Presentación práctica Backtracking y Branch and Bound

Asignatura: Algorítmica

Rubén Morales Pérez Francisco Javier Morales Piqueras
Bruno Santindrian Manzanedo Ignacio de Loyola Barragan Lozano Francisco Leopoldo Gallego Salido

7 de junio de 2016

Índice

Enunciado

Equipos

Se desea dividir un conjunto de n personas para formar dos equipos que competirán entre sí. Cada persona tiene un cierto nivel de competición, que viene representado por una puntuación (un valor numérico entero). Con el objeto de que los dos equipos tengan un nivel similar, se pretende construir los equipos de forma que la suma de las puntuaciones de sus miembros sea lo más similar posible. Diseña e implementa un algoritmo vuelta atrás para resolver este problema.

Puntuación de cada jugador

Nivel individual

Tenemos n jugadores con puntuación respectiva $x_i \ \forall i \in I = [1, 1000] \cap \mathbb{N}$. Clasificamos a cada jugador solamente por su puntuación.

¿Problema?

Esto permite que si hay dos jugadores con nivel de competición x e y con $x = y$ y cada uno está en un equipo podrían intercambiar sus posiciones sin descompensar los equipos. Así obtenemos una mayor flexibilidad, por si queremos tener en cuenta factores de compenetración entre los jugadores dependiendo del equipo en el que estén.

Rango inicial diferente

Si inicialmente tuviésemos los niveles de los jugadores en otro rango $J = [a, b]$ $a, b \in \mathbb{N} : a < b$ podemos dejarlo así, ya que el rango de los niveles no afecta al problema.

Excepción

Si aplicamos una transformación que reduzca el rango, podríamos perder información al aproximar a números enteros, ya que el enunciado pide que sean números enteros.

Transformación

En cualquier caso la transformación al dominio del problema (números naturales dentro de I) del intervalo J es: $\forall j \in J$ aplicamos una función $F : [a, b] \to [1, 1000] \cap \mathbb{N}$

Función $E(x)$

Teniendo en cuenta que $E(x)$ es la función parte entera:

Fórmula

$$F(x) = \begin{cases} E\left(\frac{x-a}{b-a} \cdot 1000\right) + 1 & \text{si } \frac{x-a}{b-a} \cdot 1000 < E\left(\frac{x-a}{b-a} \cdot 1000\right) + \frac{1}{2} \\ E\left(\frac{x-a}{b-a} \cdot 1000\right) & \text{en otro caso} \end{cases}$$

