Métodos Constructivos

Camilo Andrés Rodríguez Garzón

March 20, 2018

1 Descripción

En el presente trabajo se desea resolver el problema del viajante. Para esto, se elaboró dos métodos constructivos que permitiecen obtener una ruta y de esta manera resolver dicho problema. Para este trabajo se eligieron el método del vecino más cercano y un método creado y basado en el método pilot (url con implementación de los métodos Tsp). A continuación se presenta el algoritmo correspondiente a cada uno de los métodos:

Algorithm 1 Greedy Tsp

procedure BUILD ROUTE(nameFile)
    $coor \leftarrow ReadInput(nameFile)$ ▷ obtener las coordenadas del archivo
    $coorR \leftarrow coor.pop(0)$ ▷ se obtiene la primera coordenada y se elimina de la lista
    $dt \leftarrow 0$ ▷ distancia total en 0
    while *coor* **do**
        $d, p_2 \leftarrow NearestNeighbors(coor[length - 1])$ ▷ se obtiene las coordenadas del vecino más cercano y su distancia
        $coor.remove(p_2)$ ▷ se remueve la coordenada p_2 de la lista
        $dt \leftarrow dt + d$
        $coorR.append(p_2)$
    end while
    $coorR.append(coorR[0])$ ▷ se une punto final con inicial
    $drawTsp(coorR.x, coorR.y, dt)$
end procedure

Algorithm 2 Semi-Pilot Tsp

```
1: procedure BUILD ROUTE(nameFile, breadth, pilot)
2:     coor ← ReadInput(nameFile)    ▷ obtener las coordenadas del archivo
3:     coorR ← coor.pop(0)    ▷ se obtiene la primera coordenada y se elimina
   de la lista
4:     dt ← 0                                              ▷ distancia total en 0
5:     coorBase ← copy(coor)          ▷ se hace una copia de las coordenadas
6:     buildPilot(coorBase, pilot)    ▷ se construye pilot y se finaliza ruta con
   nearest neighbors
7:     dt ← dt + getDistance(coorR[length − 1], coorR[0])          ▷ se obtiene
   distancia final
8:     coorR.append(coorR[0])                    ▷ se une punto final con inicial
9:     drawTsp(coorR.x, coorR.y, dt)
10: end procedure
11:
12: procedure BUILD PILOT(root, pilot, breadth)
13:     root.children ← []
14:     if pilot ≥ 1 then                ▷ caso cuando es usado el método pilot
15:         root.children ← addChild(root, breadth)
16:         pilot ← pilot − 1
17:     else                          ▷ caso cuando se termina de completar la ruta
18:         root.children ← addChild(root, 1)
19:     end if
20:     if root.children is not empty then
21:         for child in root.children do
22:             buildPilot(child, pilot, breadth)
23:         end for
24:     else                       ▷ cuando la construcción de la ruta es terminada
25:         if dt == 0 or dt > root.d then                ▷ se guarda la mejor ruta
26:             dt ← root.d
27:             while root do
28:                 coorR.insert(0,root.value)    ▷ se inserta en la lista resultante
   en la primera posición las coordenadas
29:                 root = root.parent
30:             end while
31:         end if
32:     end if
33: end procedure
```

Se eligió dichos métodos y esto se resume al hecho de aprender en un ambiente más real cuales son las dificultades de implementarlos así como también de adquirir la experiencia de poder construir cada unos de estos.

2 Resultados

2.1 Datos pr439

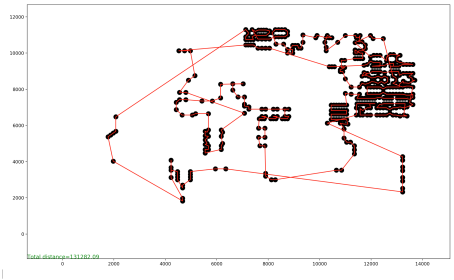

Figure 1: Tsp greedy data pr439

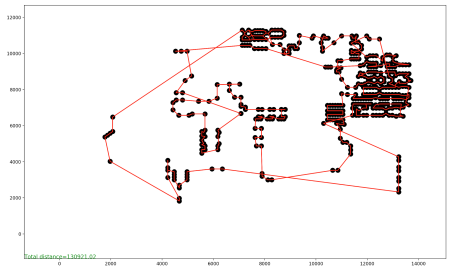

Figure 2: Tsp semipilot data pr439

Table 1: Comparativa de tiempo de ejecución y distancia recorrida de Algoritmos para pr439

Algoritmo	Tiempo Inicial	Tiempo final	Distancia
Vecino más cercano	2018-03-05 23:15:40.434047	2018-03-05 23:15:40.813967	131282.09
Semi-pilot	2018-03-05 23:11:56.054095	2018-03-05 23:11:56.874281	130921.02
TSPLIB	-	-	107217

