Resolviendo Traveling Salesman Problem With Time Windows haciendo uso de la estrategia Colonia de Hormigas (ACO) modificado*

Germán Andrés López Pacheco
Maestría en Informática
Escuela Colombiana de Ingeniería Julio Garavito
Bogotá D.C, Colombia
german.lopez-p@mail.escuelaing.edu.co

***Abstract*—El problema Traveling Salesman Problem With Time Windows o mejor conocido como TSPTW, es un problema NP-completo, consiste en que un vendedor tiene que pasar por un conjunto de ciudades o nodos, de tal manera que visite todos los nodos por lo menos una vez y regrese al punto de incio, realizando el trayecto más óptimo, muy parecido al TSP original, pero en este caso se le añade una restricción de ventana de tiempo a cada nodo, decir que el vendedor solo puede atender al nodo cuando este este disponible o abierto, básicamente como cumplir con el horario del nodo, si el vendedor se pasa del tiempo disponible del nodo este ya no puede ser atendido. También se le añade un tiempo de servicio a cada nodo sumandole tiempo al vendedor, por lo que se vuelve un problema más díficil que el TSP, ya que no todos los caminos son válidos. Para poder resolver este problema se hace uso de la estrategia Colonia de Hormigas (ACO) el cual es un algoritmo que depende de decisiones tomadas por medio de la probabilidad, para esto se tomó un algoritmo ya implementado que resuelve el TSP y por medio de unas heurísticas se adaptó para poder resolver el TSPTW.**

I. Estado del Arte

Para entender mejor la estrategia de Colonia de Hormigas revisé material audio-visual, uno de los mejores videos [4] que explican esta estrategia es el de Andrés Medina el cual explica a detalle las fórmulas a revisar y mostrando con un ejemplo implementado en Java su funcionamiento. También me basé en el paper "DESARROLLO DE UN ALGORITMO ANT COLONY OPTIMIZATION PARA TAREAS DE CLUSTERING EN APACHE SPARK" [5] realizado por Alejandro Ortiz Martin, estudiante de la Universidad Autónoma de Madrid, en donde en su estado del arte explica las tres fórmulas que componen el algoritmo ACO, en su paper muestra como adaptó el algoritmo a la programación distribuida que ofrece Spark, haciendo uso de clustering para lanzar hormigas en "paralelo". Para acercarme más a la implementación de ACO, tomé un código hecho en Python [3] realizado por el profesor Juan David Carvajal Hernández de la Universidad ICESI de Cali , el cual tiene un vídeo [2] él lo explica de forma detallada como este resuelve el problema TSP. A diferencia de la estrategia original de ACO, el profesor Juan le agrega a su código un parámetro entero llamado "nbest", el cual hace referencia a las hormigas, acotadas a ese entero, que tienen las mejores trayectorias y solo sobre estas se hacía la actualización de las feromonas. Otra cosa a destacar de este código es que no realiza el factor de evaporación de feromonas después de cada iteración, aunque curiosamente en los parámetros está el ratio de evaporación, pero nunca se utiliza en el código. Para resolver el TSPTW, me basé en el paper titulado "A modified ant colony system for solving the travelling salesman problem with time Windows" [1] de los autores Chi-Bin Cheng y Chun-Pin Mao, donde toman la estrategia original del ACO y por medio de heurísticas locales, inspiradas del paper "ALGORITHMS FOR THE VEHICLE ROUTING AND SCHEDULING PROBLEMS WITH TIME WINDOW CONSTRAINT" [6] realizado por Marius M. Solomon, logrando acoplar a las fórmulas las restricciones de ventana de tiempo obteniendo un algoritmo eficiente para resolver este problema. Por último implementé el código a partir del que ya desarrolló el profesor Juan Carvajal y lo adapté para que resuelva el TSPTW, tomando características de la información literaria y audio-visual anteriormente mencionada, pero también agregué mis propias funcionalidades para obtener un mejor resultado, que se explican a continuación.

II. Introducción

Para solucionar el problema de TSPTW, se escogió la estrategia conocida como Optimización por Colonia de Hormigas (ACO), el cual hace uso de técnicas probabilísticas para poder resolver problemas de encontrar el mejor camino o la ruta óptima. Este algoritmo trata de imitar el comportamiento de las hormigas, al momento de conseguir comida para el nido; la forma de comunicación que tienen entre ellas es por medio de las feromonas, las cuales indica cual es el mejor camino para llegar al alimento, dependiendo de su intensidad. Eso en la biología, pero en el algoritmo las hormigas tienen las siguientes ventajas:

- Conocen las distancias entre los diferentes puntos.
- Tienen memoria y no volverán a pasar por un punto que ya visitaron.
- Si la distancia entre dos caminos es la misma optarán por el recorrido que posea más intensidad de feromonas.

