Metadata	No editar manualmente esta tabla.
Título	Optimización de los desarrollos de un cambio
Descripción	Selección de los platos y piñones de una bici que mejor aproximan un cierto conjunto de desarrollos por medio de un algoritmo genético.
Author	Miguel Angel Sanchez Quintanilla
Organisation	Universidad de Sevilla
Date	Diciembre 2015

Diseño del cambio de una bici.

Profesor responsable: Miguel Ángel Sánchez Quintanilla.

Nota: este problema está inspirado en el problema Best Bike, que aparece en el libro: Insight trough computing, de Charles F. Van Loan y K.-Y. Daisy Fan, ISBN 978-0-898716-91-7.

Planteamiento del problema

Supongamos que nos hemos comprado una bicicleta de montaña y que nos dejan escoger los platos y los piñones del cambio a nuestro gusto. Podemos escoger tres platos con cualquier número de dientes entre 32 y 52 y siete piñones con cualquier número de dientes entre 12 y 42. El número de desarrollos del cambio es de $3 \times 7 = 21$. El valor de cada desarrollo se encuentra de dividir el número N de dientes del plato entre el número M de dientes del piñón, de modo que si, por ejemplo, para una combinación dada de plato y piñón D= N/M=3 , el desarrollo D de ese cambio es tal que la rueda trasera gira tres vueltas completas cada vez que los pedales giran una vuelta completa.

Nos gustaría que los 21 desarrollos posibles estén escalonados a intervalos regulares entre D=1 (una vuelta de la rueda por cada vuelta de los pedales) hasta D=4 (cuatro vueltas de la rueda por cada vuelta de los pedales, es decir, queremos un conjunto de 21 desarrollos dado por:

$$D_k{}^{opt} = 1 + 0.15 \times (k-1) \quad k = 1,2,\ldots,21$$

Los valores numéricos que se obtienen de esta fórmula son

k	1	2	3	4	5	6	7	8	9	10	11
$D_k{}^{opt}$	1.00	1.15	1.30	1.45	1.60	1.75	1.90	2.05	2.20	2.35	2.50
k	12	13	14	15	16	17	18	19	20	21	
$D_k{}^{opt}$	2.65	2.80	2.95	3.10	3.25	3.40	3.55	3.70	3.85	4.00	

Sin embargo, tal combinación es imposible de lograr: a lo más que podemos aspirar es a acercarnos lo más posible a ella, por eso hemos llamado a esos valores $D_k{}^{opt}$, donde la "opt" indica *óptimo*. Llamemos simplemente D_k a los valores que podemos obtener con tres platos de número de dientes N_1, N_2 y N_3 y siete piñones de número de dientes M_1, M_2, M_3, M_4, M_5, M_6 y M_7, donde se supone que hemos ordenado los platos y los piñones de menor a mayor número de dientes.

Mediremos la proximidad de un conjunto de platos y piñones al óptimo mediante la norma:

$$Normr = \sqrt{\sum_{k=1}^{21} \left(D_k - {D_k}^{opt}\right)^2}$$

Donde hemos supuesto que los desarrollos D_k están ordenados de menor a mayor.

Restricciones.

Primera restricción. Puesto que hemos dicho que ordenamos nuestra elección de platos y piñones de menor a mayor y obviamente no montaremos más de un plato o piñón de cada número de dientes, sea cual sea nuestra elección tendremos que escoger los platos N_1, N_2 y N_3 de modo que:

$$N_1 < N_2 < N_3$$

Y los piñones M_1 a M_7 de modo que:

$$M_1 < M_2 < M_3 < M_4 < M_5 < M_6 < M_7$$

Segunda restricción. Puesto que queremos tener disponible el desarrollo D=1, tiene que cumplirse que:

$$M_7 = N_1$$

Tercera restricción. Puesto que queremos que el máximo desarrollo de nuestra bici sea D=4, tiene que cumplirse la restricción:

$M_1=round(N_3/4)$

Donde *round(x)* es la función de Matlab que calcula el número entero más próximo al número real *x*. Como el piñón más pequeños del que disponemos tiene 12 dientes, esta restricción implica que el plato N_3 no puede tener menos de 48 dientes.

