Instituto Tecnológico Autónomo de México

Investigación de Operaciones

Proyecto Final

Algoritmo ara el problema del agente viajero

Elaborado por:
Pablo Barranco-151528
Sofía De la Mora - 160062
José Miguel Carmona Jiménez - 157600
Francisco Castañeda - 155664
Gerardo Hurtado - 157389

Profesor:
Marya Nuñez Lopez

12 de Diciembre del 2019

Índice

1. El problema del agente viajero

El problema del viajero o TSP llamado así por las siglas en inglés de *Travelling Salesman Problem* fue planteado por primera vez en 1930. Este tiene como objetivo hallar la ruta más corta entre un numero n de ciudades, tal que todas sean visitadas y se regrese a la ciudad de donde se partió. El TSP es famoso por su complejidad computacional cuando se trata de un numero considerablemente grande de ciudades. Por lo mismo, se utiliza como test para medir la eficiencia en tiempo de ejecución para métodos de optimización.

El propósito de este trabajo es resolver el problema del viajero para el caso de 634 ciudades en Luxemburgo. Con el objetivo de visitar todas las ciudades del país de la manera más eficiente posible (en este caso el viajero considera que una ruta es más eficiente que otra si tiene que recorrer una menor distancia total).

Para dar contexto al lector Luxemburgo tiene una superficie 2.586 km^2, lo que representa aproximadamente dos terceras partes de la zona metropolitana del Valle de México. Analizaremos 634 localidades de Luxemburgo; por lo tanto, hay 633! recorridos posibles, un número exageradamente grande, por lo que tendría un gran costo computacional, si se considera que el número más grande que se puede guardar en una maquina de 64-bits es 100! Entonces, debido a la imposibilidad de calcular todas las rutas posibles optamos por implementar un algoritmo en Python que calcule la ruta óptima.

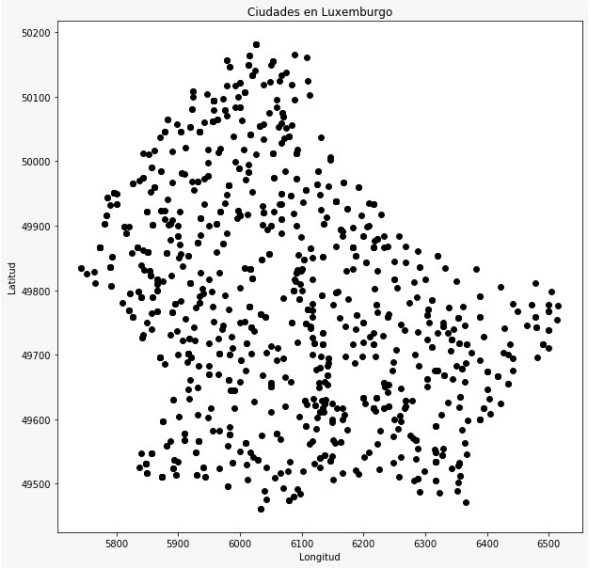

Figura 1: Posición de todas las ciudades de Luxemburgo

2. El algoritmo

Queremos encontrar la ruta más eficiente para recorrer todas las ciudades de Luxemburgo, para esto utilizamos las coordenadas geográficas de todas las ciudades: Las distancias entre ciudades se calculan de la manera usual (métrica euclidiana) y se redondean al entero más cercano. Suponemos que el costo de viajar de una ciudad a otra es el mismo que el de viajar en la dirección opuesta y que se tienen que recorrer las 634 ciudades totales.

Definimos los siguientes términos antes de continuar:

- C - el conjunto de todas las ciudades de luxemburgo
- c_i - el nodo que representa a la ciudad en la posición i de la lista
- $d_{i,j}$ - la distancia de la ciudad en la posición i de la lista a la que se encuentra en la posición j, redondeada al entero más cercano.