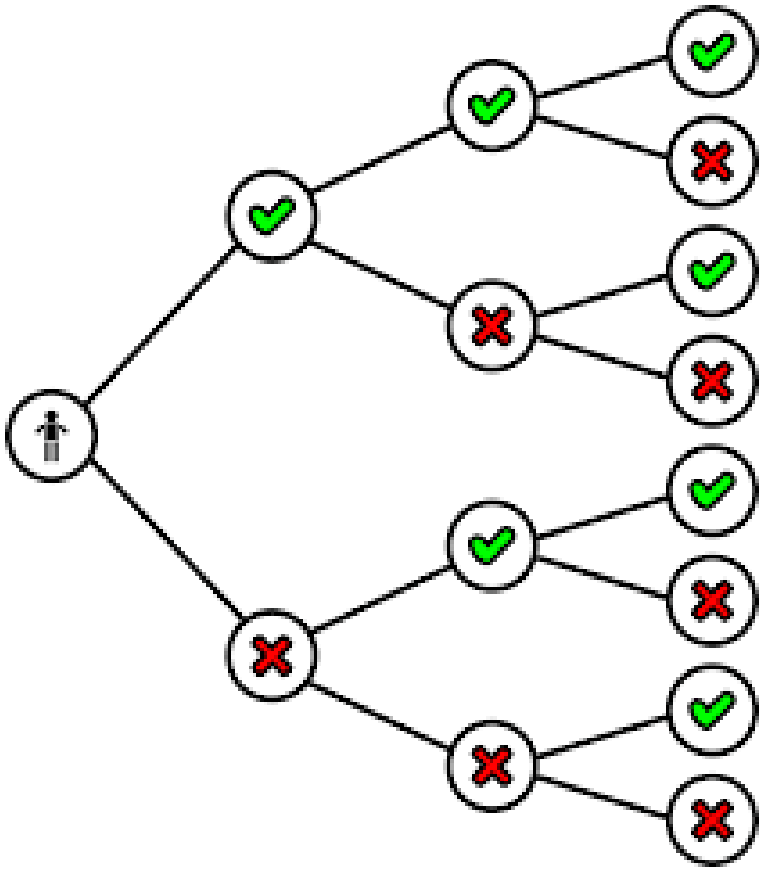
Desarrollo

Tomando el primer jugador, puede estar en un equipo o en otro, 2 combinaciones. Cogemos el siguiente jugador, puede estar en uno u otro, independientemente de donde estuviese el anterior, 2^2 combinaciones. El tercer jugador vuelve a tener 2 opciones independientes del resto, 2^3 combinaciones.
Tenemos que pasar por todas las opciones, por tanto la eficiencia del algoritmo es $O(n) = 2^n$

Representación

Podemos representar el problema como un vector de bool de n elementos. Si un jugador está en el primer equipo el valor será *true*, en caso contrario será *false*.

Al tratarse de un algoritmo de vuelta atrás (backtraking) tenemos que recorrer todas las opciones, por lo que el problema queda representado como un árbol binario donde cada nodo tiene dos opciones, estar en un equipo o en otro.



