempos hasta 7 veces más pequeños que utilizando la función *random.choice* de *numpy* para este mismo propósito.

2.2. Optimización de P con recocido simulado

En este apartado se desarrollará la solución adoptada para resolver el aparatado E, que consiste en obtener el mejor grupo de probabilidades P de seleccionar cada apuesta, de forma que se optimicen las ganancias medias a lo largo del mes. Para ello se hará uso de la metaheurística de Recocido Simulado, que intenta imitar el proceso de calentar metales para después enfriarlos lentamente de forma controlada, para intentar mejorar sus propiedades físicas. Este mecanismo permite obtener configuraciones óptimas para todo tipo de problemas evitando los óptimos locales, siempre y cuando se ajusten correctamente todos sus hiperparámetros. La función de energía se basa en el uso de la clase

2.2.1. Diseño

El algoritmo más básico y genérico para recocido simulado es el mostrado en el pseudocódigo 3:

Algoritmo 3: `Recocido Simulado`

Entrada: $(x_0, f, T_0, E(\cdot), F(\cdot), S(\cdot))$,
x_0: Solución inicial
f: Función de energía
T_0: Temperatura inicial, $T_0 > 0$
E: Definición de entorno
F: Predicado condición de parada
S: Función de actualización de temperatura
Resultado: (x^*, f^*), estimación de solución óptima y su energía

inicio
  $x^* \leftarrow x_0, \;\; f^* \leftarrow f(x_0), \;\; t \leftarrow 0$;
  mientras $\neg F()$ **hacer**
    $y \leftarrow seleccion\_aleatoria(E(x_t))$;
    si $f(y) < f^*$ **entonces**
      $x_{t+1} \leftarrow y, \;\; x^* \leftarrow y, \;\; f^* \leftarrow f(y)$;
    en otro caso
      $u \sim U(0,1)$;
      $p_t \leftarrow \exp(-(f(y) - f(x_t) \,/\, T_t))$;
      si $u < p_t$ **entonces**
        $x_{t+1} \leftarrow y$;
      en otro caso
        $x_{t+1} \leftarrow x_t$;
      fin
    fin
    $T_{t+1} = S(T_t)$;
    $t \leftarrow t + 1$
  fin
fin

Como se puede observar el algoritmo en sí es muy sencillo, pero a cambio hay que establecer una gran cantidad de hiperparámetros que determinaran el correcto funcionamiento del mismo. En cuanto al hiperparámetro de la solución inicial x_0 se ha escogido a partir del conjunto de probabilidades equiprobables de elección de cada apuesta, puesto que contamos con 7 tipos de apuestas distintos, nuestro conjunto inicial de probabilidades será $P = [1/7, 1/7, 1/7, 1/7, 1/7, 1/7, 1/7]$. La función de energía f escogida es la explicada en el pseudocódigo 1, en cual partiendo de las probabilidades de P extrae el número medio de fichas tras la simulación. Es importante mencionar que dicha media es multiplicada por -1, ya que el Recocido Simulado trata de optimizar un problema de minimización. La temperatura inicial T_0 se ha de establecer manualmente, típicamente para alcanzar tasas de aceptación del 98 %.

La función de cambio de entorno $E()$, ha sido probada con múltiples mecanismos de actualización pero la mayoría recaía en una modificación demasiado leve del entorno, es por ello que al final se ha decidido escoger una posición aleatoria del conjunto de probabilidades P y sustituir su contenido por un valor generado de forma aleatoria entre $[0, 1]$ seguido de una normalización de P para que se siga cumpliendo

el teorema de la probabilidad total, o lo que es lo mismo, que la suma de todas las probabilidades de P siga dando 1. En cuanto a la función que controla la función de parada $F()$ se han incorporado dos posibles controles, el primero es el más básico y simplemente comprueba que no se supere un máximo número de iteraciones, mientras que el segundo es más fiable ya que comprueba que pasadas k conjuntos de iteraciones de tamaño L el porcentaje de soluciones aceptadas por el algoritmo sea menor que un porcentaje de aceptación p_{acc}, en cuyo caso se para el algoritmo ya que significaría que habría convergido. Por último la función de actualización de temperatura decrece la misma de forma escalonada y exponencial siguiendo la formula típica de $T_{hL} = \alpha^h T_0$, donde la temperatura se actualiza cada L iteraciones.

2.2.2. Implementación

Para la implementación de este apartado, de nuevo, se ha usado Python, con ayuda de la librería externa *simanneal* de la que se obtiene una versión simplificada del algoritmo Recocido Simulado, junto con *Jupyter, matplotlib* para ejecución de código y visualización de resultados. En el Anexo B se explica como lanzar el código y liberías asociadas.

El principal motivo por el que se eligió esta librería y no otra cualquiera, es porque además de tener una clase que implementa el algoritmo de Recocido Simulado, de tal forma que permite expandirla de una forma muy sencilla, también contaba con una función llamada **auto(minutes, steps)** que devuelve una temperatura inicial y final adecuada para el problema en cuestión. Para ello, internamente lanza el algoritmo de recocido simulado repetidas veces, en primera instancia modificando la temperatura inicial hasta que la tasa de aceptación de soluciones ronda el $98\,\%$, y una vez se cumpla dicha tasa de aceptación continua modificando la temperatura máxima hasta que la tasa de mejora de las soluciones alcanza un $0\,\%$. El parámetro de *minutes* sirve para estimar el número de iteraciones máximas que podrá hacer en dicho tiempo medido en minutos, mientras que *steps* determina el número de iteraciones que realizará el Recocido Simulado antes de cada actualización de temperatura inicial y final.

En el código se ha añadido una función de graficado, en donde una vez terminada una ejecución del Recocido Simulado muestra la evolución de la temperatura, la energía, la mejor energía y la tasa de aceptación a lo largo de las iteraciones tal y como se puede ver en la Figura 1.

