Proyecto final

Alumno : Velarde Moreno, Joaquin Arturo

Profesor: Alvarez Socarras, Ada Margarita

9 de mayo de 2022

Resumen: En este proyecto trabajamos con el problema de latencia mínima y proponemos estudiar un algoritmo multi-arranque para resolverlo. Para construir una solución se sugiere un constructivo con un heurístico de mejor inserción y añadimos una fase de mejora aleatoria, después realizamos un experimento con una instancia de 100 nodos y comparamos los resultados con el algoritmo aleatorio y sus componentes.

1. Introducción

El problema de latencia mínima es un problema que consiste en encontrar una permutación con el menor valor de latencia, este problema tiene su definición en teoría de redes, ya que una solución consiste en un camino Hamiltoniano el cual empieza del nodo raíz a cada vértice del nodo.

1.1. Historia

Este problema fue introducido formalmente por Fox, Kenneth R en 1973 [2] donde lo ilustro con ejemplos de la industria de cerveza. En este tipo de problemas la localización y tiempos de servicio de los clientes son conocidos y el objetivo es minimizar el tiempo total de espera de estos. A través del tiempo nuevas formulaciones han sido propuestas como por ejemplo Méndez días [3], también diferentes metaheuristicas han sido diseñadas tales como el VNS , Grasp combinado con VNS , Grasp combinado con VNS [4], en específico un algoritmo multiarranque [1].

1.2. Formulación del problema

El MLP está definido sobre un grafo completo $G = (V, A)$ donde $V = \{0, 1, 2, .., n\}$ es el conjunto de nodos y A es el conjunto de arcos.
El nodo 0 corresponde al nodo raíz, tenemos también una matriz $C = (c_{ij})$ de costos no negativos asociados a los arcos $(i, j) \in A$. Sea $P = \{0, [1], [2], ..., [n]\}$ de V, una permutación o camino Hamiltoniano con el nodo 0 en la primera posición y n es el número de nodos en P.
La latencia $l_{[i]}$ de un nodo en la i-esima posición de la permutación es calculada como:

$$l_{[i]} = \sum_{j=1}^{i} C_{[j-1][j]}. \tag{1}$$

La latencia total de una permutación P es la suma de las latencias de todos los nodos y es calculada como:

$$L = \sum_{i=1}^{n} l_{[i]} = \sum_{i=1}^{n} (n - i + 1) C_{[i-1][i]}. \tag{2}$$

2. Algoritmo multi-arranque para el MLP

Un método multi-arranque tiene como característica que se ejecuta varias veces partiendo de diferentes soluciones para encontrar soluciones más variadas suelen estar divididas por dos fases: 1) un método constructor y diversificador, 2) un método de búsqueda local. El algoritmo que ser diseñado como propuesta tiene ambas partes, pero con un ligero cambio, el cual es que los métodos de mejora son escogidos aleatoriamente en una distribución uniforme, con el objetivo de lograr más diversificación en la solución final.

Algoritmo 1: Algoritmo completo

```
1  Contador = 1;
2  Intentos  = 4;
3  while (Contador ≥ Intentos) do
4  |   r = random(0,1);
5  |   x = Constructivo();
6  |   if r == 1 then
7  |   |   x' = BusquedaLocal-Intercambio(x);
8  |   else
9  |   |   x' = BusquedaLocal-Posicion(x);
10 |   end
11 |   if f(x') mejor f(x) then
12 |   |   x = x';
13 |   |   Contador = 1;
14 |   end
15 |   Contador++;
16 end
```

2.1. Constructivo

Algoritmo 2: Constructivo(Alfa)

```
1  Solucion.add(Clientes[0]);
2  while (Solucion.size() < n) do
3  |   Candidatos = [];
4  |   for i = 0; i < n; i++ do
5  |   |   if Cliente[i] not in solucion then
6  |   |   |   k = n;
7  |   |   |   suma = 0;
8  |   |   |   q = 1;
9  |   |   |   m1 = (k+1)(C_[0][i]) +
   |   |   |    k(C_[i][1] − C_[0][1]) ;
10 |   |   |   for j = 2; j < k; j++ do
11 |   |   |   |   suma += C_[j−2][j−1];
12 |   |   |   |   delta = suma +
   |   |   |   |    (k − j + 2)(C_[j−1][i]) + (k − j +
   |   |   |   |    i)(C_[i][j]−C_[j−1][j]));
13 |   |   |   |   if m1 > delta then
14 |   |   |   |   |   m1 = delta;
15 |   |   |   |   |   q = j;
16 |   |   |   |   end
17 |   |   |   |   suma = C_[k−2][k−1];
18 |   |   |   |   delta = suma + C_[k−1][i];
19 |   |   |   |   if m1 > delta then
20 |   |   |   |   |   m1 = delta;
21 |   |   |   |   |   q = k;
22 |   |   |   |   end
23 |   |   |   |   Candidatos.add(i);
24 |   |   |   end
25 |   |   end
26 |   end
27 |   Insertar-Candidato(Candidatos, q, Alfa);
28 end
```