Otra versión

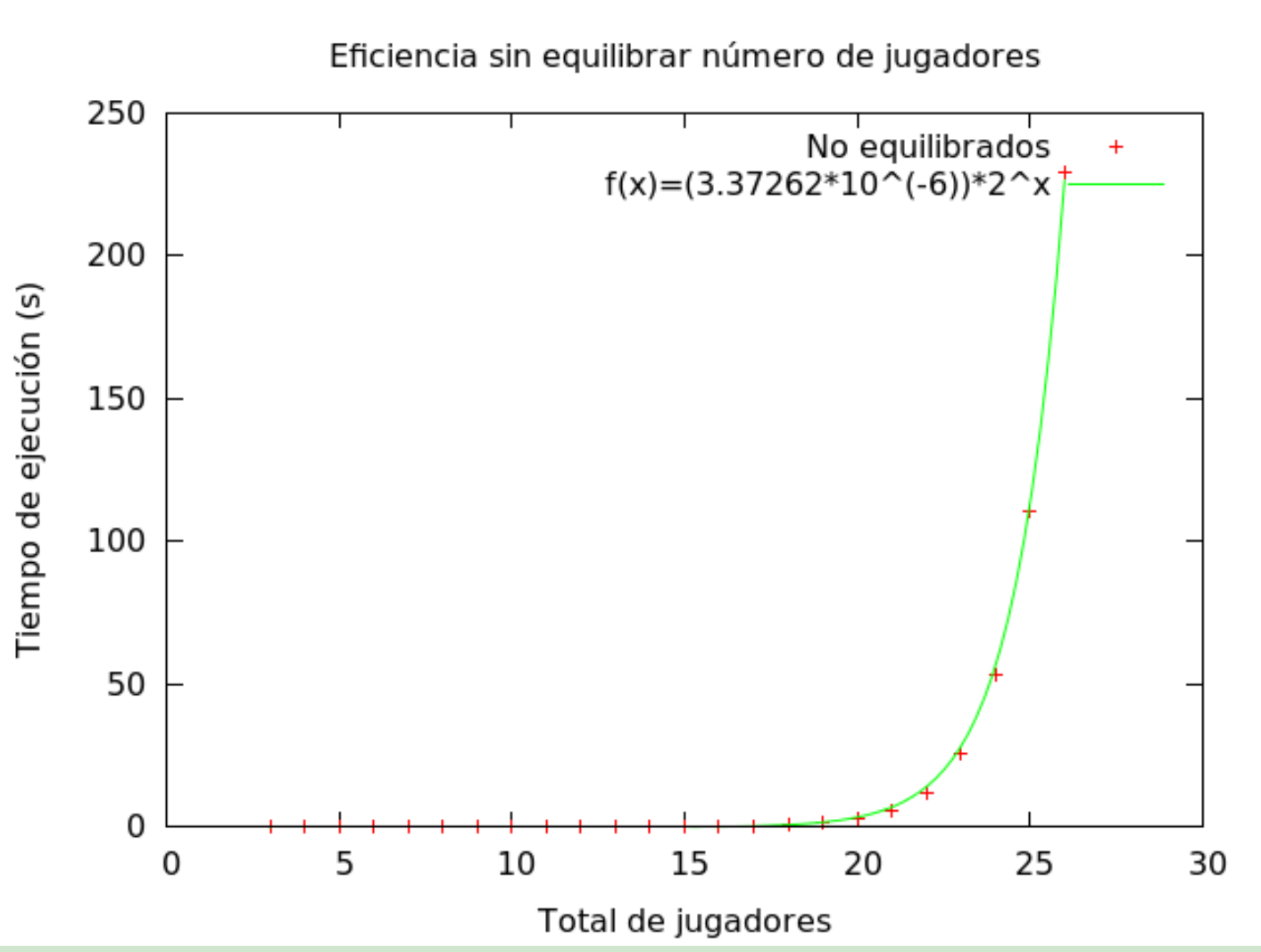
Mejora

Forzaremos que la diferencia de jugadores entre ambos equipos no sea mayor de 1. Si hay un número n par los equipos tendrán el mismo número de jugadores. El algoritmo sigue teniendo eficiencia $O(n) = 2^n$ ya que pasaremos por todas las opciones, pero cuando una combinación no cumpla dicha restricción no haremos sus cálculos asociados.