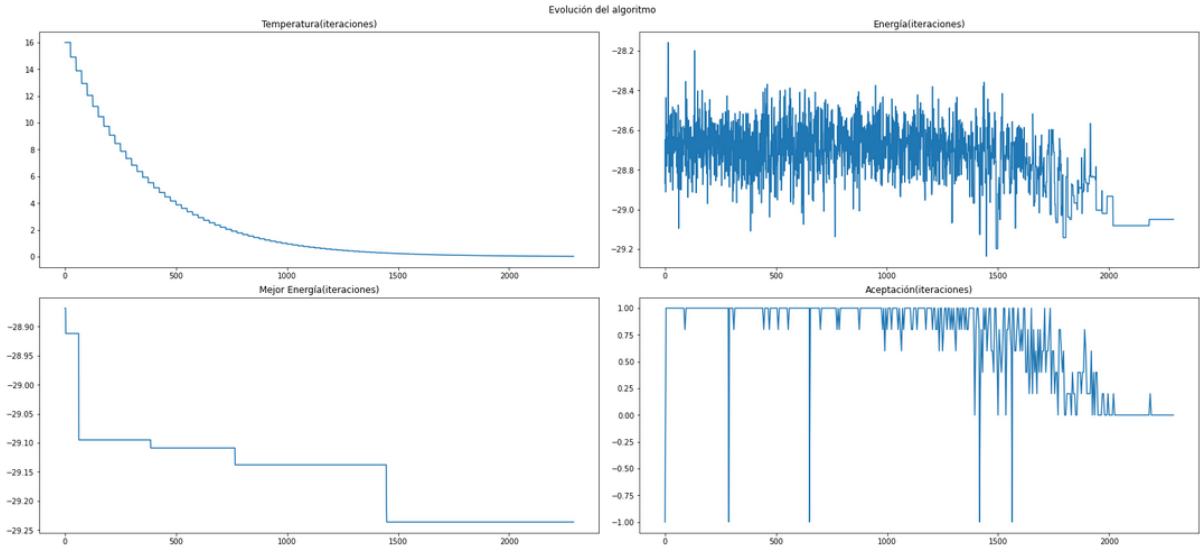


Figura 1: Ejemplo del resultado de Recocido Simulado

Como último detalle de implementación, se ha tenido que implementar dentro del algoritmo un detector de interrupciones como es el caso del típico *Ctrl+C* para poder parar ejecuciones de forma controlada guardando en un archivo el estado actual del algoritmo para poder retomarlo en cualquier otro momento. Esta decisión se tomo ya que para realizar una prueba medianamente realista, la ejecución iba a tardar múltiples días y no se disponía de ningún dispositivo el cual pudiera estar ejecutando tanto tiempo seguido sin interrupciones el algoritmo.

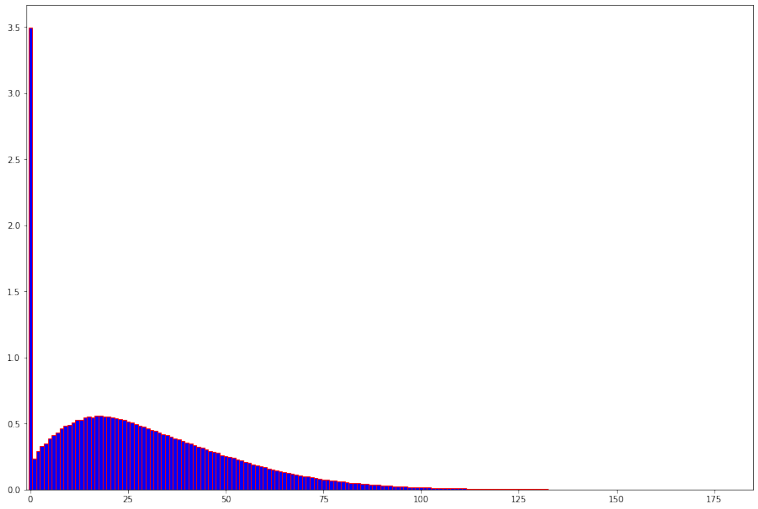
3. Resultados

3.1. Simulación de la cadena de Markov

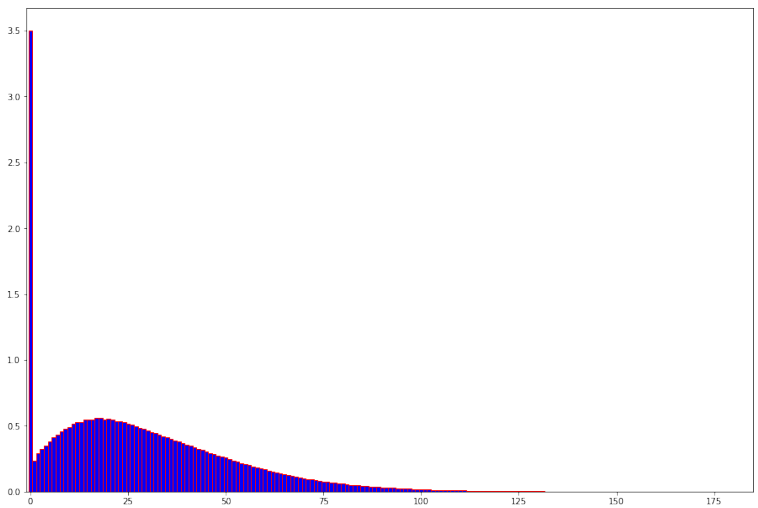
Se han realizado múltiples pruebas para comprobar y medir los posibles resultados. En la primera de ellas se han realizado 1000000 simulaciones de 30 días para obtener los resultados que se van a exponer a continuación. En estos resultados se da respuesta a los apartados A), B) y C) de la Sección 1. Donde es posible, se contrastan los resultados con el método de matriz de transiciones.

Apartado A. Proporción de días en bancarrota: $0{,}1165$. Resultado obtenido con matriz de transiciones: $0{,}1166$.

Apartado B. Distribución de fichas al final del día. Promedio: $28{,}6632$, desviación estándar: $23{,}5887$. Con matriz de transiciones, promedio: $28{,}6756$. La distribución se muestra en la Figura 2.