III. Explicación del ACO

El algoritmo ACO consta de 3 ecuaciones para poderse efectuar:

- Hay una probabilidad $P^k ij$ para seleccionar un nodo "j" por una hormiga "k", estando en el nodo "i" y está dada por la siguiente fórmula:

$$p_{ij}^k(t) = \begin{cases} \frac{\tau_{ij}^{\alpha}(t)\eta_{ij}^{\beta}(t)}{\sum_{u \in \mathcal{N}_i^k} \tau_{iu}^{\alpha}(t)\eta_{iu}^{\beta}(t)} & \text{if } j \in \mathcal{N}_i^k \\ 0 & \text{if } j \notin \mathcal{N}_i^k \end{cases} \tag{1}$$

  Donde $\tau^{\alpha} ij$ es la intensidad del trayecto de feromonas entre el nodo "i" y el nodo "j", αes el peso o importancia de la intensidad de las feromonas (En esta solución se representan con una matriz de feromonas). Luego está $\eta^{\beta} ij$ el cual es la visibilidad del nodo "j" desde el nodo "i" el cual se interpreta como el inverso de esta distancia o mejor dicho $1/dij$. El denominador de la ecuación es la suma de la fórmula del numerador, pero en este caso se aplica a todos los nodos a los que están conectados a "i". De tal forma que se obtiene la probabilidad de ir del nodo "i" al nodo "j", teniendo en cuenta los otros nodos, esto se hace para todos los nodos que están conectados a "i" para obtener sus probabilidades y luego la hormiga "k" escoge su siguiente destino teniendo en cuenta estas probabilidades. Esta ecuación solo se aplica para los nodos que no han sido visitados, si un nodo ya fue visitado, entonces su probabilidad a ser escogido es de 0.
- Esta ecuación es para la evaporación de feromonas:

$$\tau_{ij}(t) = (1-\rho)\tau_{ij}(t-1) \tag{2}$$

  Donde $0 < \rho < 1$ es la ratio de evaporación de feromonas. Donde $\tau ij(t-1)$ hace referencia al valor de feromona que tenía esa arista entre "i" y "j" pero en la iteración anterior.
- La última ecuación es la actualización o incremento de feromonas de por parte de las n-hormigas que recorrieron todo el grafo.

$$\Delta\tau_{ijk}(t) = \begin{cases} \frac{Q}{L_k}, & \text{if } (i,j) \in T_k, \\ 0, & \text{otherwise,} \end{cases} \tag{3}$$

  En este caso las hormigas irán actualizando su propio recorrido. Donde Q/Lk (para este caso se tomó Q = 1) hace referencia al inverso de la distancia entre un nodo "i" y un nodo "j" y se le suma a esa posición en la matriz de feromonas, por lo que si la distancia es menor la cantidad de feromonas que esparcen es mayor.

IV. Heurísticas locales aplicadas al ACO para resolver el TSPTW

Para poder acoplar el algoritmo ACO para solucionar el problema TSPTW, tomé la solución desarrollada por Chi-Bin Cheng y Chun-Pin Mao, en su artículo titulado "A modified ant colony system for solving the travelling salesman problem with time Windows", donde hacen uso del algoritmo original ACO para resolver el problema TSP y lo expresaron en términos de las ventanas de tiempo y los tiempos de servicio en cada nodo de la siguiente forma:

$$(t_i' + s_i + c_{ij})x_{ij} \le b_j, \quad \forall (i,j), \tag{4}$$

Donde t_i' representa el tiempo de espera que se debe realizar cuando el nodo "i" puede o no puede estar disponible para ser servido, S_i hace referencia al tiempo de servicio en el nodo "i", C_{ij} representa la distancia entre el nodo "i" y el nodo "j", $\chi_{ij} \in \{0,1\}$, el cual es una variable de decisión donde 1 es que el arco(i,j) ya fue recorrido o en otro caso es 0 y por último b_j que representa el tiempo muerto o el tiempo en el que el nodo "j" ya no puede ser atendido. En este punto los autores recurren a un artículo escrito por Marius M. Solomon titulado "Algorithms for the vehicle routing and scheduling problems with time windows constraints", donde especifica que para poder tratar con la restricción de las ventanas de tiempo se incorporó dos heurísticas locales para ser usadas por las hormigas para poder escoger su siguiente movimiento. En este caso se reemplaza el aspecto de la distancia en la ecuación 1, es decir $\eta^{\beta} ij$ por los siguientes dos valores:

$$P_{ij}(t) = \begin{cases} \frac{(\tau_{ij})(g_{ij})^{\beta}(h_{ij})^{\gamma}}{\sum_{z \in N_i} (\tau_{iz})(g_{iz})^{\beta}(h_{iz})^{\gamma}}, & j \in N_i \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