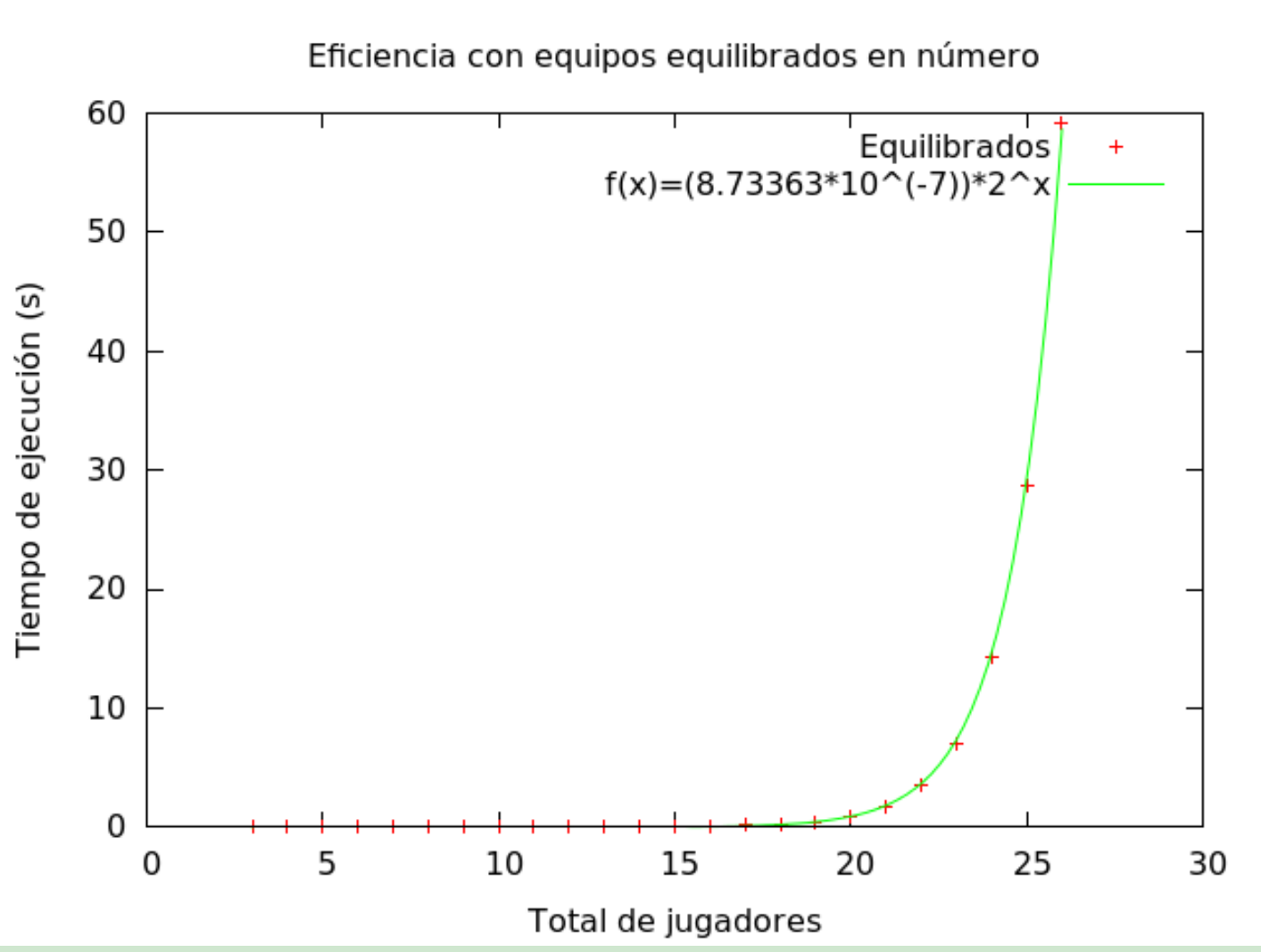
Ajustes

El rango usado ha sido $I = [3, 26] \cap \mathbb{N}$. Hemos ajustado funciones $f(n) = a \cdot 2^n \ a \in \mathbb{R} \ \forall n \in I$. Sus coeficientes r^2 son 1 en ambos casos (hay pocos elementos) y el término a está especificado en las gráficas.