(a) Estimación mediante simulación



(b) Estimación mediante matriz de transiciones

Figura 2: **Distribución de fichas al final día**

En este caso, la distribución resultante del número de fichas al final del día recuerda a una binomial negativa.

Apartado C. Partidas medias antes de la bancarrota: $41{,}4835$

Para el apartado D), se han hecho también 1000000 simulaciones, pero de un solo día y sin límite de

partidas, y se ha obtenido el siguiente resultado:

Apartado D. Proporción de días en bancarrota: $0{,}1165$.

Como se puede observar, la estrategia equiprobable es buena si queremos pasar mucho rato en el casino, pues si se acaba en bancarrota, en promedio es tras 41 partidas. Además, solo uno de cada 10 días aproximadamente se vuelve a casa con los bolsillos vacíos. Por supuesto, a largo plazo se pierde dinero, por la presencia del 0 en la ruleta, margen de beneficio del casino.

Para tener una mejor idea de la diferencia que hace cambiar el vector de probabilidades P, se incluye en el Anexo A los resultados de los apartados A, B y C cuando solo se hace un tipo de apuesta, haciendo 1000000 de simulaciones para cada tipo.

3.2. Optimización de P con Recocido Simulado

Sin ninguna duda, este apartado es el que más problemas ha dado, y el principal motivo por el que se ha acabado optimizando tanto el código (e.g. introducción de multithreading).

El primer problema con el que nos enfrentamos fue la lentitud del código, motivo por el cual solo se podrían hacer unas pocas iteraciones y los resultados obtenidos no eran nada buenos. El segundo problema fue que el cambio de entorno inicial que se planteó en el que solo se decrementaba o aumentaba una cantidad pequeña sobre una de las probabilidades al azar para después normalizarlas todas, realizaba cambios mínimos en el entorno y por ello no se hacía una exploración real de todo el entorno. Por ello, se decidió cambiar el cambio de entorno escogiendo una probabilidad al azar y modificándola por completo generando un nuevo número aleatoria entre $[0, 1]$ seguido del proceso de normalización, aumentando así la exploración sobre el espacio de búsqueda.

Finalmente, el resultado que se va a exponer ha sido calculado usando como función de energía la simulación del casino como una cadena de Markov, en donde se han realizado para cada iteración 7500 simulaciones de 30 días, partiendo cada día de 30 fichas y teniendo un límite máximo de 50 partidas cada noche y la obligación de retirarse ya sea por bancarrota o por alcanzar las 150 fichas. Además como hiperparámetros del propio Recocido Simulado se ha establecido una temperatura máxima de 16 que puede decrecer hasta una temperatura mínima de $1{,}1e^{-05}$, la cual se actualiza cada $L = 25$ iteraciones y cuyo criterio de parada viene determinado por un número máximo de iteraciones igual a 5000, en donde cada $k = 4$ conjuntos de L iteraciones se comprueba si la tasa de aceptación es menor a $p_{acc} = \frac{1}{k*L} = 1/100$ en cuyo caso se para el algoritmo.

El resultado de la ejecución de recocido simulado con los hiperparámetros anteriores ha sido

$$P = [0{,}0024, 0{,}0099, 0{,}1765, 0{,}0059, 0{,}0745, 0{,}4902, 0{,}2406]$$

donde la energía final que corresponde con la media de fichas a lo largo de la mejor simulación ha sido de $29{,}2363$ fichas. El resultado de la simulación se puede observar en la Figura 3. Cabe destacar como la energía decrece exponencialmente pero de forma escalonada. La energía como esa previsible inicialmente varía mucho ya que se encuentra en su fase exploratoria, mientras que al final se estabiliza ya que empieza a converger según decrece la temperatura. Los valores de -1 en la tasa de aceptación se corresponden con las interrupciones del algoritmo cuando la máquina dejaba de estar disponible (la ejecución ha durado varios días).

```
Mejores probabilidades: [0.0024163977794117958, 0.009854513439548228, 0.17652626705805785, 0.0058777893904716876, 0.0745480819397543
3, 0.49017240121885447, 0.24060454917390167]
Fichas medias: 29.236284444444546
Nº epochs 2600
```