Dado lo anterior, se obtiene la matriz de adyacencia D con entradas $d_{i,j}$ asociada al problema de Luxemburgo. A continuación se presenta una submatriz de D con las primeras 10 ciudades de la lista. Nótese que la matriz D resulta ser simétrica, dado que el grafo no es dirigido, y se puede ir en ambas direcciones y se tienen los mismos costos por arista.

0	457	473	346	350	474	582	376	582	307
457	0	144	198	587	21	147	207	147	166
473	144	0	313	678	158	137	117	137	173
346	198	313	0	392	202	344	309	344	196
350	587	678	392	0	593	734	626	734	514
474	21	158	202	593	0	143	227	143	185
582	147	137	344	734	143	0	252	0	275
376	207	117	309	626	227	252	0	252	118
582	147	137	344	734	143	0	252	0	275
307	166	173	196	514	185	275	118	275	0

Tabla 1: Matriz de distancias para las primeras diez ciudades

Para encontrar la mejor elección de este conjunto de arcos creamos las siguientes variables $x_{i,j}$ como sigue:

$$x_{i,j} = \begin{cases} 1 & \text{si la ciudad } i \text{ y la ciudad } j \text{ están conectadas} \\ 0 & \text{si la ciudad } i \text{ y la ciudad } j \text{ no están conectadas} \end{cases}$$

Lo que buscamos es entonces minimizar la suma de todas las distancias recorridas por los arcos, sin tomar en cuenta aquellas que no recorrimos. Siguiendo el procedimiento de Taha, esto se puede formular como el siguiente problema de programación linear:

$$\begin{aligned} \text{mín } z = & \sum_{i=1}^{634}\sum_{j=1}^{634} d_{i,j}x_{i,j} \\ \text{s.a. } & \sum_{i=1}^{n} x_{i,j} = 1 \quad \forall j \in \{1,2,\ldots,634\} \\ & \sum_{j=1}^{n} x_{i,j} = 1 \quad \forall i \in \{1,2,\ldots,634\} \\ & x_{i,j} \in \{0,1\} \quad \forall i,j \in \{1,2,\ldots,634\} \end{aligned}$$

Para evitar que se creen "subrutas", tenemos que agregar una restricción adicional. Sea S un subconjunto propio de C.

$$\sum_{\forall (i,j)\in S \;\; i\neq j} x_{i,j} \leq |S| - 1, \qquad \forall S \subset C, S \neq \emptyset$$

Esta restricción se hace con el fin de que la trayectoria no caiga en ciclos en algún nodo.

3. Implementación

Se nos proporcionaron 980 datos sobre longitud y latitud de ciudades; sin embargo, previo a la implementación, pudimos observar que dentro de los datos había coordenadas repetidas. Por lo cual, eliminamos los datos repetidos; es decir, ciudades repetidas.
Para resolver este problema, nuestro primer intento fue implementar el método de *Ramificación y Acotamiento*. En la practica pudimos observar que el algoritmo resolvía en un tiempo considerablemente bueno cuando usábamos pocas ciudades. Pero al resolver para mas de 100 ciudades resultó

exageradamente costo en tiempo (aproximadamente 4 hrs). Debido a esto, se implemento el algoritmo heurístico del *Vecino más cercano*, con lo cual logramos reducir el tiempo computacional hasta 22 minutos con 75 segundos.
Lo que hace el algoritmo del *Vecino mas cercano* es lo siguiente.

1. Escoger un nodo en el cual iniciar.
2. Ver el costo de todos los posibles arcos que conectan con el nodo inicial que no han sido visitados y escoger el nodo mas cercano (el de menor costo).
3. Repetir el proceso hasta que todos los nodos sean visitados al menos una vez.
4. Verificar si todos los nodos han sido visitados. Si es así, regresar al nodo inicial.
5. Traficar y escribir la ruta, y calcular la distancia de la ruta.

Una manera mas detallada de ver el funcionamiento operativo del algoritmo se puede ver en la siguiente figura del pseudo algoritmo.