2.2 Datos linhp318

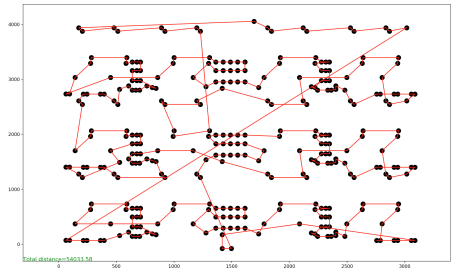

Figure 3: Tsp greedy data linhp318

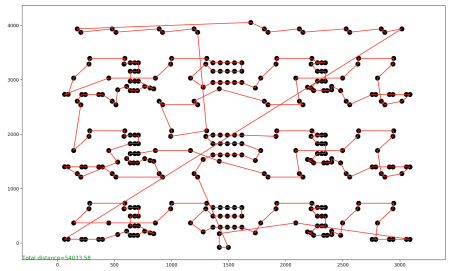

Figure 4: Tsp semipilot data linhp318

Table 2: Comparativa de tiempo de ejecución y distancia recorrida de Algoritmos para linhp318

Algoritmo	Tiempo Inicial	Tiempo final	Distancia
Vecino más cercano	2018-03-05 23:19:02.562361	2018-03-05 23:19:02.910580	54033.58
Semi-pilot	2018-03-05 23:21:31.198604	2018-03-05 23:21:31.767005	54033.58
TSPLIB	-	-	41345

2.3 Datos ch150

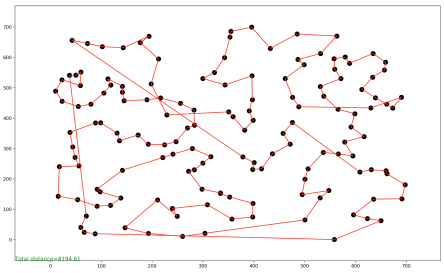

Figure 5: Tsp greedy data ch150

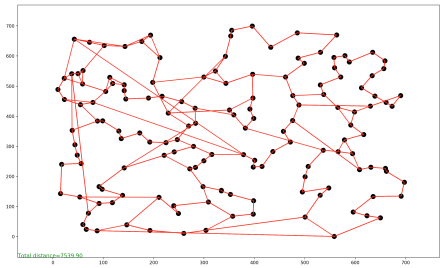

Figure 6: Tsp semipilot data ch150

Table 3: Comparativa de tiempo de ejecución y distancia recorrida de Algoritmos para ch150

Algoritmo	Tiempo Inicial	Tiempo final	Distancia
Vecino más cercano	2018-03-05 23:23:58.057231	2018-03-05 23:23:58.404496	8194.61
Semi-pilot	2018-03-05 23:24:53.177867	2018-03-05 23:24:53.578829	8145.80
TSPLIB	-	-	6528

2.4 Datos Berlin52

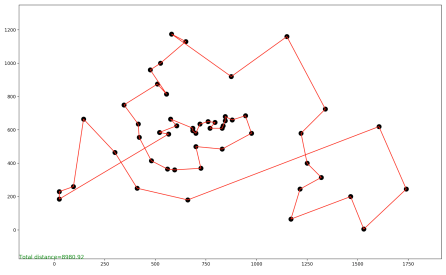

Figure 7: Tsp greedy data berlin52

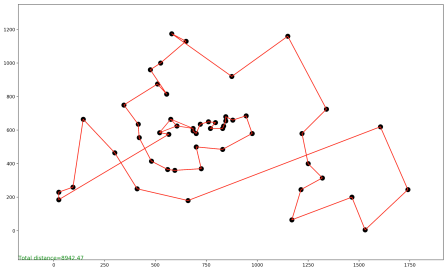

Figure 8: Tsp semipilot data berlin52

Table 4: Comparativa de tiempo de ejecución y distancia recorrida de Algoritmos para berlin52

Algoritmo	Tiempo Inicial	Tiempo final	Distancia
Vecino más cercano	2018-03-05 23:26:28.184412	2018-03-05 23:26:28.520302	8980.92
Semi-pilot	2018-03-05 23:27:14.091881	2018-03-05 23:27:14.421187	8942.47
TSPLIB	-	-	7542

3 Conclusiones

- El método de semi-pilot para un pilot de 2 genera mejor solución que el método del vecino más cercano.
- Los tiempos de ejecución son más extensos en el algoritmo semi-pilot, esto es debido a la evaluación de varios caminos.
- El método semi-pilot para pilot muy grandes los tiempos de ejecución se elevan exponencialmente y las soluciones no mejoran significativamente.
- El método del vecino más cercano es un algoritmo muy potente en relación a los tiempos de ejecución.
- En los datos de linhp318 los dos algoritmos obtuvieron las mismas distancias lo que indica que la estrategia semi-pilot es vulnerable a ciertos patrones en las coordenadas.
- Ambos algoritmos por la estrategia greedy que manejan son castigados al unir la coordenada final con la inicial.
- Las implementaciones realizadas en TSPLIB superan significativamente a las soluciones construidas.