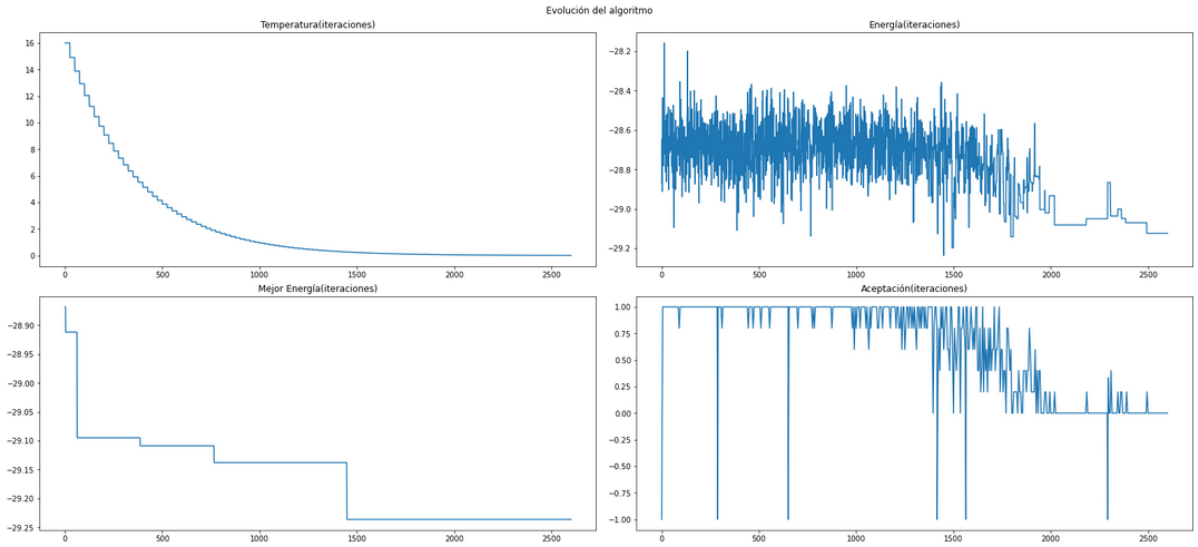


Figura 3: Apartado E. Optimización de P mediante Recocido Simulado

Para asegurarnos de que las probabilidades de la ejecución del Recocido Simulado han obtenido una media honesta, se ha lanzado de nuevo la simulación del apartado A, B y C para corroborar que la media de fichas es parecida, pero aumentando el número de simulaciones de 7500 a 50000. Como era de esperar al aumentar el número de simulaciones la media se ve ciertamente decrecida dando como resultado $28{,}7759$, tal y como se puede observar en la Figura 4. Esto nos indica que la energía alcanzada en el recocido simulado ha sido por mera suerte; lamentablemente, 7500 simulaciones era el número factible máximo para el recocido simulado por la carga computacional.

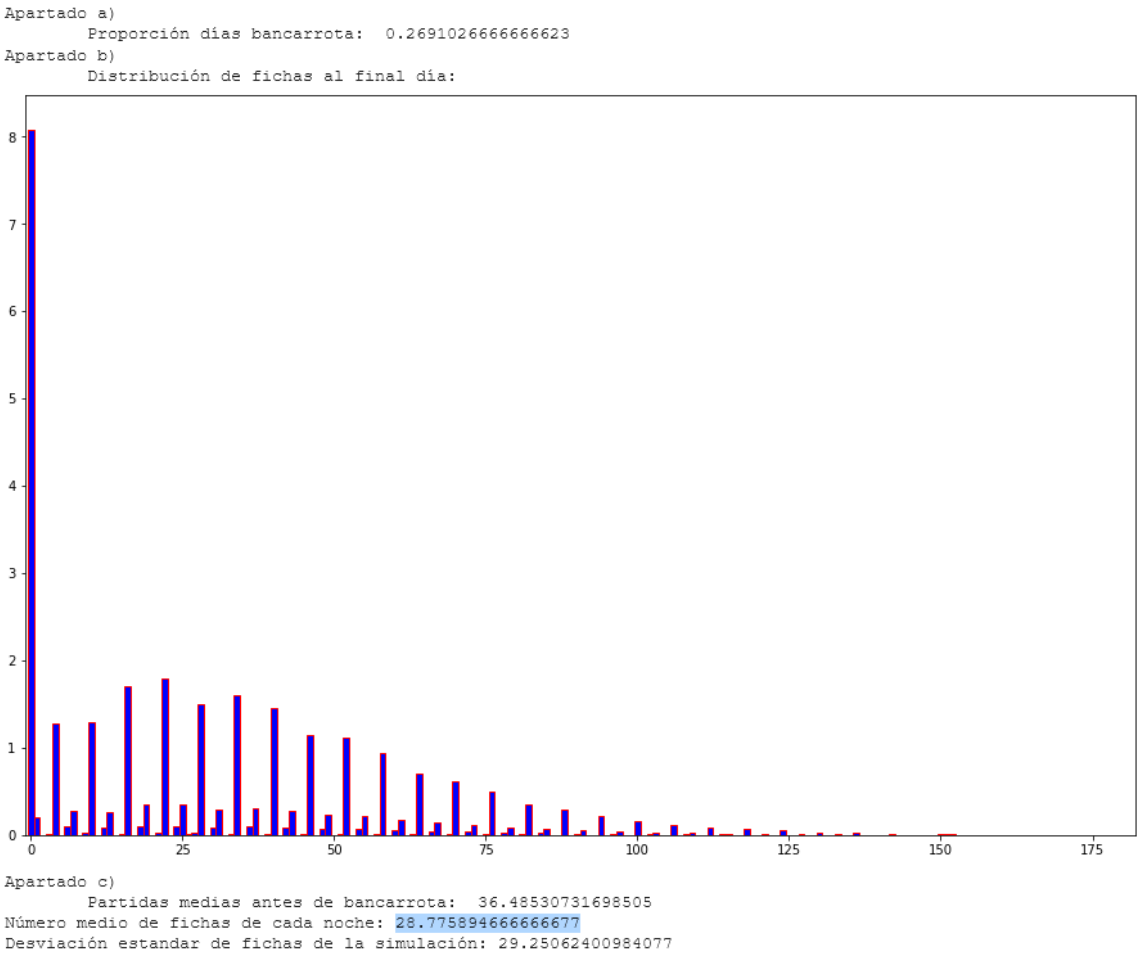
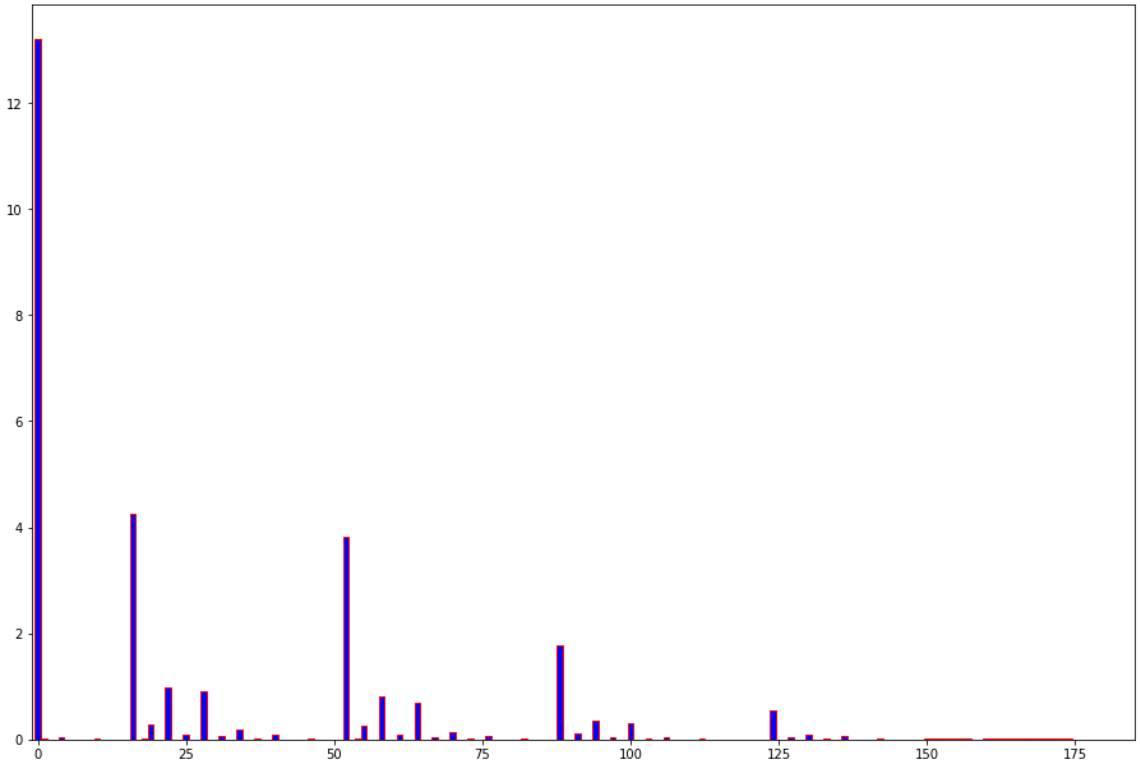


