Universidad Nacional Autónoma de México

Facultad de Ciencias

Tarea 2 - Reporte

Computo Evólutivo

Autores:
Diego García V.
Julio Vázquez A.

16 de marzo de 2023

Problema de la Mochila 0-1

Dada una colección de n objetos y una mochila, tal que:

$$p_j = \text{beneficio (ganancia) del objeto } j$$
$$w_j = \text{peso del objeto } j$$
$$c = \text{capacidad de la mochila}$$

selecciona un subconjunto de los elementos tal que se maximice:

$$z = \sum_{j=1}^{n} p_j x_j$$

sujeto a:

$$\sum_{j=1}^{n} w_j x_j \leq c$$

donde:

$$x_j = \begin{cases} 1, & \text{si el elemento } j \text{ es seleccionado} \\ 0, & \text{e.o.c} \end{cases}$$

Descripción de la implementación

Tenemos 2 implementaciones:

- Recocido simulado
- Aleatorio con vecindades.

En el recocido simulado es un algoritmo de optimización que previene el estacamiento dentro de un óptimo local. La menera de poder prevenirlo es mediante un mecanismo que sino encontramos un estado que mejore la función de evaluación actual aún así damos una oportunidad de tomar otro estado en función de qué la diferencia de su función de evaluación y la temperatura.

A mayor temperatura, mayor la probabilidad de tomar el estado, a menor temperatura, la búsqueda es más estable, al contrario que si tuviesemos una mayor temperatura tendríamos una búsqueda más agresiva.

En la implementación de vecindades partimos de un estado actual, una vecindad contendrá a todos los elementos que resulten de alterar un bit al estado actual.

Y los estados aleatorios se generan a partir de una vecindad y generaremos un vector aleatorio, generamos un valor enterado aleatoriamente mayor o igual a cero y menor o igual al número de bits total de la representación binaria del estado actual.

Este número entero corresponderá al número de bits a alterar de un estado actual, iterando hasta a haber alterano n bits, donde n es el número del estado actual.

Descripción del esquema de representación de soluciones

Se ha elegido un sistema de representación binaria de valores discretos, cada estado consta de un vector de valores binarios, cada bit representa si un objeto es almacenado en la mochila (1) o no (0). Se ha utilizado esta representación ya que un arreglo de este tipo contiene información suficiente de los estados, sin embargo, conforme aumente el tamaño del conjunto, aumenta la dimension del vector, pudiendo ser pesado en terminos de memoria, sin embargo es el precio a pagar por un procesamiento mas rapido y natural.

El espacio de busqueda es el conjunto $\{(x_1, x_2, ..., x_n)|x_i \text{ pertenece a } \{1, 0\}\}$. Es sencillo notar que el tamaño del espacio es 2^n y un estado será solución siempre que su coste no rebase cierto limite. Para saber si un estado es o no solución es necesario generarlo y evaluarlo. También es importante notar que la representación logra entregar un vector a cualquier estado del espacio de buqueda.

Ventajas

La ventaja obviamente es que vamos a sólo obtener los datos que nos interesan, en este caso, los elementos a cargar. Implicitamente sabemos los que no estamos trayendo, pero los descartamos debido a su innecesaria información al no ser una combinación óptima.

Consideramos esto una manera natural de definir lo que estamos cargando y lo que no.

Desventajas

La desventaja de esto es que nos tomó más tiempo de implementación retrasando la entrega, pero dentro del esquema una gran desventaja es que siempre será necesario cargar con la información de los pesos y beneficio adicional de cada estado para poder evaluarla.

Si el tamaño del objeto es demasiado grande la representación binaria crece de la misma forma o peor. Por lo que la vecindad a explorar se vuelve inmanegable, debido a que crece de acuerdo al número de bits.

Descripción de la función de evaluación

Nuestra función de evaluación es la siguiente:

$$z = \sum_{j=1}^{n} p_j x_j$$

Por lo que estamos buscando un subconjunto de elementos talq ue podamos maximizar dicha expresión, pero tenemos un limitante en nuestra expresión, el cuál sera que la suma de los elementos que vayamos a tener en nuestra mochila nos den el maximo beneficio sin que sobrepasen su capacidad.