Donde g_{ij} es la primera heurística la cual se explica cómo maximizar el tiempo de holgura en el camino recorrido tal que se minimice el riesgo de violar las restricciones de las ventanas de tiempo en los nodos posteriores del trayecto. Definen a g_{ij} de la siguiente forma:

$$g_{ij} = \begin{cases} \frac{1}{1+\exp(\delta(G_{ij}-\mu))}, & \text{if } G_{ij} \ge 0 \\ 0, & \text{otherwise,} \end{cases} \tag{6}$$

Donde la holgura de la ventana de tiempo del nodo "j" está dada por $G_{ij} = b_j - t_j$ en este caso t_j es el tiempo que se demora ir de un nodo "i" a ese nodo "j". Se le aplica una relación inversa de tal forma que los tiempos de llegada de las hormigas están más cerca al tiempo muerto del nodo "j", con esto va a haber más probabilidad de que los nodos que no han sido visitados tengan un tiempo muerto superior. Por último, está β el se encarga de otorgar un peso de importancia.

Luego está h_{ij} que es la segunda heurística la cual hace referencia a reducir o minimizar el tiempo de espera cuando se llega a un nodo, lo definen de la siguiente forma:

$$h_{ij} = \begin{cases} \frac{1}{1+\exp(\lambda(H_{ij}-\nu))}, & \text{if } H_{ij} > 0 \\ 1, & \text{otherwise} \end{cases} \tag{7}$$

El tiempo de espera del nodo "j" está dado por $H_{ij} = a_j - t_j$ igual que en la ecuación G_{ij}, t_j es el tiempo que se demora ir de un nodo "i" a ese nodo "j". También se aplica una relación inversa de tal forma que el tiempo de llegada de las hormigas está más cerca al tiempo en el que el nodo "j" está disponible

para ser atendido. Por último, está γ, se encarga de otorgar un peso de importancia.

Los autores muestran su algoritmo con las nuevas ecuaciones en acción:

Step 0. Initialization
Set $t = 0$
Set initial value $\tau_{ij}(t)$ for all $(i, j) \in \mathcal{A}$
Place m ants at the depot
Set $\Delta\tau_{ij}(t) = 0$ for all $(i, j) \in \mathcal{A}$
Step 1. Route Construction
For $i = 0$ to n do {n is the number of nodes; $i = 0$ denotes the depot}
For every ant $k = 1$ to m do
If N_i not empty
Step 1.1 Local heuristics
Calculate g_{ij} and h_{ij} from Eqs. (11) and (12), respectively, $\forall j \in N_i$
If $g_{ij} = 0, \forall j \in N_i$, then give up the tour and go to next k {no more feasible paths}
Step 1.2 Move
Choose the node j to move to in accordance with the rules given in Eqs. (9) and (10)
Remove j from N_i
Step 2. Memorize the best tour found up to now, T^*, and its travel time, L^*
Step 3. Reset $N_i = \mathcal{N}$ for all ants
Step 4. Local Updating
For all $(i, j) \in \mathcal{A}$ do
Pheromone updating by Eq. (7)
Step 5. Global Updating
For all $(i, j) \in T^*$ do
Pheromone updating by Eq. (6)
Step 6. Stop Criterion
If the stop criterion is satisfied then stop, otherwise
Go to Step 1

Como se puede notar en el step 4, hay una fórmula que no he explicado, la cual los autores la utilizan para que en cada iteración la hormiga con mejor resultado haga una actualización en la matriz de feromonas, de tal forma que su recorrido tendrá un peso mucho mayor que los otros. Los autores lo denominan "La actualización local" y está dada por esta fórmula:

$$\tau_{ij}(t) = (1 - \omega) \cdot \tau_{ij}(t - 1) + \omega \cdot \Delta\tau_{ij}(t), \quad (8)$$

Pero para mi solución decidí no tomar en cuenta este paso, debido a que este algoritmo depende fuertemente de la probabilidad, por lo que darle doble peso al mejor recorrido de una hormiga por cada iteración puedo estar sesgando mi conjunto de resultados a un mínimo local dejando de lado los mínimos globales.