Figura 4: Comprobación del resultado P obtenido del Recocido Simulado

No contentos con el resultado final, se ha aplicado un proceso de búsqueda local sobre el conjunto de probabilidades P que había obtenido el Recocido Simulado. Para ello, se prueba a incrementar un valor a cada una de las probabilidades por separado seguido de una normalización. En caso de que incrementar alguna probabilidad mejore el resultado se sigue mejorando a partir de ella. En caso de que ningún incremento mejore la solución se decrementa el valor del incremento y se repite el proceso hasta que el incremento sea menor a un valor establecido. Tras la realización de esta búsqueda local se ha obtenido una lista de probabilidades tal que $P = [0{,}0001, 0{,}0040, 0{,}0249, 0{,}0012, 0{,}0366, 0{,}0001, 0{,}9332]$. De nuevo se volvió a realizar la simulación de los apartados A, B y C para comprobar la media de fichas obtenida que fue igual a $28{,}8811$, dando evidencias de que había mejorado tal y como se observa en la Figura 5.

```
Apartado a)
        Proporción días bancarrota:  0.44003583333286334
Apartado b)
        Distribución de fichas al final día:
```




```
Apartado c)
        Partidas medias antes de bancarrota:  31.51934166968239
Número medio de fichas de cada noche: 28.881133833333347
Desviación estandar de fichas de la simulación: 37.19073050901568
```

Figura 5: Comprobación del resultado P obtenido tras la mejora mediante búsqueda local

El resultado indica que la forma de obtener la mayor cantidad de beneficio, que claramente es perder la menor cantidad de fichas, se consigue dando mayor prioridad a la apuesta que mayor riesgo tiene (Apuesta 35 a 1, que implicar apostar a un único número) pero a la vez mayores ganancias. A largo plazo, se pierde menos dinero con esta apuesta, pero se pasa mucho menos tiempo cada noche en el casino en promedio, como se ve en el promedio de partidas antes de bancarrota y probabilidad de terminar en bancarrota.

Para comprobar que los resultados obtenidos fueran los esperados, se ha vuelto a ejecutar el algoritmo de Recocido Simulado, pero esta vez en vez de usar como función de energía la simulación de las cadenas de Markov, se ha usado el método mencionado en el pseudocódigo 2, el cual tarda mucho menos en ejecutarse. Los resultados observables en la Figura 6, aportan unos resultados muy similares a los obtenidos anteriormente, donde las probabilidades resultantes $P = [0{,}0004, 0{,}006, 0{,}0008, 0{,}0002, 0{,}0001, 0{,}0001, 0{,}9978]$ de nuevo le otorga prácticamente todo el peso a la Apuesta 35 a 1, dando un número de fichas medias de $28{,}8903$.

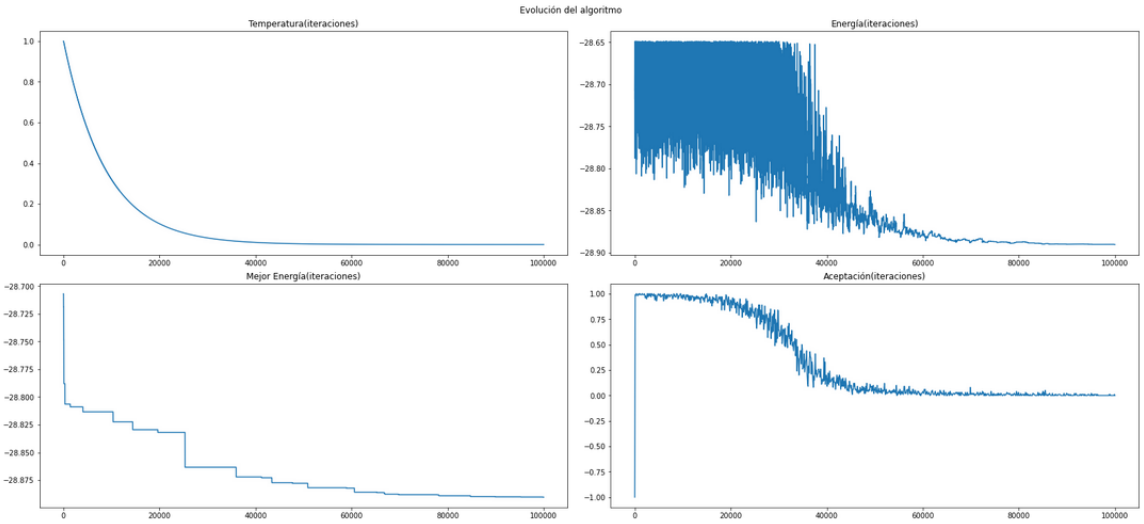


Figura 6: Comprobación del resultado P a través de Recocido Simulado usando matriz de transiciones

4. Conclusión

En la realización de este trabajo se ha observado el efecto de distintas estrategias bajo unas determinadas restricciones de jugar a la ruleta en el casino. En general se ha visto que pese a ser un juego de azar, los tipos de apuesta elegidos afectan a la experiencia de juego, donde algunas permiten pasar más tiempo jugando, pero implican mayores pérdidas a largo plazo, mientras que otras aceleran la bancarrota cada día, pero implican menores pérdidas a largo plazo. Naturalmente, todas implican perder dinero a largo plazo.