Podemos verlo con el siguiente ejemplo:

Imaginemos que iremos de excursión a un bosque, por lo que necesitaremos herramientas de supervivencia para sobrevivir. Un experto nos da una lista de elementos a comprar, supongamos los siguientes elementos:

- *Linterna*
- *Bolsa para dormir*
- *Cerillos*
- *Botiquin de primeros auxilios*

Lo ideal es que pudieramos comprar todo lo de la lista, pero debido a que tenemos una restricción digamos que cada uno de los objetos a comprar tiene un determinado precio que nunca cambiará y a nosotros se nos adjudico una cierta cantidad de dinero a gastar, quedando de la siguiente manera la lista:
Dinero máximo a gastar: $100

- *Linterna* - $20.00
- *Bolsa para dormir* - $50.00
- *Cerillos* - $10.00
- *Botiquin de primeros auxilios* - $90.00

Ahora, esta restricción es importante, ya que no podemos comprar todo lo que se nos propone, pero algo más que tenemos que tener en cuenta es que debemos de obtener un beneficio dentro de nuestra compra, en otras palabras, tenemos que saber si lo que compramos tendrá mejor o menor beneficio de acuerdo a dicha compra, por lo que tendremos que anexarlo para poder hacer una compra mucho más inteligente:
Dinero máximo a gastar: $100
Anexando un beneficio del 1-10.

- *Linterna* - $20.00 - *7 de beneficio*
- *Bolsa para dormir* - $50.00 - *8 de beneficio*
- *Cerillos* - $10.00 - *9 de beneficio*
- *Botiquin de primeros auxilios* - $90.00 - *10 de beneficio*

Por lo que nuestro ideal sería la suma de toda lista, dando un beneficio de 34*, pero dado que no podemos comprar todo tendremos que maximizar nuestro beneficio con los* $100 *que tenemos, por lo que el beneficio máximo de esta compra sería:* 24*. Ya que elegiríamos los siguientes productos:*

- *Linterna* - $20.00 - *7 de beneficio*
- *Bolsa para dormir* - $50.00 - *8 de beneficio*
- *Cerillos* - $10.00 - *9 de beneficio*

Gasto: $80.00*, Beneficio:* 24

Función de evaluación de soluciones

Nuestra función de evaluación de soluciones será la restricción de que no debemos exceder el peso de la mochila; si una solución (aleatoria) excediera el peso tendríamos que deshecharla.

Por lo que tendremos que estar atados a lo siguiente:

$$\sum_{j=1}^{n} w_j x_j \leq c$$

Usando lo anterior, las soluciones que tomen al *Botiquin de primeros auxilios* como objeto a comprar serán descartadas aquellas que agreguen:

- *La linterna, debido a que sobrepasan la cantidad de dinero*
- *La bolsa para dormir, por la razón anterior.*
- *Los tres objetos: Bolsa para dormir, linterna y cerillos*

Por lo que la única combinación que podríamos aceptar al comprar el *botiquin de primeros auxilios* sería:

- *Cerillos; debido a que la suma da exactamente* $100.00

Algo que tenemos que tener en cuenta es que las propuestas de soluciones no siempre serán las que maximicen el beneficio de nuestra mochila, debido a que en este ejemplo al sólo elegir el botiquin de primeros auxilios y los cerillos estamos obteniendo un beneficio de 19*, por lo que no es el beneficio máximo que podremos obtener que fue de* 24.

Para la implementacion de dicha funcion haremos uso de la interface previamente desarrollada `EvalFunction` y la clase Abstracta `EvalUtils`, que proporcionan el mecanismo genérico de una función de evalación, sin embargo, para el caso particular de esta función de evaluación (que llamaremos `DiscreteWeightsFunction`) se añadira la propiedad de los pesos, de esta manera, la funcion recibe como parámetro un vector del espacio de busqueda como única entrada, facilitando la implementación general y desacoplando la representacion de los estados de cualquier dependencia del problema particular como lo son los pesos.

La función previamente mencionada tambien servirá para calcular la utilidad de un estado, ya que la operacion es la misma, simplemente se cambian los pesos del coste por los pesos de la ganancia.

$g = \sum_{i=1}^{n} (g_i * x_i)$

Descripción del generador de soluciones aleatorias

Como tenemos una mochila con una capacidad máxima para cargar cierta cantidad de objetos, y cada objeto tendrá un peso y un beneficio asociado. Nuestro objetivo será determinar qué objetos debes incluir en la mochila para

maximizar el beneficio total de los objetos, sin exceder la capacidad máxima de la mochila.

Para encontrar soluciones aleatorias al problema, primero elegiremos un número de objetos que quepan en la mochila, ya que tendremos un valor máximo dado al inicio del archivo. Seleccionaremos a lo largo del archivo un subconjunto de los objetos descritos. La selección debe tener en cuenta que no podemos exceder la capacidad máxima de la mochila. Pero podemos ir seleccionando objetos y sumando sus pesos hasta alcanzar la capacidad máxima. Si nos pasamos de la capacidad máxima, debemos eliminar uno o más objetos para ajustarnos a la capacidad máxima.

Una vez que hayamos seleccionado un subconjunto de elementos que caben en la mochila, debemos calcular el beneficio total. Esto lo haremos sumando los valores de los objetos seleccionados.