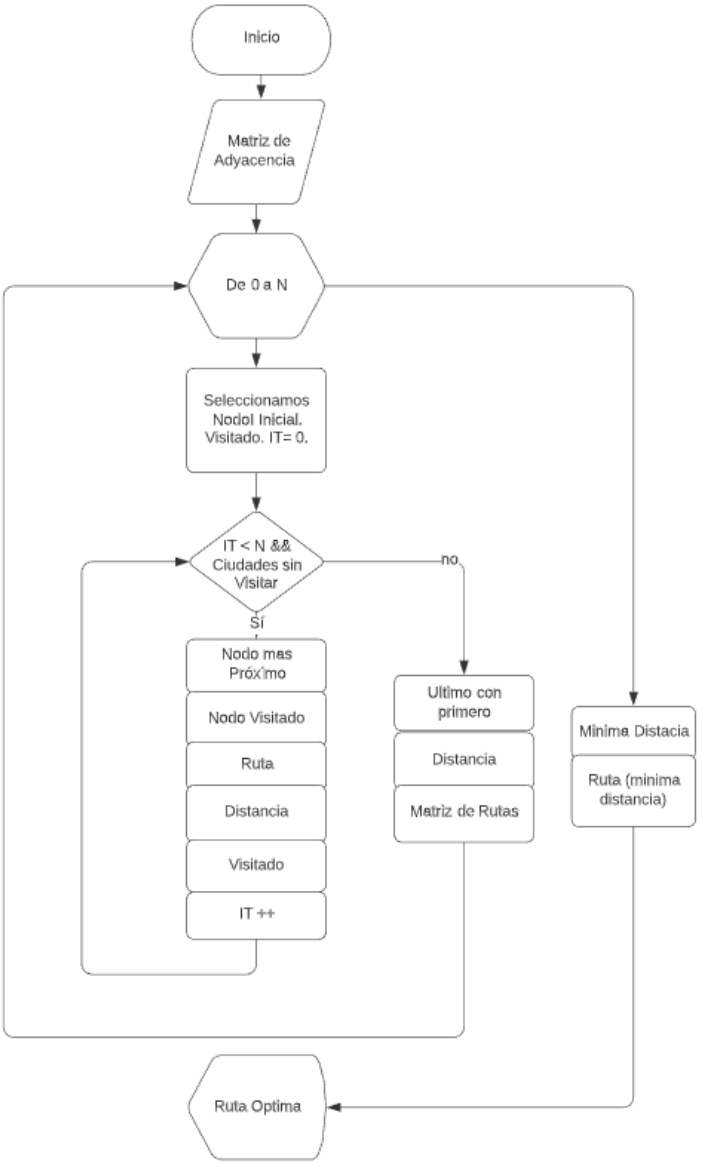

Figura 2: Pseudo Algoritmo

La implementación heurística, a pesar de no asegurar optimalidad, puede llegar a aproximar un resultado óptimo con una probabilidad de 2 % - 3 % de errar.
De la anterior figura, podemos ver que al tener un *for* dentro de un *while*, el orden del algoritmo sería cuadrado, pero debido a que al terminar el algoritmo se realiza una búsqueda secuencial, eso eleva la complejidad computación a un orden cubico. Esto se podrá observar mejor en el siguiente apartado de resultados.

4. Resultados

Se realizo una prueba de complejidad del algoritmo para verificar el orden cúbico propuesto en el inciso anterior. La prueba se realizo seleccionando las primeras n ciudades de la lista y tomando el tiempo que tomaba al algoritmo encontrar la ruta óptima utilizando la heurística de vecino más cercano. La implementación se llevo a cabo en un *Jupyter Notebook* y el tiempo se midio utilizando la herramienta *timeit* de Python. Finalmente, la prueba se realizo inicialmente con 10 ciudades y se añadieron diez ciudades en cada iteración hasta llegar a 610.