V. Explicación detallada del algoritmo implementado

El algoritmo a continuación tiene como parámetros, la cantidad de iteraciones que se van a realizar para generación de hormigas, es decir es la condición de terminación, también está la cantidad de hormigas, el ratio de evaporación de feromonas, el peso de importancia en las feromonas, el peso de importancia en el tiempo de holgura y el peso para el mejor tiempo de espera.

Inicialización
Paso 0 Inicializar la matriz de feromonas en 0.
Las n hormigas se posicionan en el nodo 0.
Paso 1 Para cada hormiga hacer
tiempo_hormiga = 0
Para cada nodo hacer
Paso 2 Si algún nodo que no ha sido visitado tiempo_hormiga + dist(prev,next) > b_n
Paso 2.1 Si incumple actualizar la matriz de feromonas disminuyendo las cantidades de la trayectoria que no cumple.
Paso 2.2 Romper el ciclo.
Paso 3 Aplicar la fórmula (5) para obtener las probabilidades de todos los nodos
Paso 4 Escoger el nodo dependiendo de las probabilidades obtenidas.
Paso 5 tiempo_hormiga += dist(prev,next) y tiempo_hormiga = a_n si tiene que esperar
Paso 6 Guardar todas las trayectorias y distancias obtenidas por las hormigas.
Paso 7 Para todos los arcos(i , j) hacer
Aplicar fórmula (2) evaporación de feromonas.
Paso 8 Para cada hormiga hacer
Aplicar fórmula (3) cada hormiga actualiza su trayectoria en la matriz de feromonas aumentando su valor.
Paso 9 Criterio para detenerse
Si el criterio se satisface detener la iteración.
De lo contrario volver al Paso 1.

Como se puede observar esta implementación es muy similar a la que hicieron Chi-BinCheng y Chun-Pin Mao, pero con varias modificaciones, lo primero que se puede notar es que antes de que la hormiga vaya a seleccionar el siguiente nodo, se realiza una validación a todos los nodos que no ha visitado, de tal forma que hace una comparación del tiempo de la hormiga sumado con la distancia de los nodos contra la ventana de tiempo superior de estos mismos nodos y si es mayor, quiere decir que la hormiga encontró un camino inválido, luego se toma esa trayectoria actualizando la matriz de feromonas disminuyendo el valor de esos arcos para que las siguientes hormigas no tengan en cuenta y busquen otra combinación de caminos, luego rompe la iteración y continua la siguiente hormiga. Para disminuir la probabilidad de una trayectoria incorrecta, se intentaron 3 técnicas:

- Disminuir la probabilidad, en igual cantidad, de todos los nodos cuya trayectoria es incorrecta. Esta estrategia no fue efectiva debido a que al hacer esto los nodos que podrían llevar a una trayectoria correcta, al no encontrar un camino completo, disminuia cada vez más su probabilidad, de tal forma que llegaba a un punto donde las hormigas no volvían a optar por ese nodo y en consecuencia nunca encontrarían un camino correcto.
- Disminuir la probabilidad, en forma ascendente, es decir que entre más largo es el camino incorrecto, más probabilidad se le reduce a los últimos nodos. Esta solución, funcionó mejor, pero tampoco fue efectiva, en las primeras iteraciones encontraba caminos más largos, pero al final tuvo el mismo problema que la primera estrategia y las hormigas se quedaban estancadas en un nodo durante la iteración.
- Disminuir la probabilidad solamente en el penúltimo nodo, esta fue la mejor solución, debido a que restarle importancia al nodo antes de siga con un camino imposible, las hormigas no le restarán importancia a los nodos anteriores los cuales pueden tener más opciones para encontrar la trayectoria correcta.

Para que la hormiga escoja el nodo se hace uso de la fórmula (5) que se aplica desde el nodo donde la hormiga se encuentra contra todos los nodos que todavía no han sido visitados, dependiendo de la probabilidad la hormiga escogerá

su próximo destino.

Si hubieron hormigas con trayectos correctos, es decir, que recorrieron todos los nodos sin incumplir la restricción de tiempo, se procede a realizar la formula (2), es decir, la evaporación de feromonas, aquí aplica el ratio de evaporación anteriormente mencionado, se realiza para todos los arcos del grafo.

Luego se aplica la fórmula (3) donde cada hormiga se ocupa de actualizar la matriz de feromonas agregando más cantidad en los arcos de su trayectoria.