Un algoritmo ciego es aquel que siempre se va con el mejor candidato a en cada paso de la construcción de una solución, aunque puede ser lo racional al construirla, es posible que esto nos lleve un punto muerto, en el que no se exploren más soluciones, por esto se suele agregar una característica aleatoria al seleccionar en cada paso a cada candidato de la solución, este método utiliza una lista de candidatos restringida por medio de un valor α el cual ajusta a los candidatos a cierto rango, después por medio de la probabilidad se escoge al candidato que es añadido en el mejor punto de inserción de la solución por medio un proceso previo que da como resultado la mejor posición q.

2.2. Intercambio de nodos en solución

Una búsqueda local tiene como objetivo mejorar una solución dada por medio de elegir dentro de un vecindario una nueva solución, el cual fue previamente construido con una serie de movimientos o alteraciones que se hacen a la solución.
El movimiento más famoso para este tipo de soluciones es cambiar de posición un nodo en la posición $[i]$ con otro en una posición $[j]$ donde $i \neq j$.

Algoritmo 3: Intercambio(m,Solucion)

```
1  Delta = 0;
2  for i = 0; i < n; i++ do
3      for j = 1; j < n; j++ do
4          if i != j then
5              temp = Solucion[i];
6              Solucion[i] = Solucion[j];
7              Solucion[j] = temp;
8              Delta = (n-i+1)(C_[i-1][i+1] - C_[i-1][i]) + (n-i)(C_[i+1][i] - C_[i][i+1]) + (n-i-1)(C_[i][i+2] - C_[i+1][i+2]);
9              if Delta > 0 then
10                 return Solucion;
11             end
12             Solucion[j] = Solucion[i];
13             Solucion[i] = temp;
14         end
15     end
16 end
```

Algoritmo 4: Posicion(n,Solucion)

```
1  Delta = 0;
2  TempSolucion = Solucion;
3  for i = 0; i < n; i++ do
4      for j = 1; j < n; j++ do
5          if i < j then
6              Delta = 0;
7              x = i;
8              y = j;
9              while x < y) do
10                 temp = Solucion[x];
11                 Solucion[x] = Solucion[y+1];
12                 Solucion[x+1] = temp;
13                 Delta += (n-x+1)(C_[X-1][X+1] - C_[X-1][X]) + (n-x)(C_[X+1][X] - C_[x][x+1]) + (n-x-1)(C_[x][x+2] - C_[x+1][x+2]);
14                 x++;
15                 if Delta < 0 then
16                     return Solucion;
17                 end
18                 Solucion = TempSolucion;
19             end
20         end
21     end
22 end
```

2.3. Cambio de posición para un nodo

Otra manera de generar una nueva solución en la etapa de mejora es por medio de cambiar de posición a un nodo que se encuentra en la posición $[i]$ de la permutación y moverlo a una posición $[j]$ de esta. Para este método obtenemos un sub vecindario para un nodo i el cual es compuesto de todas las posiciones j en el cual puede ser insertado, este sub vecindario después es explorado por medio de cambios sucesivos a posiciones adyacentes mientras sumamos los valores de cambio. Al final tendremos un Δ con el que podremos evaluar si escogemos la solución.

3. Experimentación

Las instancias con las que se trabajó son de $n = 100$ clientes dadas por la profesora Ada Álvarez, en las instancias no se tienen los tiempos de servicio, pero si se tienen los tiempos de traslado como una matriz C de costos no negativos. Para cada instancia se repitió 100 veces cada experimento obteniendo un vector de soluciones y tiempos de realización.