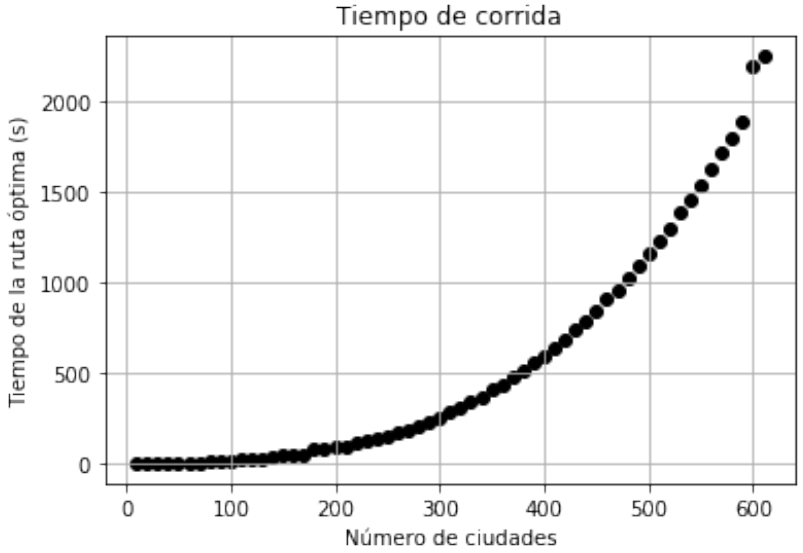

Figura 3: Prueba de complejidad del algoritmo

Se observa un crecimiento suave en el tiempo de corrido de algoritmo en relación con el número de ciudades a excepción de las últimas dos iteraciones, esto es probablemente debido a una perturbación que se le hizo a la computadora mientras iteraba el código. Aún asi, creemos que esto es evidencia suficiente de la complejidad del algoritmo.

Para verificar el orden cúbico se tomaron los datos generados en el experimento y se les aplicó raíz cúbica. Los resultados se pueden apreciar en la siguiente gráfica.

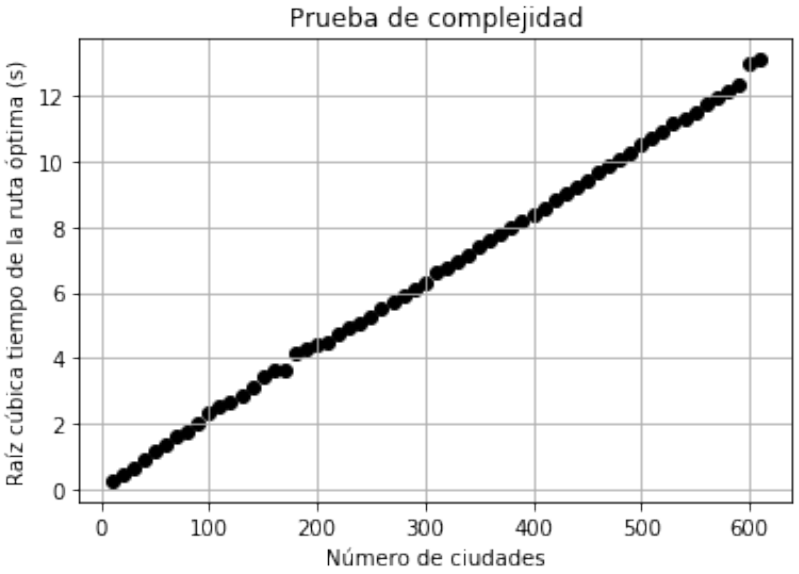

Figura 4: Evidencia del orden cúbico del algoritmo

Se aprecia un crecimiento lineal en este caso y gracias a eso podemos comprobar el orden teórico de grado tres. Adicionalmente, generamos la distancia de la ruta óptima para todas las iteraciones del experimento. Los resultados se pueden apreciar en la siguiente gráfica