VI. Pruebas y resultados

Para las pruebas se realizaron con las 30 intancias otorgadas por SolomonPotvinBengio, los parámetros fueron tomados de esta forma: Parámetros fijos:

- número de iteraciones = 200
- tiempo de ejecición = 60 segundos

Parámetros variables:

- número de hormigas = 250 y 500
- ratio de feronomas = 0.7 y 0.5
- importancia de feromonas = 2,3
- importancia de holgura = 2,3
- importance de tiempo de espera = 2,3

Se realizaron 12 permutaciones de parámetros donde en cada uno se probaron los 30 caso anteriormente mencionados y estos fueron los resultados:

TABLE I
Resultados de ejecutar el ACO modificado para las instancias de SolomonPotvinBengio

Instancia	Nodos	Promedio	Mejor	Óptimo
rc201.1.txt	20	551.4736	542.8902	444.54
rc201.2.txt	26	750.5074364	742.8998	711.54
rc201.3.txt	32	0	0	790.61
rc201.4.txt	26	852.0622417	844.044	793.64
rc202.1.txt	33	0	0	771.78
rc202.2.txt	14	410.4346083	395.546	304.14
rc202.3.txt	29	899.1511	892.1344	837.72
rc202.4.txt	28	0	0	793.03
rc203.1.txt	19	576.8314417	553.6464	453.48
rc203.2.txt	33	0	0	784.16
rc203.3.txt	37	0	0	817.53
rc203.4.txt	15	395.4981917	395.3757	314.29
rc204.1.txt	46	0	0	878.64
rc204.2.txt	33	862.1475467	814.13094	662.16
rc204.3.txt	24	563.030075	550.2731	455.03
rc205.1.txt	14	400.9499417	397.3856	343.21
rc205.2.txt	27	0	0	755.93
rc205.3.txt	35	0	0	825.06
rc205.4.txt	28	854.4538333	817.9342	760.47
rc206.1.txt	4	117.8479	117.8479	117.85
rc206.2.txt	37	0	0	828.06
rc206.3.txt	25	656.3020667	650.6058	574.42
rc206.4.txt	38	0	0	831.67
rc207.1.txt	34	985.3885	985.3885	732.68
rc207.2.txt	31	0	0	701.25
rc207.3.txt	33	0	0	682.40
rc207.4.txt	6	119.6388	119.6388	119.64
rc208.1.txt	38	0	0	789.25
rc208.2.txt	29	660.88962	609.8415	533.78
rc208.3.txt	36	975.81	971.5481	634.44

VII. Conclusiones

- La estrategia ACO al depender tanto de la probabilidad, se deben controlar varios parámetros para que el código no falle, en este caso para resolver TSPTW, en varios intentos los resultados incumplían con las restricciones de tiempo, por lo que en varias ocasiones se ajustó el código de tal forma que la probabilidad no tuviera forma de lanzar excepción.
- Los resultados obtenidos de varias pruebas que se realizaron alternando los parámetros, retornaron resultados decentes, pero en muchos casos se aleja de la respuesta óptima, por lo que se debe indagar más en la estrategia de dar menos importancia a los caminos incorrectos y también hacer mejor uso de la permutación de los parámetros.
- En casos de más de 30 nodos en la mayoría de los intentos el código implementado no retornaba una trayectoria válida, aun así, el número de hormigas, iteraciones y de tiempo fuera bastante amplio.
- Se puede optar por una solución en paralelo, donde cada hormiga en cada iteración hace su recorrido sin depender de que las otras hayan acabado, por lo que el número de hormigas puede ser mucho mayor y la cantidad de iteraciones menor.

References

[1] Chi-Bin Cheng and Chun-Pin Mao. A modified ant colony system for solving the travelling salesman problem with time windows. *Mathematical and Computer Modelling*, 46(9-10):1225–1235, 2007.

[2] Juan David Carvajal Hernandez. Algoritmo de colonia de hormigas en python https://www.youtube.com/watch?v=hseeaaoc-aqab$_c hannel = juandavidcarvajalhernandez$.

[3] Juan David Carvajal Hernandez. Algoritmo de colonia de hormigas tsp https://colab.research.google.com/drive/1ohqtnkrsq-vuhwdnbm9ikyt23ougrbic.

[4] Andres Medina. Computacion emergente optimizacion por colonia de hormigas https://www.youtube.com/watch?v=xzx$_d ku7x8wt = 1489sab_c hannel = andresmedina$.

[5] Alejandro Ortiz Martín et al. Desarrollo de un algoritmo ant colony optimization para tareas de clustering en apache spark. Master's thesis, 2016.

[6] Marius M Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265, 1987.