También se ha entrado en contacto con la dificultad de realizar simulaciones tan costosas computacionalmente y lo esencial de optimizar el código en estas situaciones. Quizás habría que haber considerado al principio utilizar lenguajes más rápidos como C, que aunque tienen más tiempo de desarrollo, son más rápidos en ejecución, además de ser más verdes (recordemos que se han tenido máquinas encendidas a lo largo de varios días a máxima potencia, que involucra un gasto significativo de energía).

Como puede observarse en las pruebas realizadas sobre un solo tipo de apuesta, cuanto menos segura sea ésta más probabilidad existe de acabar en bancarrota y más desplazada para la izquierda estará la distribución de fichas.

A. Apartados A,B,C haciendo un solo tipo de apuesta

A.1. Apuesta única 1 a 1

Apartado A. Proporción de días en bancarrota: $3{,}4000 \times 10^{-5}$.

Apartado B. Distribución de fichas al final del día. Promedio: $28{,}6510$, desviación estándar: $7{,}1924$.

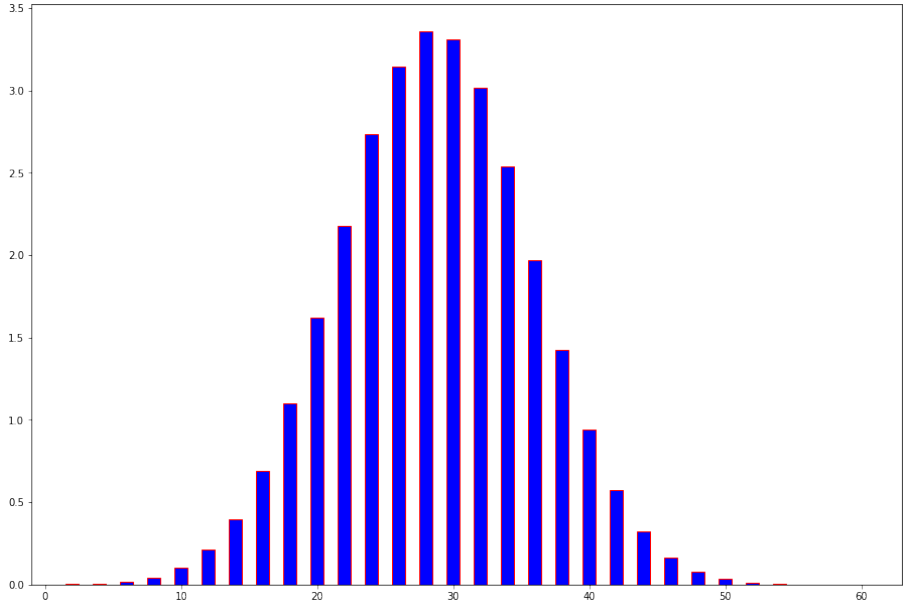


Figura 7: **Distribución de fichas al final día apostando solo 1 a 1**

Apartado C. Partidas medias antes de la bancarrota: $46{,}9412$

A.2. Apuesta única 2 a 1

Apartado A. Proporción de días en bancarrota: $0{,}0017$.

Apartado B. Distribución de fichas al final del día. Promedio: $28{,}6509$, desviación estándar: $10{,}0999$.

Apartado C. Partidas medias antes de la bancarrota: $44{,}1707$

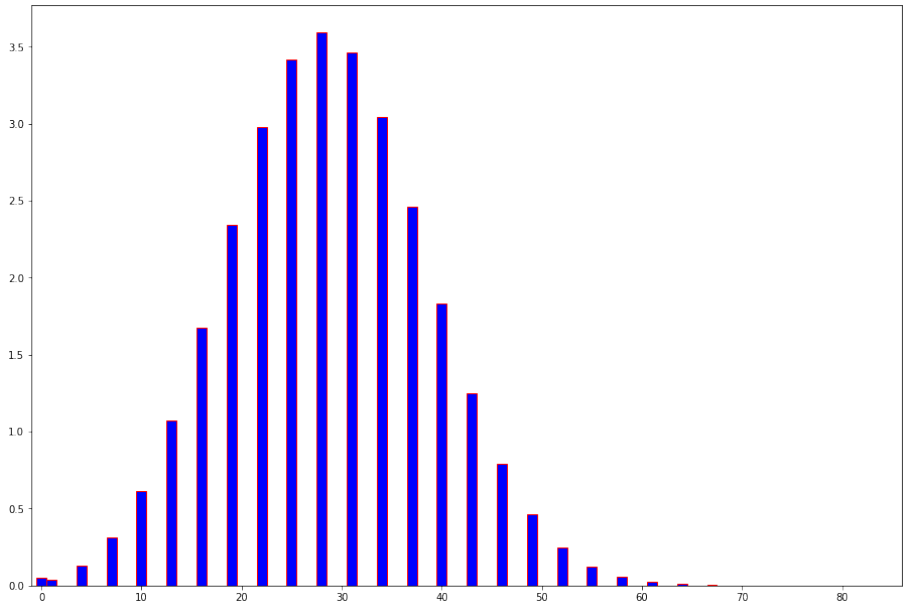


Figura 8: **Distribución de fichas al final día apostando solo 2 a 1**

A.3. Apuesta única 5 a 1

Apartado A. Proporción de días en bancarrota: 0,0446.

