Proyecto Ladrón

Lic. Arnoldo Del Toro Peña

15 de junio de 2022

Resumen

Uso de un método metaheurístico destructivo - constructivo para el problema del ladrón viajero (travelling thief problem) (Bonyadi *et al.*, 2013).

Palabras clave: python,viajero, mochila, metaheurística, ladrón.

1. Introducción

1.1. Antecedentes

Bonyadi *et al.* (2013) presentan el problema del ladrón viajero con los siguientes parámetros: TSP sub-problema:

- Número de ciudades: n
- d_ij es la distancia de la ciudad i a la ciudad j
- Velocidad v_c
- Solución: vector del tour $x = (x_1, \cdots, x_n)$ donde cada x_i es una ciudad.

KP sub-problema:

- Número de items: m
- Peso de cada item: w_k
- Valor de cada item: p_k
- Total de la capacidad de la mochila: W
- Solución: un vector binario llamado items obtenidos $(y = y_1, \cdots, y_m)$ donde cada y_k es 1 si es obtenido y 0 si no lo es.

TTP:

- Disponibilidad de item I_i en cada ciudad: $A_i \subseteq \{1, \cdots, n\}$

- Solución: dos vectores, tour x y un plan de obtención de items z. Donde $z = (z_1, \cdots, z_m)$, $z_i \in \{0 \cup A_i\} \forall i$ donde cada ciudad tiene una disponibilidad para cada item, si el item i no es obtenido entonces $z_i = 0$. La solución es un tour para el ladrón y un plan de obtención items.

Se utilizan dos funciones objetivo, el TTP (por sus siglas en inglés) se trata de un problema en el cual tenemos un ladrón el cual tiene que recorrer n ciudades, cada ciudad tiene un número de objetos posibles de recoger y solo existen m objetos diferentes; se permiten objetos que puedan ser recogidos en distintas ciudades. Cada objeto tiene asociado los parámetros de peso y ganancia, el ladrón tiene una capacidad máxima de peso por lo cual no puede recoger todos los objetos.

El ladrón tiene que salir de la ciudad uno para después pasar a otra ciudad y así sucesivamente recorrer todas las ciudades. Durante todo el tiempo en que el ladrón hace el recorrido el tiene que pagar una renta (este valor afecta nuestra función objetivo), durante este mismo tiempo el ladrón lleva una velocidad de ciudad en ciudad que depende del peso que lleve cargando y se define como:

$$v_c = \left(v_{max} - W_c \frac{v_{max} - v_{min}}{W} \right) \tag{1}$$

En 1 v_{max} es la velocidad máxima con la que puede viajar el ladrón y v_{min} es la velocidad mínima con la que se desplaza. W_c es el peso que lleva en la mochila, si la mochila esta vacía entonces $W_c = 0$ y si esta llena $W_v = W$. Para nuestro algoritmo utilizaremos la primer función presentada de la forma:

$$G(xz) = g(z) - R * f(x, z) \tag{2}$$

En la ecuación 2 obtenemos la ganancia por el viaje y los objetos adquiridos, g es el total del valor de los objetos obtenidos, R es la renta por unidad de tiempo y f (se presenta en la ecuación 3) es el total del tiempo obtenido por el tour.

$$f(x) = \sum_{i=1}^{n-1} \left(t_{x_i, x_{i+1}} \right) + t_{x_n, x_1}, \quad x = (x_1, \cdots, x_n) \tag{3}$$

$$\text{donde: } t_{x_i, x_{i+1}} = \frac{d_{x_i, x_{i+1}}}{v_c}$$

Con los parámetros definidos se presenta el pseudo-código:

Algorithm 1 Iterated Greedy

```
1: procedure IteratedGreedy
2:     s_0 = GeneralInitialSolution
3:     repeat
4:         s_p = Destruction(s*)
5:         s' = Construction(s_p)
6:         s* = Acceptance Criterion(s*, s')
```

Algoritmo para la solución inicial.

Algorithm 2 General Inicial Solution

1: Algorithm local search for vector x.
2: Algorithm minimum weight and maximum distance search for vector z.

Algoritmo para la destrucción.

Algorithm 3 Destruction

1: Determinate s_1, s_2. ▷ s_1 and s_2 is a vector
2: **if** long $s_1 > m$ or long $s_2 > n$ **then**
3: **repeat** step 1.
4: Eliminate z_{s_1} from vector z.
5: Eliminate x_{s_2} from vector x.

Algoritmo para la contrucción.

Algorithm 4 Construction

1: $s_{p_1} = (z_1, z_2, \cdots, z_{m-s_1})$, $s_{p_2} = (x_1, x_, \cdots, x_{n-s_2})$
2: $s_1^{'}$ = Algoritmo en base al cociente de rendimiento $r_i = \dfrac{b_i}{v_i}$, $\forall s_1$
3: $s_2^{'}$ = Algoritmo del vecino más cercano $\forall p_2$.

Algoritmo para la aceptación.

Algorithm 5 Acceptance Criterion

1: Acceptance Criterion
2: **if** $G(s_2^{'}, s_1^{'}) > G(s_2^*, s_1^*)$ **then**
3: $s_2^* \longleftarrow s_2^{'}, s_1^* \longleftarrow s_1^{'}$

2. Descripción pseudocódigo

El pseudocódigo se divide en 5 módulos:

1. **Módulo general**: se genera una solución inicial y se repiten en orden los módulos de destrucción, construcción y el criterio de aceptación.

2. **Solución inicial**: mediante una búsqueda local y un LRC se determina el vector x tour, para construir el vector z se escoge el item con menor peso y mayor distancia.