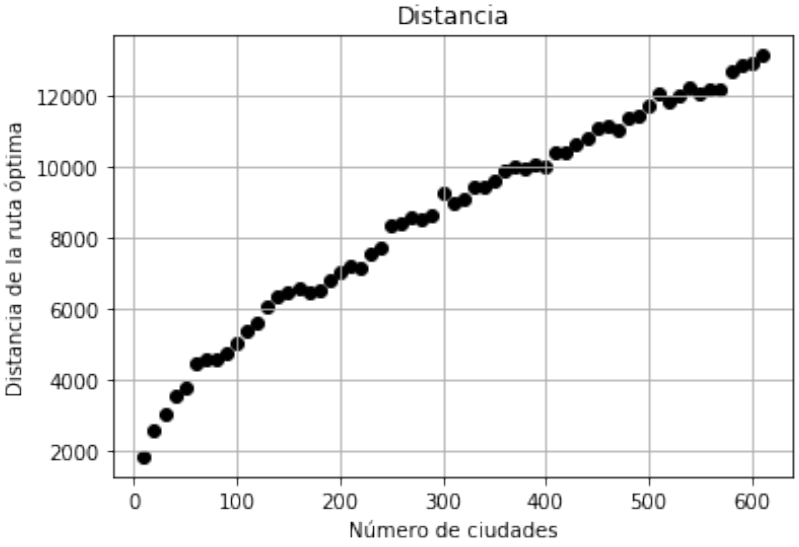

Figura 5: Distancia por iteraciones

Un detalle que puede saltar a la vista en primer momento es que el incrementar el número de ciudades no necesariamente incrementa la distancia total del recorrido. Si bien la tendencia general es de un crecimiento existen puntos por ejemplo alrededor de cien y trescientas ciudades donde las siguientes iteraciones del algoritmo reducen la distancia total. Esto no es incorrecto, aún cuando parezca contra intuitivo.

Para ejemplificar esto consideremos el caso con 22 ciudades:

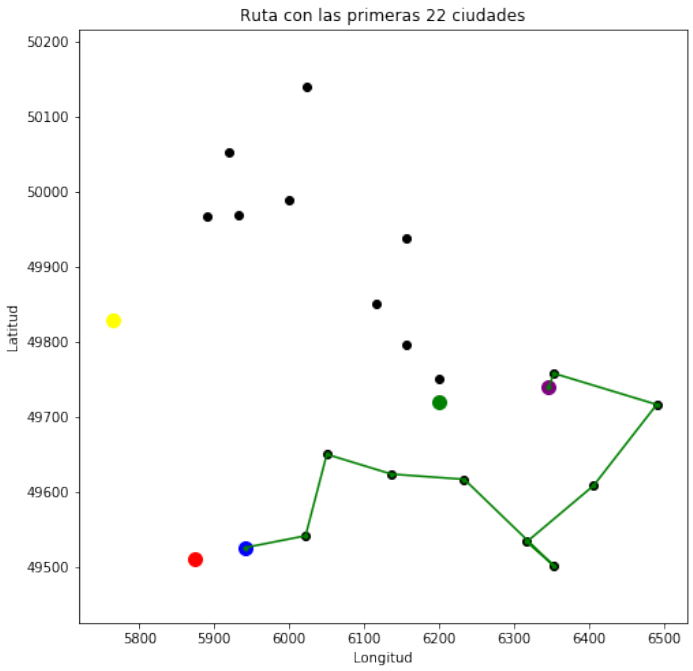

Figura 6: Ruta óptima con 22 ciudades

El recorrido inicia en el nodo de color morado y utilizando la heurística conecta todos los nodos más cercanos hasta llegar al nodo azul. El nodo rojo representa la ciudad numero 23. Si consideramos solo 22 ciudades el recorrido se termina de la siguiente forma:

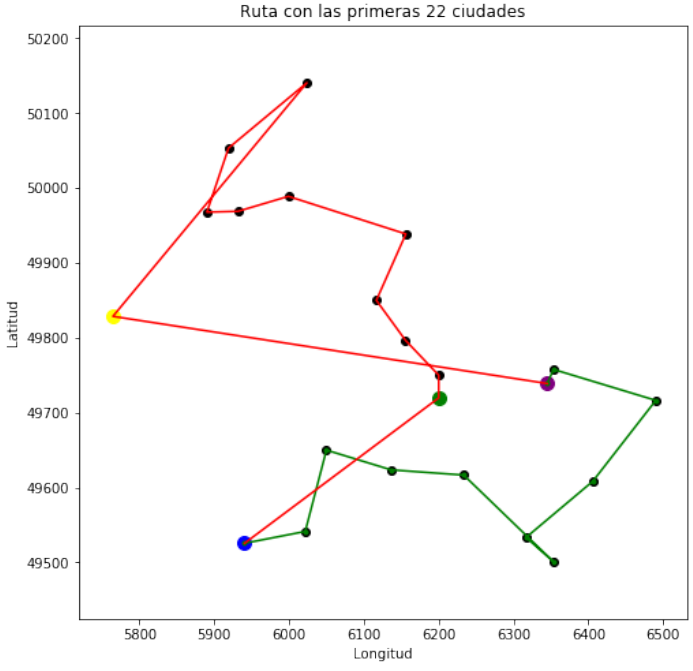

Figura 7: Ruta óptima con 22 ciudades

Al no tener otro nodo cercano la ruta salta del nodo azul al verde y sigue todos los nodos por la parte de arriba hasta llegar al amarillo punto en el cual regresa al nodo original. Esta ruta es bastante costosa debido a este último paso.

Consideremos ahora el caso donde añadimos el nodo rojo (ciudad 23):

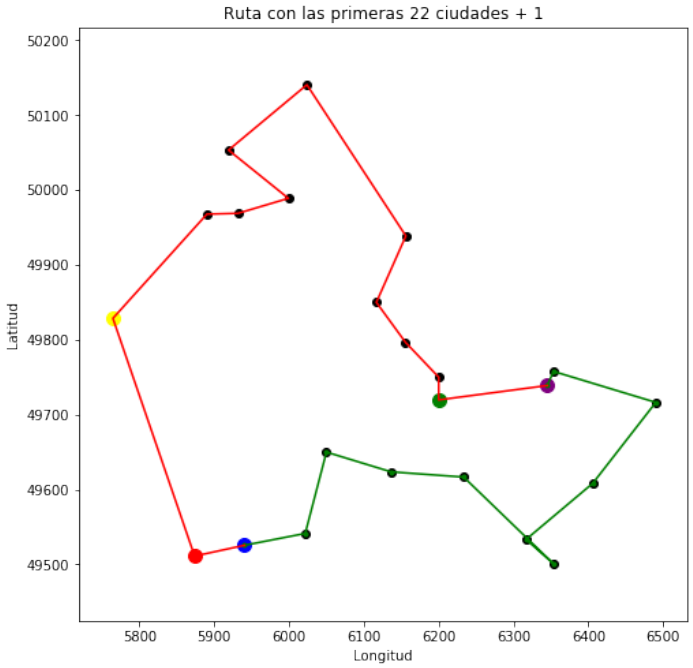

Figura 8: Cambio de la ruta añadiendo la ciudad número 23

Ahora el nodo azul se conecta con el nodo rojo y este a su vez con el amarillo. El camino continua normalmente por la parte de arriba hasta llegar al nodo verde y de ahí regresa al nodo inicial, pero esta vez de una forma mucho menos costosa. La adición del nodo rojo entonces implica un ahorro de unas 400 unidades aproximadamente en la ruta.

Ahora bien en cuanto al resultado obtenido para el problema, obtuvimos la siguiente ruta, como aproximación a la ruta óptima.