Apartado B. Distribución de fichas al final del día. Promedio: 28,6632, desviación estándar: 15,8336.

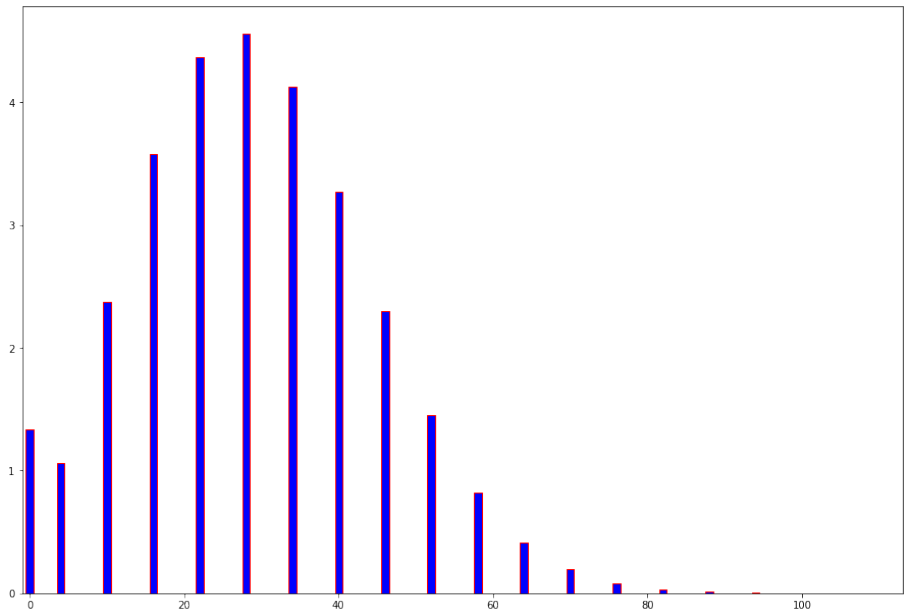


Figura 9: **Distribución de fichas al final día apostando solo 5 a 1**

Apartado C. Partidas medias antes de la bancarrota: 41,5051

A.4. Apuesta única 8 a 1

Apartado A. Proporción de días en bancarrota: 0,1169.

Apartado B. Distribución de fichas al final del día. Promedio: 28,6855, desviación estándar: 19,8154.

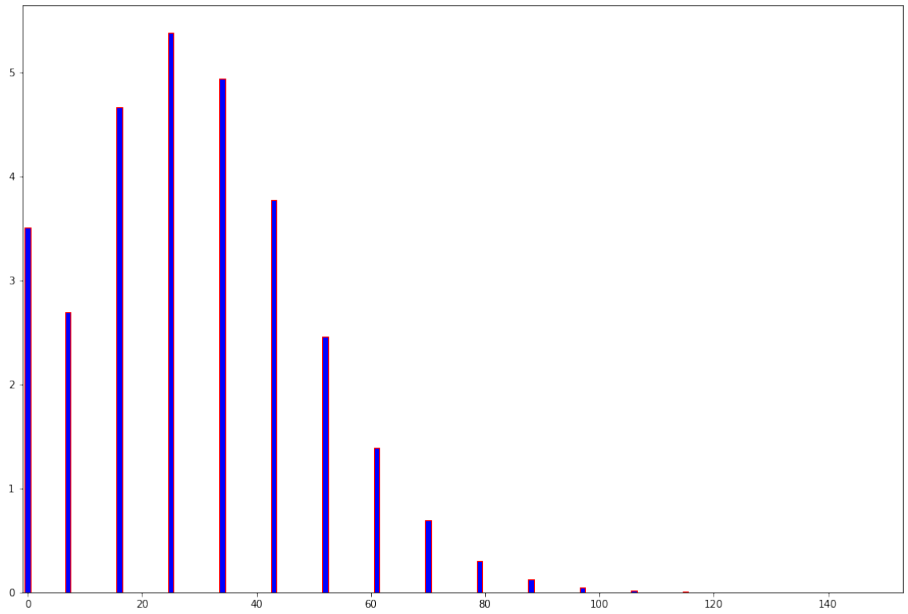


Figura 10: **Distribución de fichas al final día apostando solo 8 a 1**

Apartado C. Partidas medias antes de la bancarrota: 39,7936

A.5. Apuesta única 11 a 1

Apartado A. Proporción de días en bancarrota: 0,1549.

Apartado B. Distribución de fichas al final del día. Promedio: 28,7114, desviación estándar: 22,9551.

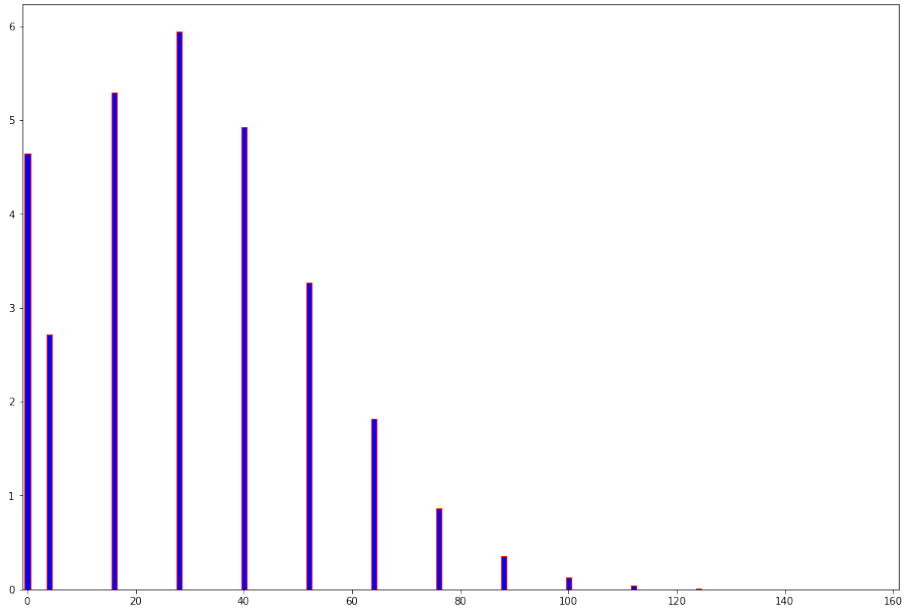


Figura 11: **Distribución de fichas al final día apostando solo 11 a 1**

Apartado C. Partidas medias antes de la bancarrota: 35,8717

A.6. Apuesta única 17 a 1

Apartado A. Proporción de días en bancarrota: 0,3077.