Repetiremos la selección de objetos y calcularemos el beneficio varias veces, para obtener distintas soluciones. A continuación, seleccionaremos la solución con el valor total más alto como solución final.

Representación de soluciones

La representación utilizada en nuestra tarea fue un vector de n localidades, donde n será el número de los elementos que tenemos.

Tendrá la siguiente forma:

$$[x_1, x_2, x_3, \dots x_n]$$

Lo cuál representará únicamente los elementos que llevaremos en nuestra mochila.

Decidimos hacerlo de esta forma debido a que en la solución lo único importante es saber qué elementos llevaremos, no nos importarán los elementos que no podamos cargar, porque justamente no los cargaremos, los elementos que no maximicen nuestra mochila en beneficio tampoco nos interesarán, debido a que no será una mochila óptima para cargar.

Descripción de la función u operador de vecindad

Para este ejercicio tomaremos una vencidad al rededor del estado S y radio r como el conjunto de estados que se obtienen de alterar a lo mas r bits de S. Es una manera muy natural de definir vecinos, ya que alterar un bit a un estado es equivalente a decir que se ha agregado o eliminado de la lsta. Así, un estado será cercano a otro mientas mas bits tengan en común.

Recocido simulado

El algoritmo de recocido simulado es un algoritmo que busca solucionar el problema de caer en optimos locales, permitiendo a ala busqueda admitir estados que no necesariamente mejoran la evauación, sin embargo nos pueden conducir a otra solucion que si mejore la evaluación, pudiendo ser incluso un optimo

global. Esto puede ocurrir mientas el sistema de encuentre lo suficieniemente *çaliente"*. Una vez el sistma esta frio, unicamente admitira mejoras si el nuevo estado mejora la evaluación. Este modelo se basa en el recocido del acero que permite a las moleculas encontrar mejores zonas de cristalizacion aumentando la calidad del acero gracias a elevar la temperatura.

Esta implementación de recocido simulado consta de 4 componenes elementales:

- La representación binaria de los estados
- Una funcion de evaluación
- Un esquema de enfriamiento
- Una variable aleatoria con una funcion de densidad dada que determinará si se toma o no un estado que no mejore la evaluación

La representación binaria a utilizar será el vector de valores discretos mencionado con anterioridad que funciona a la perfección con la función de evaluación `DiscreteWeightsFunction`, misma que será utilzada en el algoritmo de recocido.

El esquema de enfriamiento consta de un factor alfa que será multiplicado por la temperatura actual en cada iteración, el comportamiento de descenso de temperatura será exponencial de la siguiente manera:

$$T_i = T_{ini} \cdot \alpha$$

Las temperaturas finales e iniciales, asi como el factor α determinan el comportamiento del descenso de la temperatura, este a su vez, determina el comportamiento de salto entre estados que no mejoran la evaluación, en otras palabras, en función de como se elijan las temperaturas y el factor alfa, será como el algoritmo logre escapar y permanecer en optimos locales o un optimo global.

Finalmente, tenemos la variable aleatoria que determinará la probabildad de tomar o no un cambio de estado a uno que empeora la evaluación. Para ello se hara uso del modulo desarrollado que genera variables aleatorias dada una función de densidad, para este punto es importante mencionar el mecanismo de generación de variables aleatorias dada una funcioón de densidad, ya que en Java solo tenemos la generación de variables aleatorias uniformemente distribuidas dentro del intervalo (0,1).

Supongamos que tenemos una v.a. U uniformemente distribuida en el intervalo $(0, 1)$, es decir $U \backsim U(0, 1)$, supongamos que queremos obtener una variable $X \backsim N(0, 1)$, una manera intuitiva de aproximarnos a una solución es tomar a U, y partir el intervalo $(0, 1)$ en pequeños pedazos de diferentes tamaños y cada pedazo tenga asignado un valor, claramente, la probabilidad de que U caiga en determinado pedazo dependerá del tamaño del propio pedazo, asi que podriamos partir el intervalo en partes de tamaño tal que la probabildad de que caiga en un pedazo con el valor asignado 0 sea mayor que uno que tenga un valor de 1 o 2, similar a la densidad de una distribución normal estandar. Sin embargo una

variable $X \backsim N(0,1)$ es continua y no podemos discretizar sus valores, siguiendo el razonamiento anterior, la intuicion nos dice que deberiamos tomar cada vez mas pedazos dentro del intervalo tal que conserven la distribucion que se asemeja a uan normal, y si llevamos esto al limite, notaremos un comportamiento bastante interesante. El valor limite es la función inversa de la Distribución de la normal estandar, es decir, si $X \backsim N(0,1)$ donde $f(x)$ es la función de densidad y $F(x)$ la función de distribución (Recordemos que $F(x)' = f(x)$), entonces $F^{-1}(U) \backsim N(0,1)$, veamos una demostración.