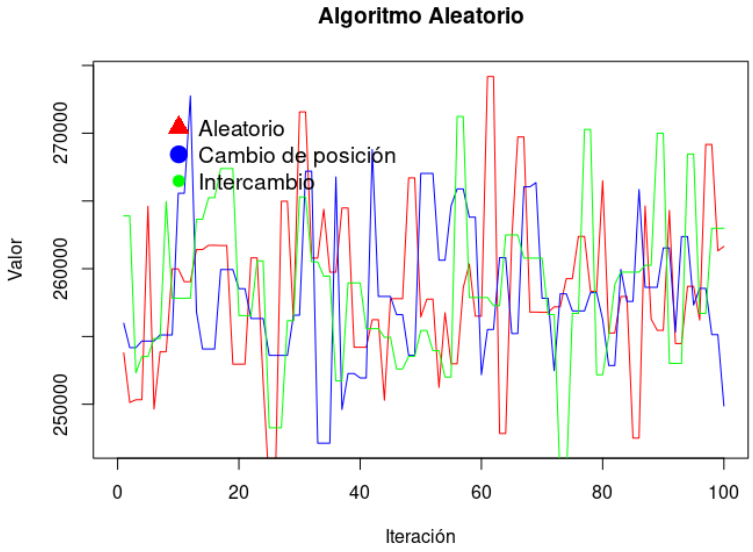
3.1. Objetivos del experimento

El objetivo del experimento es comparar los rendimientos del algoritmo final con los de sus componentes, esto son los métodos de mejora o búsquedas locales.

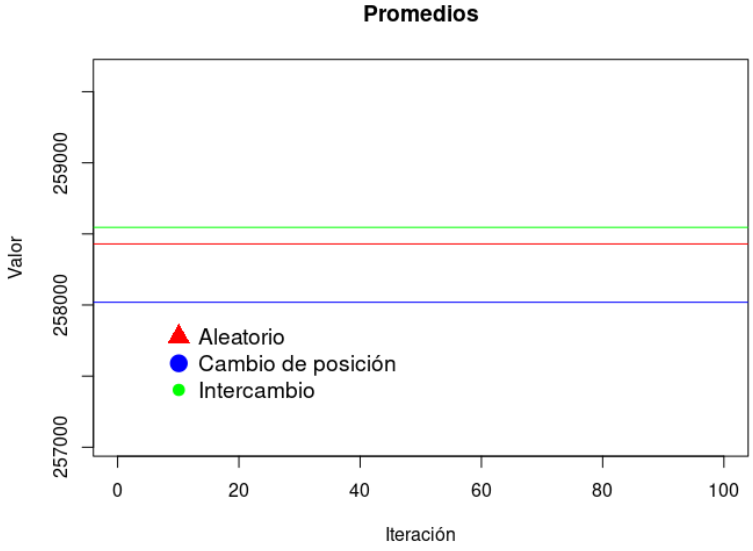
3.2. Primer Experimento

En este primer experimento analizamos los resultados que nos dan la instancia con el algoritmo final y cada uno de los métodos usados.
En la figura 1 podemos darnos cuenta de que el promedio del valor objetivo para los tres métodos es

mayor para la que solo utiliza una búsqueda local con movimientos de intercambio de nodos, siendo la búsqueda local con cambios de posición el que menos tiene en promedio.



(a)



(b)

Figura 1: (a) Valores objetivo para 100 iteraciones de los 3 métodos. (b) Promedio de los valores objetivo, siendo el método de intercambio el que tiene mejor soluciones en promedio.

3.3. Segundo Experimento

En este segundo experimento analizamos los tiempos de realización que tuvo cada iteración de las instancias con el algoritmo final y cada uno de los métodos usados.
En la figura 2 podemos darnos cuenta que el promedio del tiempo que necesito algoritmo de búsqueda local que solo utilizaba el movimiento de intercambio fue mucho menor que al de los otros dos métodos. El algoritmo aleatorio debido a su naturaleza aleatoria e inconstante es el que en promedio llega a tardar más.

4. Conclusión

En conclusión podemos observar que un algoritmo que utiliza como método de mejora el movimiento de intercambio de nodos, llega a dar mejores valores objetivos en menos tiempo requerido, esto es debido a la naturaleza iterativa del cambio de posición el cual llega a tardar más tiempo. El algoritmo aleatorio en términos de evaluación se encuentra en un término medio, debido a su aleatoriedad al escoger a ambos métodos, sin embargo, su tiempo de cálculo resulta ser mayor promedio que al emplear únicamente uno, por lo cual se puede decir que no es el método más eficiente, aunque si es el que más soluciones variadas encuentra.