Apartado B. Distribución de fichas al final del día. Promedio: 28,7728, desviación estándar: 27,9649.

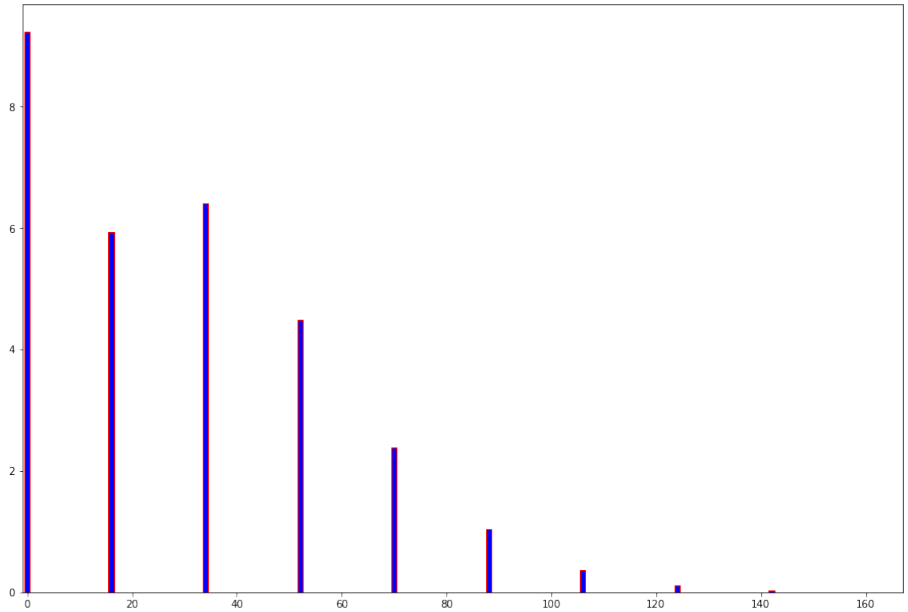


Figura 12: **Distribución de fichas al final día apostando solo 17 a 1**

Apartado C. Partidas medias antes de la bancarrota: 36,9554

A.7. Apuesta única 35 a 1

Apartado A. Proporción de días en bancarrota: 0,4394.

Apartado B. Distribución de fichas al final del día. Promedio: 28,9015, desviación estándar: 37,8814.

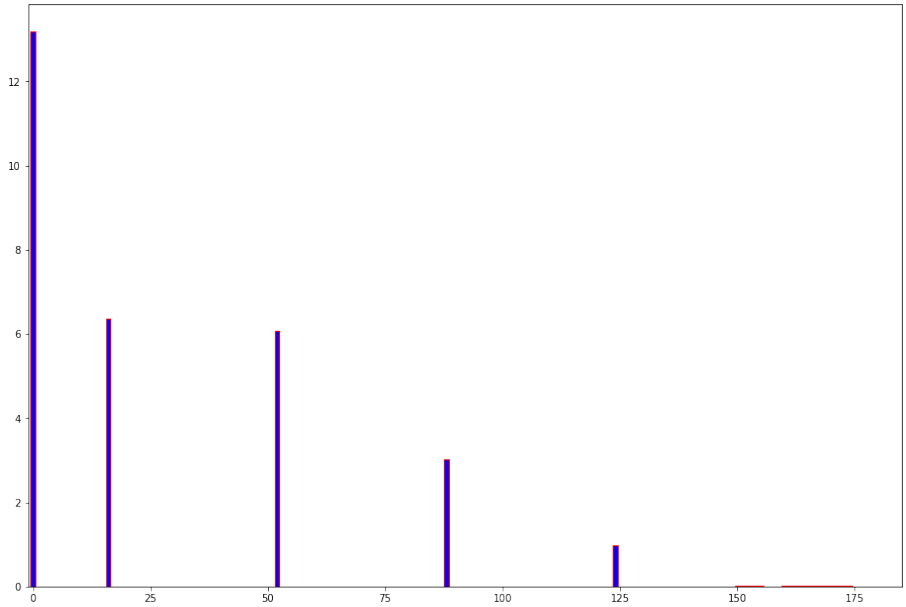


Figura 13: **Distribución de fichas al final día apostando solo 35 a 1**

Apartado C. Partidas medias antes de la bancarrota: 30,0000

B. Código

La implementación de esta práctica se ha hecho en Python, donde las librerías asociadas se gestionan en un entorno virtual usando Pipenv. Partiendo de la raíz del código adjunto a esta entrega, estas librerías quedan reflejadas en los ficheros **Pipfile** y**Pipfile.lock**. Para instalarlas se puede seguir el siguiente proceso:

Command Line

```
/:$ pip3 install pipenv # pip en Windows
/:$ pipenv install
```

Una vez hecho eso, se puede lanzar el Jupyter Notebook que lanza los experimentos mencionados, llamado **casino.ipynb**. Las funcionalidades de librería están en el *script* de Python **casino.py**.

Command Line

```
/:$ pipenv shell
/:$ jupyter-notebook casino.ipynb
```

Referencias

O'Neill, M. E. (2014). PCG: A family of simple fast space-efficient statistically good algorithms for random number generation. En: *ACM Transactions on Mathematical Software*.