560− > 58− > 547− > 326− > 536− > 439− > 53− > 115− > 154− > 590− > 164− > 166− > 143− > 155− > 211− > 205− > 93− > 489− > 364− > 358− > 610− > 299− > 278− > 492− > 599− > 156− > 346− > 179− > 273− > 535− > 351− > 412− > 517− > 252− > 397− > 396− > 222− > 473− > 542− > 124− > 208− > 14− > 36− > 480− > 472− > 89− > 327− > 330− > 37− > 503− > 87− > 169− > 303− > 626− > 15− > 486− > 254− > 206− > 521− > 428− > 69− > 171− > 572− > 147− > 509− > 264− > 432− > 105− > 210− > 186− > 370− > 424− > 251− > 32− > 228− > 161− > 17− > 45− > 445− > 290− > 265− > 218− > 587− > 413− > 277− > 243− > 342− > 31− > 447− > 90− > 437− > 459− > 613− > 64− > 68− > 528− > 514− > 355− > 18− > 468− > 42− > 187− > 110− > 419− > 530− > 24− > 598− > 525− > 307− > 125− > 304− > 580− > 158− > 239− > 593− > 133− > 427− > 163− > 91− > 274− > 293− > 360− > 289− > 390− > 569− > 233− > 75− > 393− > 44− > 389− > 608− > 601− > 234− > 295− > 172− > 527− > 285− > 375− > 440− > 531− > 253− > 132− > 86− > 495− > 12− > 508− > 332− > 226− > 162− > 404− > 442− > 30− > 564− > 352− > 622− > 464− > 385− > 136− > 317− > 579− > 603− > 291− > 176− > 391− > 276− > 401− > 444− > 33− > 402− > 70− > 523− > 257− > 235− > 184− > 99− > 261− > 225− > 8− > 119− > 463− > 456− > 417− > 378− > 88− > 367− > 159− > 198− > 219− > 229− > 97− > 183− > 57− > 255− > 127− > 182− > 334− > 181− > 451− > 544− > 596− > 586− > 407− > 577− > 350− > 562− > 546− > 388− > 1− > 624− > 386− > 146− > 325− > 526− > 549− > 232− > 47− > 529− > 116− > 39− > 485− > 600− > 609− > 48− > 405− > 584− > 153− > 380− > 589− > 466− > 52− > 578− > 73− > 55− > 194− > 49− > 369− > 372− > 320− > 573− > 227− > 505− > 504− > 268− > 140− > 567− > 266− > 324− > 452− > 415− > 359− > 54− > 149− > 328− > 259− > 230− > 470− > 371− > 556− > 118− > 177− > 56− > 263− > 576− > 616− > 487− > 74− > 518− > 506− > 287− > 571− > 10− > 394− > 522− > 59− > 60− > 627− > 185− > 108− > 209− > 106− > 411− > 11− > 545− > 204− > 516− > 555− > 221− > 379− > 408− > 35− > 122− > 301− > 501− > 16− > 430− > 362− > 104− > 23− > 461− > 363− > 28− > 551− > 376− > 581− > 309− > 395− > 347− > 297− > 306− > 606− > 217− > 197− > 368− > 170− > 139− > 313− > 553− > 333− > 540− > 247− > 292− > 537− > 615− > 114− > 336− > 25− > 214− > 193− > 189− > 249− > 246− > 51− > 557− > 414− > 454− > 377− > 267− > 335− > 256− > 620− > 337− > 554− > 13− > 319− > 548− > 72− > 592− > 270− > 373− > 344− > 345− > 425− > 180− > 200− > 128− > 275− > 421− > 121− > 341− > 95− > 145− > 467− > 533− > 496− > 85− > 103− > 318− > 431− > 399− > 76− > 207− > 381− > 223− > 494− > 120− > 512− > 188− > 216− > 196− > 173− > 129− > 312− > 272− > 202− > 43− > 366− > 81− > 446− > 475− > 50− > 148− > 552− > 418− > 315− > 262− > 220− > 392− > 288− > 619− > 245− > 582− > 321− > 160− > 92− > 281− > 374− > 175− > 4− > 400− > 131− > 130− > 566− > 450− > 561− > 449− > 614− > 570− > 260− > 633− > 298− > 354− > 565− > 286− > 409− > 340− > 199− > 62− > 594− > 507− > 568− > 84− > 365− > 67− > 40− > 403− > 460− > 63− > 71− > 539− > 384− > 476− > 339− > 478− > 458− > 316− > 469− > 244− > 167− > 595− > 192− > 3− > 98− > 201− > 248− > 20− > 453− > 101− > 621− > 271− > 21− > 322− > 236− > 433− > 96− > 625− > 559− > 356− > 493− > 575− > 237− > 102− > 80− > 283− > 484− > 258− > 311− > 481− > 583− > 398− > 150− > 574− > 422− > 607− > 471− > 279− > 142− > 429− > 482− > 349− > 550− > 611− > 5− > 113− > 524− > 602− > 410− > 511− > 34− > 308− > 438− > 604− > 38− > 165− > 443− > 284− > 416− > 231− > 65− > 269− > 305− > 387− > 280− > 178− > 515− > 534− > 434− > 242− > 238− > 563− > 435− > 436− > 383− > 126− > 77− > 499− > 6− > 41− > 250− > 628− > 215− > 195− > 502− > 2− > 357− > 302− > 420− > 82− > 151− > 329− > 632− > 612− > 26− > 94− > 9− > 152− > 112− > 538− > 490− > 491− > 462− > 241− > 488− > 623− > 457− > 519− > 27− > 46− > 137− > 61− > 477− > 111− > 532− > 585− > 426− > 343− > 558− > 479− > 134− > 617− > 7− > 618− > 29− > 323− > 630− > 465− > 361− > 203− > 543− > 353− > 500− > 541− > 631− > 588− > 597− > 79− > 19− > 498− > 157− > 168− > 240− > 448− > 109− > 338− > 66− > 123− > 474− > 497− > 0− > 483− > 605− > 190− > 513− > 212− > 629− > 174− > 294− > 213− > 191− > 224− > 117− > 100− > 83− > 510− > 591− > 296− > 520− > 441− > 382− > 348− > 331− > 78− > 406− > 423− > 455− > 314− > 135− > 144− > 141− > 310− > 282− > 22− > 138− > 107− > 300− > 560