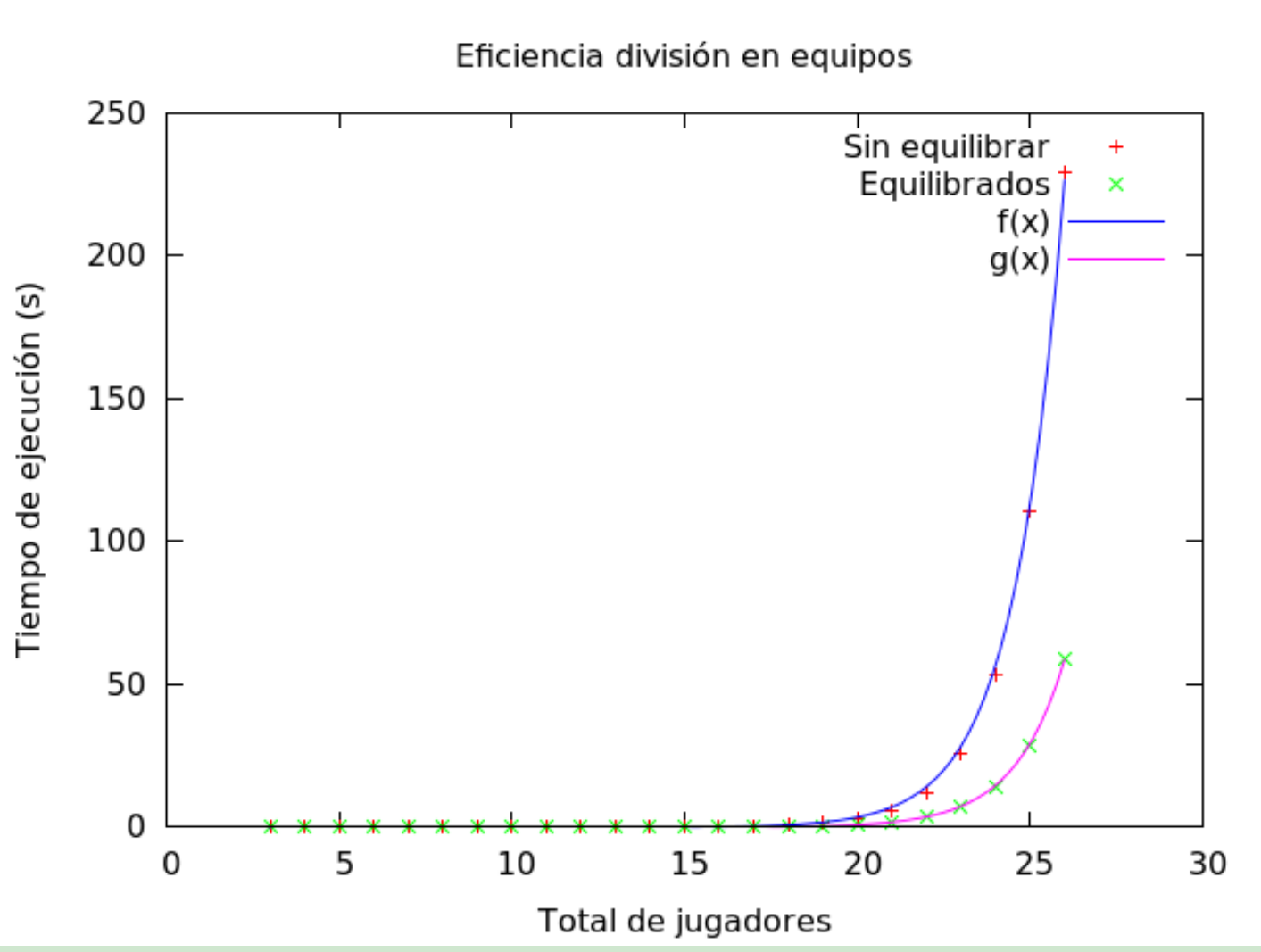
Sin equilibrio de jugadores



Equilibrando jugadores



Comparando tiempo



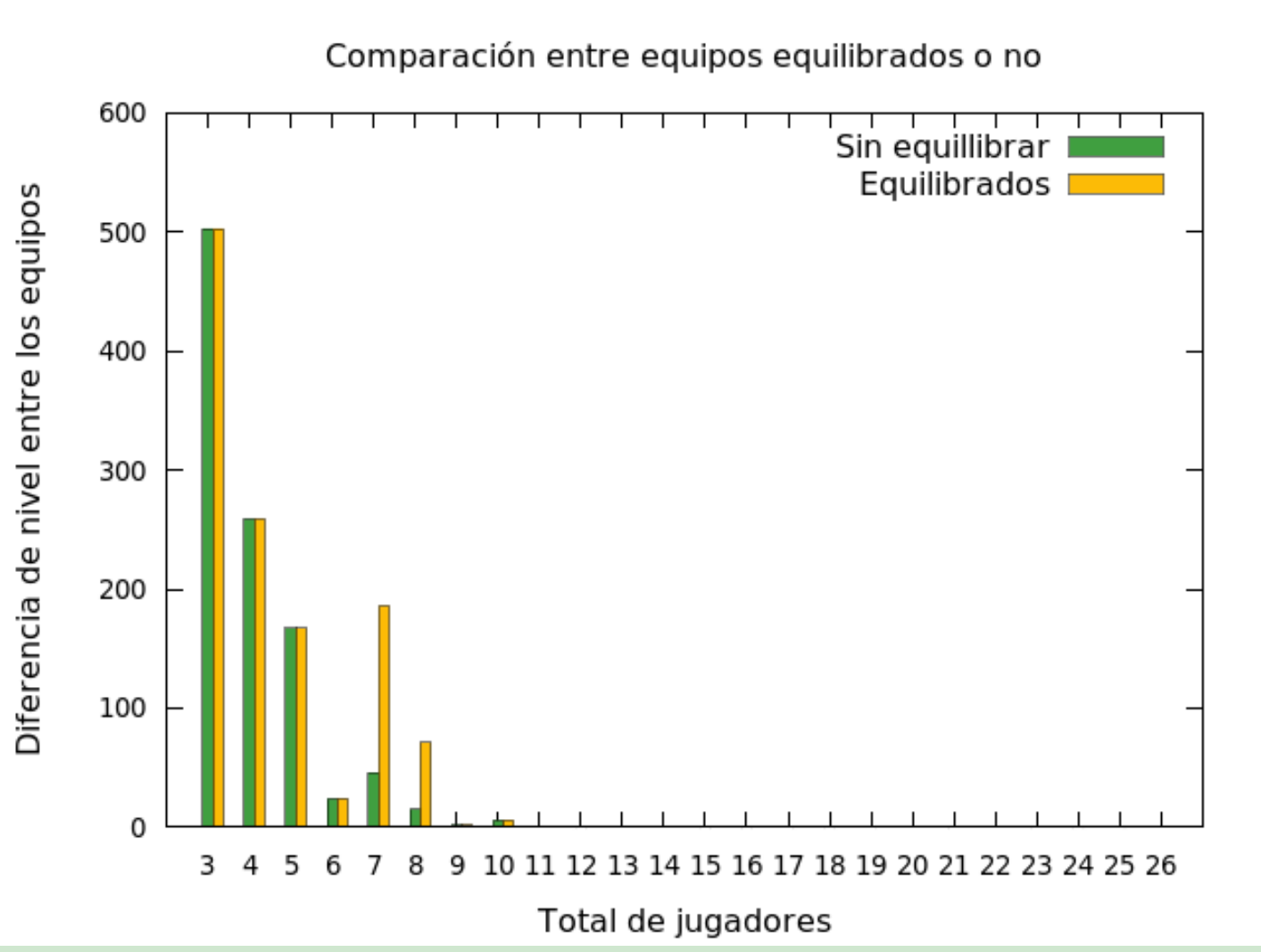
Nivel

Recordemos que un jugador tiene un nivel entre 1 y 1000, es lógico pensar que con pocos jugadores el margen de optimización del algoritmo es menor debido al gran peso que toma la aleatoriedad con la que se determinan los niveles.