PD: $X = F^{-1}(U)$ $N(0,1)$

$F_X(x)$

$= P(X < x)$

$= P(F^{-1}(U) < x)$

$= P(U < F(x))$

$= \int_0^{F(x)} dt$

$= F(x) - 0 = F(x)$

$\Rightarrow F_X(x) = F(x)$

Donde F(x) es la distribución de una normal estandar, por lo tanto $X \backsim N(0,1)$

QED

Con este mecanismo, dada f y su antiderivada F, podemos obtener una variable aleatoria con función de densidad f partiendo unicamente de una variable aleatoria Uniformemente distribuida en el intervalo (0,1)

Con la ultima pieza sobre la mesa, debemos encontrar el mecanismo que permite elegir un estado que no mejora la evaluación con la siguiente probabilidad

$$p = exp(-(ganancia(S) - ganancia(S'))/T)$$

Definimos la siguiente v.a. X funcion de densidad $f(x) = T * exp(-x/T)$, su función de distribución es $F(x) = 1 - exp(-x/T)$ Partiendo de la función de distribución anterior, podemos asegurar que:

$exp(-(ganancia(S) - ganancia(S'))/T) = 1 - F(ganancia(S) - ganancia(S')) = P(X > ganancia(S) - ganancia(S')) \Rightarrow p = P(X > ganancia(S) - ganancia(S'))$ Finalmente, obtenemos un valor aleatorio X, si este es mayor que $ganancia(S) - ganancia(S')$ entonces realizamos el cambio al nuevo estado, en otro caso, el nuevo estado es ignorado. Cabe mencionar que $ganancia(S) - ganancia(S') > 0 \Rightarrow ganancia(S) > ganancia(S')$, esto es de esperarse, pues S' no mejora la evaluación, por lo tanto su ganancia debe ser menor.

Descripción del esquema de enfriamiento

El sistema se irá enfriando cada vez que mejora un estado.

En cada iteración de enfrimiamiento múltiplica la temperatura por un factor de 0.95, el valor se obtuvo de manera experimental a modo que no terminará el proceso muy rápido, encontrando un equilibrio entre suficiente tiempo de enfriamiento y suficiente estabilidad.

El resultado de esto es un comportamiento exponencial.

Resultados obtenidos

Ejemplar	Tamaño del ejemplar	Mejor valor obtenido	Valor promedio	Peor valor obtenido	Número de iteraciones promedio
eje1n1000.txt	1000	9,989,472,185	9,989,471,947	9,989,471,716	2,522
eje2n1000.txt	1000	9,980,472,422	9,980,472,114	9,980,471,415	2,522
ejeknapPI_1_50_1000_14.txt	50	9,666,035	9,666,035	9,666,035	2,521
ejeknapPI_3_200_1000_14.txt	200	16,444	16228	16,047	2,521
ejeknapPI_11_20_1000_100.txt	20	8,242	8213	8,185	2,522
ejeknapPI_13_100_1000_18.txt	100	17,682	17,682	17,682	2,520
ejeL1n10.txt	10	293	288	283	1,000
ejeL10n20.txt	20	1,025	942	859	2,521
ejeL14n45.txt	45	1,944	1,939	1,934	2,522

Tabla 1: Resultados obtenidos en las experimentaciones

Conclusiones

En conclusión, el algoritmo de recocido simulado es una técnica heurística eficaz y ampliamente utilizada para resolver el problema de la mochila y otros problemas de optimización combinatoria. Esta técnica permite explorar de manera eficiente el espacio de soluciones y encontrar soluciones cercanas al óptimo global, incluso en casos en los que los algoritmos exactos no pueden proporcionar una solución óptima en un tiempo razonable.

Sin embargo, el éxito del recocido simulado depende en gran medida de la elección adecuada de los parámetros, como la temperatura inicial, la función de enfriamiento y la estrategia de generación de soluciones vecinas. Además, para problemas de tamaño considerable, puede ser necesario utilizar técnicas de paralelización y optimización para mejorar el rendimiento del algoritmo.

En cuanto al problema de la mochila, el recocido simulado puede proporcionar soluciones cercanas al óptimo global en un tiempo razonable para problemas con un gran número de objetos y restricciones. Por lo tanto, se puede concluir que el recocido simulado es una técnica prometedora para resolver el problema de la mochila y otros problemas de optimización combinatoria en la práctica.

Referencias

1. Korte, B., & Vygen, J. (2006). Combinatorial Optimization: Theory and Algorithms. Springer London, Limited.

2. P. J. M. van Laarhoven. (1987). Simulated annealing: Theory and applications. D. Reidel.