Cuarta restricción. Los números de dientes de platos y piñones deben ser números enteros.

Con todas estas restricciones ninguno de los algoritmos que se han visto en clase sirve para resolver este problema. Aquí compararemos dos posibles estrategias.

Primera estrategia: resolver el problema por fuerza bruta.

Podemos resolver el problema inspeccionando todas las posibles configuraciones de platos y piñones mediante un conjunto de bucles anidados. Para ello:

- La variable entera N_1 debe correr entre 32 y 52-2=50.
- La variable entera N_2 debe correr entre N_1+1 y 52-1=51.
- La variable entera N_3 debe correr entre max([N_2+1,48]) y 52.

Dentro del bucle de N_3 queda definida una elección de platos (N_1,N_2,N_3). Con esa elección podemos calcular M_1 y M_7. Si M_7-M_1<6, no existe la posibilidad de escoger cinco piñones de dientes entre M_1+1 y M_7-1, así que debemos desechar esa elección de platos. Además, si M1<12 no tenemos disponible un piñón de ese tamaño, En caso contrario, podemos escoger todos los piñones posibles entre M_1 y M_7 haciendo que:

- La variable entera M_2 corra entre M_1+1 y M_7-5.
- La variable entera M_3 corra entre M_2+1 y M_7-4.
- La variable entera M_4 corra entre M_3+1 y M_7-3.
- La variable entera M_5 corra entre M_4+1 y M_7-2.
- La variable entera M_6 corra entre M_5+1 y M_7-1.

Dentro del último bucle anidado tenemos una elección de piñones (N_1,..,N_7) que nos permite calcular todos los desarrollos D_k del cambio y la norma de su ajuste al cambio óptimo. No te olvides de ordenar los valores de D_k de menor a mayor antes de calcular la norma. El comando de Matlab:

vord=sort(v);

ordena un vector de menor a mayor.

Te aconsejamos almacenar la primera elección en una estructura con estos campos:

- *bici.platos* Aquí puedes guardar la elección de platos en un vector.
- *bici.pinones* Aquí puedes guardar la elección de piñones en un vector
- *bici.desarrollos* Aquí puedes guardar los desarrollos de la elección del conjunto de platos y piñones escogidos.
- *Bici.normr* Aquí puedes guardar el valor de la norma que mide la proximidad al conjunto de desarrollos óptimo.

Cada nueva elección de platos y piñones la puedes comparar con la que tengas almacenada. Si la nueva elección tiene un valor de *normr* menor que la que tienes almacenada en esa estructura, sustituye la nueva elección sobrescribiendo la antigua.

Cuidado cuando pruebes el programa: probablemente tardará mucho tiempo en terminar.

Segunda estrategia: mejorar una solución al azar.

Cuando hayas programado la estrategia anterior verás que es muy ineficiente. Otra forma de atacar el problema es empezar con una elección de platos y piñones al azar y mejorarla poco a poco cambiando al azar platos y piñones, aceptando los cambios si hace que los desarrollos que tiene la bicicleta se aproximen al conjunto que consideramos óptimo. Para ello haremos uso de la función de Matlab:

randi([imin,imax],n);

que crea un vector con *n* números enteros escogidos al azar entre *imin* e *imax*, ambos inclusive.

Procedimiento para hacer una elección inicial al azar.

Para crear una elección de platos y piñones al azar sigue el siguiente procedimiento:

- Escoge un plato N_1 entre 32 y 52-2=50.
- Escoge un plato N_2 entre N_1+1 y 52-1=51.
- Escoge un plato N_3 entre max([N_2+1,48]) y 52.

Con esto queda definida una elección de platos (N_1,N_2,N_3). Con esa elección podemos calcular M_1 y M_7. Si M_7-M_1<6, no existe la posibilidad de escoger cinco piñones de dientes entre M_1+1 y M_7-1, así que debemos desechar esa elección de platos. Si M_1<12, no disponemos de un piñón de ese tamaño, y también debemos desechar la elección.