3. **Destrucción**: se determinan de manera aleatoria dos vectores s_1, s_2 los cuales son eliminados de los vectores x y z respectivamente.

4. **Construcción**: de los elementos eliminados se determinan los items a obtener mediante un cociente $\frac{b_i}{v_i}$ y las ciudades mediante un algoritmo del vecino más cercano con un LRC.
5. **Criterio de aceptación**: mientras que $G' > G$ entonces aceptamos la siguiente solución.

3. Descripción del algoritmo constructivo

Como se mencionó anteriormente, la selección del nodo siguiente se baso en una lista generada por los nodos que estaban entre un cierto rango el cual esta determinado por la siguiente fórmula:

$$c_{min} \leq c_e \leq c_{min} + \alpha * (c_{max} - c_{min})$$

Donde c_e es el costo de los nodos disponibles para visitar, una vez generada la lista seleccionamos al azar un nodo de esta misma, el cual será el nodo próximo a visitar.

Esto lo repetimos hasta que todos los nodos esten visitados, agregando el nodo inicial para terminar el recorrido del agente viajero.

4. Descripción del algoritmo búsqueda local

El algoritmo se finaliza cuando todos los nodos estan marcados y tenemos que agregar el nodo inicial al final para poder cerrar el recorrido.Podemos enumerar sus pasos de la siguiente manera:

1. Elección de un vértice arbitrario respecto al vértice actual.
2. Descubra la arista de menor distancia (que cumpla con la ecuación mencionada anteriormente) que ya este conectada al vértice actual y a un vértice no visitado V.
3. Convierta el vértice actual en V.
4. Marque V como visitado.
5. Si todos los vértices del dominio estuvieran visitados, cierre el algoritmo.
6. Vaya al paso 2

El algoritmo de búsqueda local sigue los siguientes pasos:

1. Seleccionamos los nodos a intercambiar.
2. Si los nodos son adyacentes entonces: calculamos el delta de la siguiente manera

$$C_{act} = C_{i-1,i} + C_{i,j} + C_{j,j+1}$$

,

$$C_{nue} = C_{i-1,j} + C_{j,i} + C_{i,j+1}$$

$$\delta = C_{act} - C_{nue}$$

3. Si no, calculamos el delta de la siguiente manera:

$$C_{act} = C_{i-1,i} + C_{i,i+1} + C_{j-1,j} + C_{j,j+1}$$

,

$$C_{nue} = C_{i-1,j} + C_{j,i+1} + C_{j-1,i} + C_{i,j+1}$$

$$\delta = C_{act} - C_{nue}$$

4. Si $\delta \geq 0$ entonces intercambiamos los nodos.

5. Si no hemos hecho esto para todos los nodos regresamos al paso 1.

5. Implementación

La implementación se realizó en lenguaje de programación python se tomaron alfas distintos que se presentarán más adelante y se dio una tolerancia de 1000 mejoras.

6. Resultados

Los resultados se pueden ver en el mismo enlace de git-hub, en los documentos txt, sin embargo se presentarán a continuación en una forma más ordenada:

A continuación se muestran las tablas de los valores de la primera instancia:

alfa:	0.1	Gxz:	-103556.36
alfa:	0.01	Gxz:	-156626.45
alfa:	0.001	Gxz:	-116451.28
alfa:	0.0001	Gxz:	-102966.83
alfa:	1e-05	Gxz:	-125238.87
alfa:	1e-06	Gxz:	-46565.520
alfa:	0.1	Gxz:	-163343.18
alfa:	0.01	Gxz:	-74085.743
alfa:	0.001	Gxz:	-61343.588
alfa:	0.0001	Gxz:	-238723.52
alfa:	1e-05	Gxz:	-138268.66
alfa:	1e-06	Gxz:	-113330.60

alfa:	0.1	Gxz:	-80355.99
alfa:	0.01	Gxz:	-79737.92
alfa:	0.001	Gxz:	-42127.17
alfa:	0.0001	Gxz:	-64159.61
alfa:	1e-05	Gxz:	-32660.29
alfa:	1e-06	Gxz:	-99522.65

En la tabla anterior podemos observar que los valores son todos negativos lo cual queda muy lejos del valor buscado, sin embargo en la segunda tabla se muestran valores negativos pero de mejores valores.

Se utilizaron dos instancias, en las cuales los valores esperados son de: 16099 y 235856.

La siguiente tabla muestra los resultados para la segunda instancia:

alfa:	0.1	Gxz:	-2223757.30
alfa:	0.01	Gxz:	-2441913.85
alfa:	0.001	Gxz:	-2058655.57
alfa:	0.0001	Gxz:	-5825746.00
alfa:	1e-05	Gxz:	-4531215.67
alfa:	1e-06	Gxz:	-3944743.90

Como podemos ver todos los valores se presentan de manera dispersa, sin embargo ya se ven de una manera más uniforme, esto debido a que en el segundo cuadro se aumento la tolerancia de mejora.

Los resultados pueden ser observados en el archivo resultados.txt que se encuentra en el repositorio de git hub.

7. Mejoras

Mejorar la búsqueda local, tomar una decisión de cambiar el criterio de aceptación.

8. Conclusiones

Las soluciones obtenidas son muy carentes a lo que se desea, por lo cual no cabe duda que existe trabajo futuro para mejorar el código.

Referencias

Bonyadi, M. R., Michalewicz, Z., y Barone, L. (2013). The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. En *2013 IEEE Congress on Evolutionary Computation*, pp. 1037–1044. IEEE.