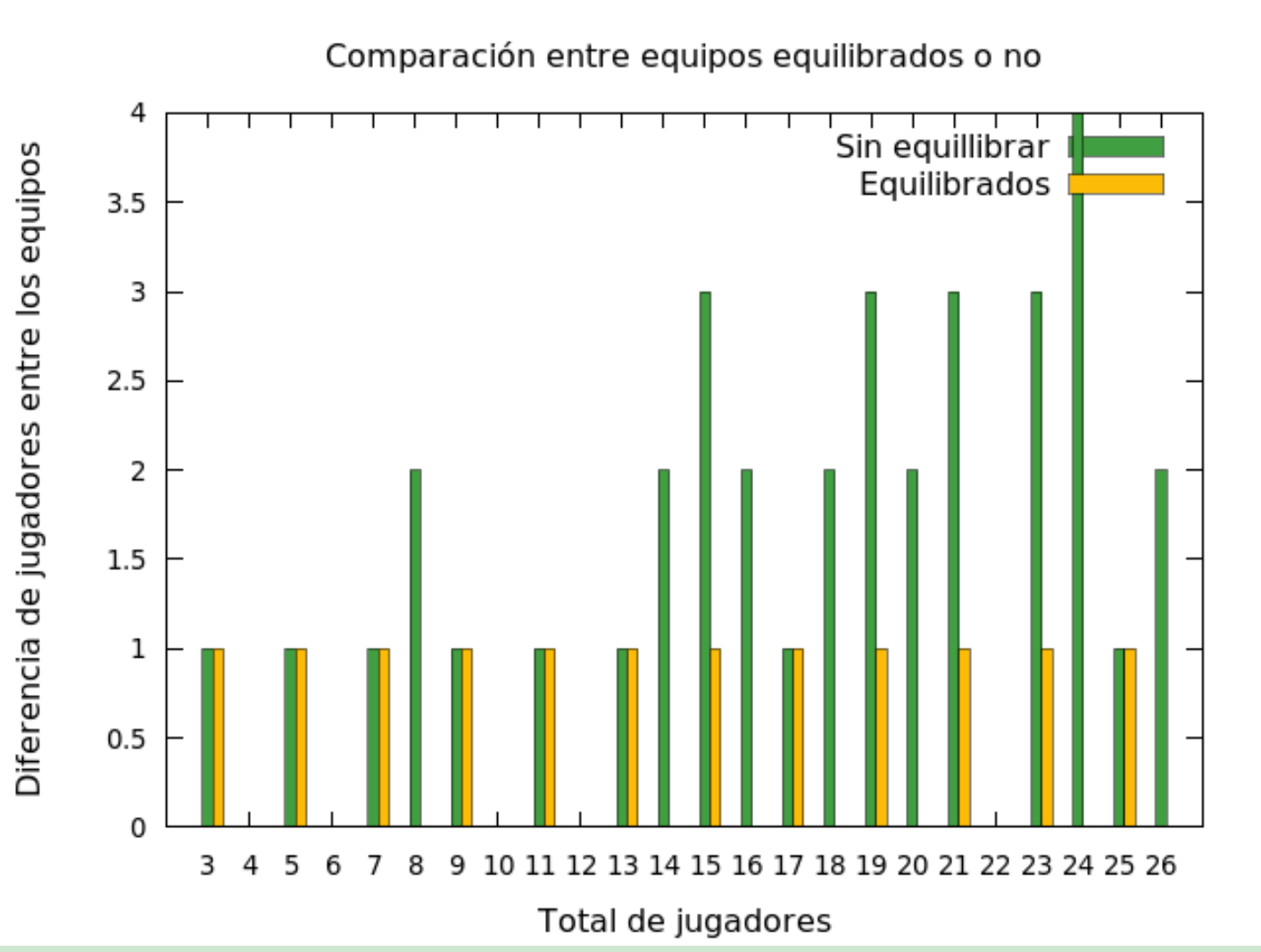
Tendencia

El primer algoritmo tiende a desajustar el número de jugadores, llegando a tener los equipos una diferencia de 4 jugadores cuando siendo 24 en total.

Diferencia de nivel entre los equipos



Número de jugadores de diferencia



Opciones

Para ejecuciones con un número pequeño de elementos los algoritmos planteados nos sirven, pero si aumentamos el número esa fuerza bruta no es tan buena idea.
Opciones:

- Poda el árbol: “Branch and Bound".
- x, y nivel de los equipos, conformarnos con $|x - y| < n \in \mathbb{N}$
- Paralelismo, cada procesador comprueba una rama

Conclusión

Como hemos comprobado, el segundo algoritmo iguala al primero en eficacia, pero lo supera en eficiencia. Esto nos hace pensar que puede no ser necesario siempre recorrer todo el árbol de soluciones para asegurarnos la optimalidad.

Opciones

Para ejecuciones con un número pequeño de elementos los algoritmos planteados nos sirven, pero si aumentamos el número esa fuerza bruta no es tan buena idea.
Opciones:

- Poda el árbol: “Branch and Bound".
- x, y nivel de los equipos, conformarnos con $|x - y| < n \in \mathbb{N}$
- Paralelismo, cada procesador comprueba una rama

Conclusión

Como hemos comprobado, el segundo algoritmo iguala al primero en eficacia, pero lo supera en eficiencia. Esto nos hace pensar que puede no ser necesario siempre recorrer todo el árbol de soluciones para asegurarnos la optimalidad.

viajante de comercio con vuelta atrás y ramificación y poda.

Cuando para un nivel no queden más ciudades válidas, el algoritmo hace una vuelta atrás proponiendo una nueva ciudad válida para el nivel anterior.

Para el algoritmo de ramificación y poda es necesario utilizar una cota inferior para cada nodo (solución parcial), un valor c menor o igual que el verdadero coste de la mejor solución (la de menor coste) que se puede obtener a partir de la solución parcial en la que nos encontremos.

Para realizar la poda, guardamos en todo momento en una variable C el costo de la mejor solución obtenida hasta ahora, si para una solución parcial, $c > C$ entonces se puede podar.

Como criterio para seleccionar el siguiente nodo vivo se empleará el criterio LC o “más prometedor", aquel que presente el menor valor de cota inferior.

Hemos utilizado dos algoritmos para encontrar una cota inferior.

Algoritmo propuesto

Para encontrar la longitud del grafo con los menores arcos sumamos el menor elemento de cada fila de la matriz de adyacencia.