Una vez tengamos una elección de platos adecuada, podemos escoger cinco piñones entre M_1 y M_7:

- Escogiendo un piñón M_2 entre M_1+1 y M_7-5.
- Escogiendo un piñón M_3 entre M_2+1 y M_7-4.
- Escogiendo un piñón M_4 entre M_3+1 y M_7-3.
- Escogiendo un piñón M_5 entre M_4+1 y M_7-2.
- Escogiendo un piñón M_6 entre M_5+1 y M_7-1.

Se recomienda guardar las elecciones en una estructura del mismo tipo que el que se comenta en el apartado anterior. Se puede hacer este procedimiento de elección al azar desde cero un número moderado de veces (p. ej, 5 veces) y escoger la elección que arroje un menor valor de *Normr*.

Procedimiento para cambiar una elección.

Una vez hecha una elección de partida, podemos mejorarla mediante cambios al azar. Hay ocho posibles cambios: cambiar uno de los tres platos (3 opciones) o cambiar uno de los 5 piñones que van desde el M_2 hasta el M_6. Los piñones M_1 y M_7 vienen determinados por la elección de platos y no pueden cambiarse al azar. Para que todas las opciones sean equiprobables, selecciona un número *p* al azar entre 1 y 8.

- Si *p*=1, quita el plato más pequeño y sustitúyelo por otro escogido al azar entre 32 y 52 dientes.
- Si *p*=2, quita el plato mediano y sustitúyelo por otro escogido al azar entre 32 y 52 dientes.
- Si *p*=3, quita el plato más grande y sustitúyelo por otro escogido al azar entre 32 y 52 dientes.

Tras cualquiera de estas posibles elecciones, tendrás que reordenar los platos y volver a calcular los piñones M_1 y M_7. Comprueba si el resto de piñones está comprendido entre los

nuevos piñones M_1 y M_7. Si es así, acepta los cambios y si no es así, vuelve a la elección de platos anterior a la elección del número aleatorio *p*. Rechaza también los cambios si al elegir un plato al azar se escoge un plato con un número de dientes que ya está montado.

Para el resto de posibles valores del número aleatorio *p*, escoge un piñón con un número de dientes al azar entre M_1+1 y M_7-1 y:

- Si *p*=4, sustitúyelo por el piñón M_2.
- Si *p*=5, sustitúyelo por el piñón M_3.
- Si *p*=6, sustitúyelo por el piñón M_4.
- Si *p*=7, sustitúyelo por el piñón M_5.
- Si *p*=8, sustitúyelo por el piñón M_6.

Rechaza el cambio si al escoger un piñón al azar entre M_1+1 y M_7-1 resulta un piñón con un número de dientes igual a uno que ya está montado.

Cuando pruebes este procedimiento, te darás cuenta de que al elegir un nuevo plato al azar, es más fácil pasar de un conjunto de platos con números de dientes parecidos a otro conjunto de platos con números de dientes muy diferentes que el proceso inverso. Esto significa que la elección que hacemos al azar está sesgada, pero permitimos el sesgo porque la intuición nos dice que ese es el sentido en el que tiene que cambiar una elección de partida para mejorar.

Procedimiento para mejorar la elección inicial.

Tras cada cambio a la elección inicial, calcula la proximidad del conjunto de desarrollos que nos da la elección de platos y piñones que hemos hecho al conjunto de desarrollos que consideramos óptimo, es decir, calcula el valor de *Normr*. Si el nuevo conjunto de platos y piñones mejorar el valor de *Normr*, acéptalo como nuevo punto de partida en el que hacer un cambio al azar. Si el nuevo conjunto de platos y piñones no mejora el valor de *Normr*, deshaz el cambio y vuelve al conjunto de platos y piñones que tuvieras antes del último cambio al azar.

Te recomendamos hacer unos 100 intentos de mejora al azar y guardar el valor de *Normr* que tiene el conjunto de platos y piñones que mantienes como elección después de cada intento. Puede ser instructivo representar la evolución de *Normr* con el número de intentos.

Cómo medir el tiempo de computación de un programa.

Para medir el tiempo que tarda en ejecutarse un programa se usan los comandos:

tic;

…

Líneas de código

…

tiempo=toc;

La variable tiempo almacena el tiempo en segundos que tardan en ejecutarse las líneas de código entre *tic* y *toc*.

Puedes usar estos comandos para medir el tiempo que tardan en ejecutarse los códigos de ambos procedimientos.