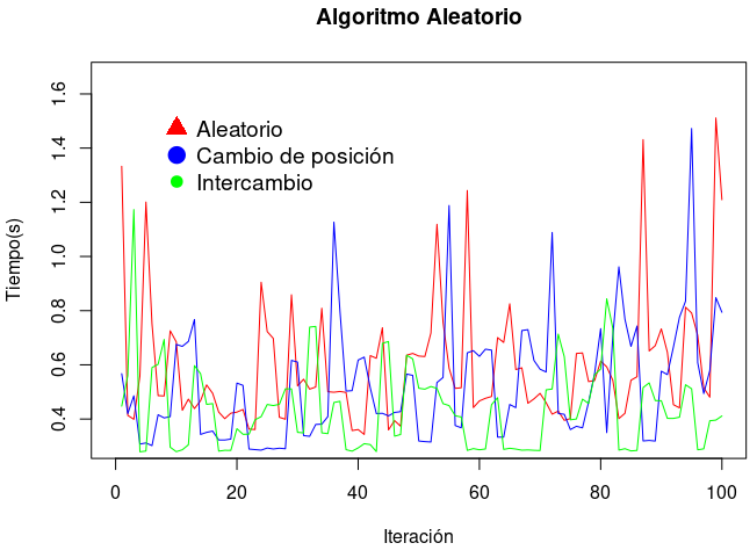
Referencias

[1] Francisco Angel-Bello, Ada Alvarez e Irma García. “A multi-start procedure for the minimum latency problem”. En: *IFAC Proceedings Volumes (IFAC-PapersOnline)* 46.9 (2013), págs. 436-441. ISSN: 14746670. DOI: 10.3182/20130619-3-RU-3018.00107.

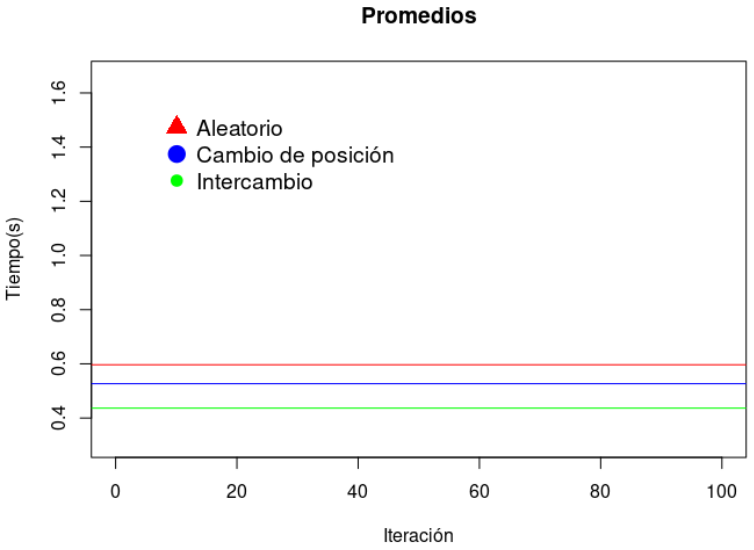
[2] Kenneth R Fox. “Production scheduling on parallel lines with dependencies”. Tesis doct. Johns Hopkins University, 1973.

[3] Isabel Méndez-Díaz, Paula Zabala y Abilio Lucena. “A new formulation for the Traveling Deliveryman Problem”. En: *Discrete Applied Mathematics* 156.17 (2008). Cologne/Twente Workshop on Graphs and Combinatorial Optimization, págs. 3223-3237. ISSN: 0166-218X. DOI: https://doi.org/10.1016/j.dam.2008.05.009. URL: https://www.sciencedirect.com/science/article/pii/S0166218X08002163.

[4] Amir Salehipour, K Sörensen y Peter Goos. “An efficient GRASP+ VND metaheuristic for the traveling repairman problem”. En: *Working Papers* 32.0 (2008), págs. 1-15. URL: http://ideas.repec.org/p/ant/wpaper/2008008.html.



(a)



(b)

Figura 2: (a) Tiempos de calculo para 100 iteraciones de los 3 métodos. (b) Promedio de los tiempos de calculo, siendo el método de intercambio el que menor tiempo tarda en promedio.