Problema

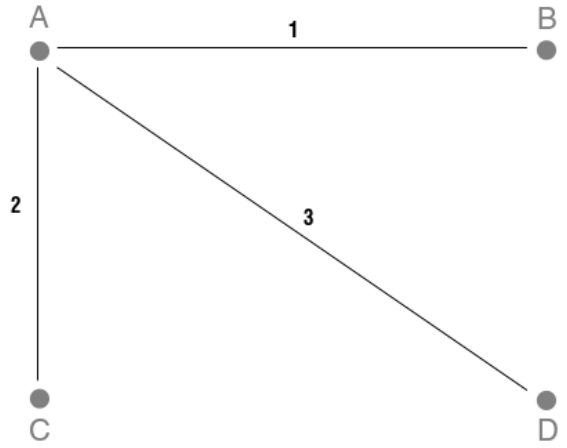
En matrices simétricas podemos repetir elementos simétricos y acabar con un grafo inconexo.

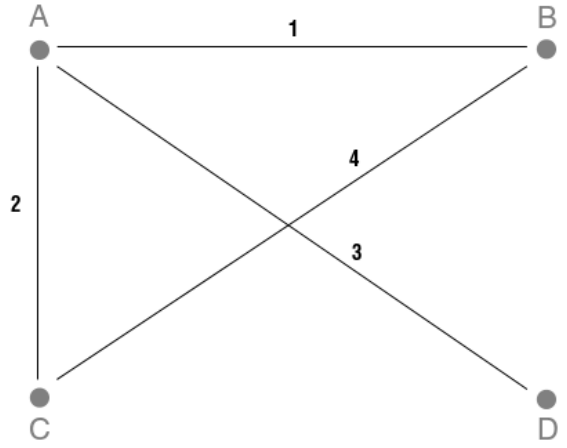
Algoritmo mejorado

Si hemos usado el simétrico de un elemento escogemos el siguiente menor.

Ejemplo

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 4 & 5 \\ 2 & 4 & 0 & 6 \\ 3 & 5 & 6 & 0 \end{bmatrix}$$





Eficiencia recorriendo el arbol

Tenemos que evaluar las permutaciones de un conjunto de $n-1$ elementos, por tanto la eficiencia será $O(n!)$.

Eficiencia de la primera cota

Recorremos los elementos de todas las filas. Por tanto $O(n^2)$.

Eficiencia de la segunda cota

Como sabemos en que fila se encuentra el simétrico de cualquier elemento y solo guardamos un elemento por fila, la comprobación es $O(1)$ y el algoritmo $O(n^2)$.

Comparativa

Funciones de cota

Las diferentes funciones para conseguir una cota inferior hacen que en un mismo mapa el algoritmo de ramificación y poda varíe.

Datos

- Nodos expandidos
- Mayor cola de nodos vivos
- Cantidad de cortes
- Tiempo

Cortes

Representatividad

Esta información puede no ser representativa de la magnitud de los cortes. Dados dos algoritmos X, Y que han hecho x_i, y_i cortes respectivamente con $x_1 > y_1$ no nos está diciendo que el primero sea más rapido. El primer algoritmo podría haber recorrido más nodos y hacer los cortes en niveles del árbol más profundos.

Imagen

Figura : Cortes efectuados

Tamaño de la lista de nodos vivos

A continuación exponemos el mayor tamaño alcanzado por la cola con prioridad. Esta información nos puede dar una idea de lo bueno que es nuestro algoritmo para el cálculo de una cota inferior.

Otras versiones

Hay versiones de ramificación y poda para el problema del viajante de comercio que no usan por ejemplo la aproximación greedy inicial, por lo que ha podido sobrecargar la cola con prioridad mientras busca la primera solución con la que podará después.

Imagen

Figura : Cola con prioridad más grande

Nodos

Utilidad

Este dato no nos asegura que tarde menos tiempo, ya que un algoritmo con más nodos expandidos que otro ha podido usar una función para calcular las cotas inferiores demasiado costosa.

Imagen

Figura : Nodos expandidos

Imagen

Figura : Tiempo algoritmo 1 y su ajuste

Imagen

Figura : Tiempo algoritmo 2 y su ajuste

Imagen

Figura : Comparativa de tiempos

A

unque los algoritmos de ramificación y poda nos aseguraran la optimalidad a problemas con tiempos de ejecución exponenciales ahorrándonos pasos, no resultan prácticos cuando el tamaño del problema crece. Los algoritmos greedy estudiados anteriormente nos pueden dar una aproximación más que razonable para el problema.