Este mismo recorrido se puede observar mas detalladamente en la siguiente figura.

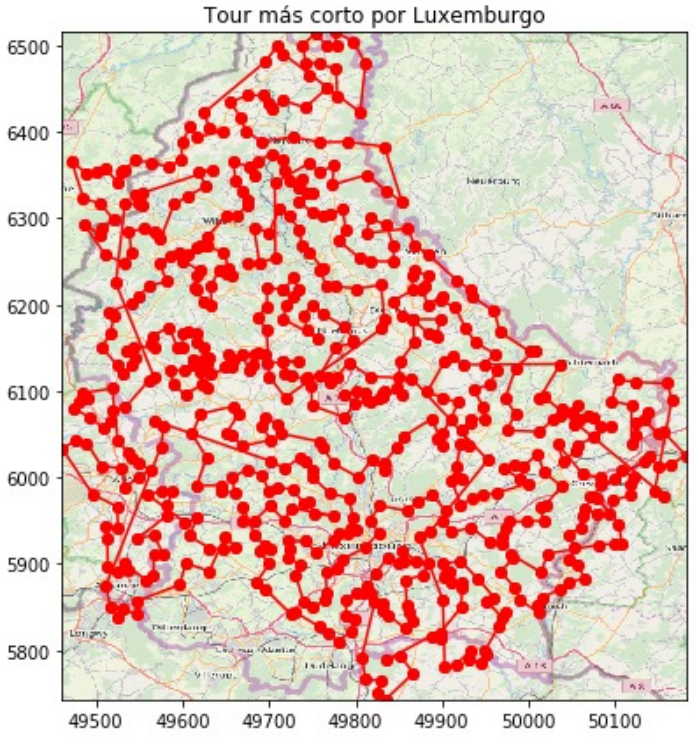

Figura 9: Ruta mínima

Con una distancia total del recorrido de 13126 km, que fue la distancia mínima encontrada por el algoritmo.

5. Bibliografía y Referencias

Referencias

[1] HAMDY A. TAHA *Investigación de Operaciones*H.T (2009) Pearson Educación, México, 2012 9ª edición.