Resolviendo el problema de recolección de basuras mediante un algoritmo goloso

Alejandro Guzmán[1]

[1] Universidad de los Andes
Cra 1 Nº 18A - 12, 28922, Bogotá, Colombia

Abstract

La recolección de basuras se puede modelar como un problema de ruteo de arcos capacitados, que es un tipo de problema de optimización que implica encontrar las rutas más eficientes que deben tomar los vehículos para recolectar y eliminar la basura. En este tipo de problema, los arcos representan las rutas que pueden tomar los vehículos, y el peso de cada arco representa la distancia de estas rutas. El objetivo de modelar este de esta manera, es para tratarlo como un problema de optimización, en el cual el objetivo es encontrar las rutas que permitirán que los vehículos recolecten toda la basura mientras se mantienen dentro de sus límites de capacidad. En este documento se buscará una solución para este problema mediante la aplicación de un algoritmo goloso, con el fin de brindar una solución factible y de bajo costo computacional.

1. Introducción

Los problemas de ruteo de arcos, se desarrollan en un grafo que consta de nodos y conexiones entre nodos, estas conexiones se denominan arcos cuando están dirigidas de un nodo a otro y aristas cuando no están dirigidos, estos problemas se identifican por tener solicitudes en las aristas de una red. Actualmente los problemas de ruteo de arcos independientemente de ser un problema de ruteo, se investigan por separado, de los problemas enrutamiento de vehículos esto debido a las diferencias en la formulación matemática, así como a las diferencias en las complejidades para solucionar sus respectivas variantes un ejemplo de esto es la comparación entre, el problema del cartero chino (que requiere visitar todos los arcos) el cual puede resolverse en tiempo polinomial, mientras que el problema del viajante de comercio (que requiere visitar todos los nodos) es NP-Hard, sin embargo para ambos tipos de problemas se han implementado una gran variedad de metodologías para su resolución, tanto exactas como heurísticas, tales como: Reinforcement Learning, Genetic Algorithm, Path Scanning, Búsqueda Tabú, entre muchas otras . Los problemas de enrutamiento de arcos poseen una gran variedad de aplicaciones, que incluyen: recolección de basura, mantenimiento y servicios postales a su vez, los problemas prácticos introducen una variedad de restricciones, objetivos combinados y conjuntos de decisiones. El propósito de este estudio es brindar una solución el problema de recolección de basuras, el cual como ya se mencionó se puede catalogar como un problema de ruteo de arcos capacitados mediante la aplicación de un algoritmo goloso.

1.1. Descripción del problema

Uno de los problemas más famosos en el ámbito de la optimización es el de la recolección de basuras, en este caso que se va a tratar que se nos planteó que existen una serie los usuarios conectados por aristas, las cuales tienen un costo y una demanda de recolección asociadas, también poseemos una flota de camiones cada uno de estos con una misma capacidad, estos parámetros están condicionados por las instancias definidas en Golden y Wong [1]. Entendido esto, se debe aplicar un algoritmo que configure las rutas que deben recorrer los camiones en función de atender la demanda de todas aristas, mientras se procura recorrer la mínima distancia posible.

2. Marco teórico

2.1. El CARP y formas de abordarlo

Los problemas de ruteo de arcos capacitados (CARP) es un tipo de problema de optimización combinatoria en el que se debe determinar un conjunto de rutas para que una flota de vehículos las recorran en su totalidad satisfaciendo ciertas restricciones. El objetivo es encontrar las rutas más eficientes que minimicen la distancia total recorrida o el costo total de las rutas, en el CARP cada vehículo tiene una capacidad limitada, que usualmente representa la cantidad máxima de mercancías, material, carga etc. Transportable por un vehículo, este problema generalmente implica la consideración de otras restricciones, como ventanas de tiempo, en las que cada ruta debe visitarse dentro de un marco de tiempo determinado.

Una forma de resolver el CARP es mediante el uso de técnicas de programación matemática, como la programación lineal o la programación entera, estos métodos implican formular el problema como un modelo matemático y usar algoritmos para encontrar la solución óptima, otro enfoque es utilizar algoritmos heurísticos para situaciones en las que se requiera encontrar rápidamente una solución, y no se requiera exactamente la óptima, la aplicación de heurísticas generalmente involucran el uso de reglas simples y eficientes para guiar la búsqueda de una solución, pueden ser útiles para resolver problemas grandes y complejos que son difíciles de resolver con exactitud. En general, el problema de ruteo de arcos capacitados es un problema importante en el campo de la investigación de operaciones y la logística, ya que se ocupa del enrutamiento eficiente de los vehículos para visitar un conjunto de rutas y satisfacer varias restricciones. Al encontrar las rutas óptimas, las organizaciones pueden ahorrar tiempo y recursos y mejorar la eficiencia de sus operaciones [3].

2.1. Greedy algorithm.

Un algoritmo goloso (Greedy algorithm), es un tipo de algoritmo que sigue el principio del comportamiento "codicioso", que consiste en hacer la elección óptima localmente en cada etapa de la optimización con la esperanza de encontrar un óptimo global, en otras palabras un algoritmo goloso toma la mejor decisión en el momento actual sin considerar las consecuencias para los pasos futuros, por ejemplo, considere el problema de encontrar la cantidad mínima de monedas necesarias para completar una cierta cantidad de dinero, un algoritmo codicioso siempre elegiría la moneda más grande disponible en cada paso, con la esperanza de usar la menor cantidad de monedas, entonces, si las monedas disponibles son de 1, 5, 10 y 25 centavos, el algoritmo elegiría primero una moneda de 25 centavos, luego una moneda de 10 centavos, luego una moneda de 5 centavos y finalmente una moneda de 1 centavo para hacer 41 centavos, si bien este enfoque puede funcionar en algunos casos, no se garantiza que encuentre la solución óptima a un problema. En el ejemplo de la moneda, el algoritmo greedy no encontraría el número mínimo de monedas si la cantidad de dinero fuera de 31 céntimos, ya que se necesitarían dos monedas de 10 céntimos y una moneda de 1 céntimo, mientras que el algoritmo voraz elegiría tres monedas de 10 céntimos sin embargo, a pesar de sus limitaciones, el algoritmo voraz puede ser un enfoque útil para resolver problemas de optimización, particularmente cuando el problema tiene propiedades específicas que permiten que la estrategia voraz produzca una solución óptima. También es relativamente fácil de implementar y puede ejecutarse rápidamente para grandes entradas. Sin embargo, es importante considerar cuidadosamente el problema y sus restricciones para determinar si un algoritmo voraz es apropiado [4].

3. Metodología.

3.1 Modelo matemático del problema de recolección

Este modelo es una adaptación del modelo matemático original del problema de ruteo de arcos

capacitado, el cual se centra en disminuir costos y cumplir con las solicitudes de las aristas.

En la ecuación 1 se presenta la función objetivo para el problema descrito, la cual consta en minimizar los costos de recorrer todas las aristas de un espacio.

$$min: \sum_{k \in C} \sum_{a \in A} c_a x_a^k + \sum_{k \in C} \sum_{a \in A} c_a y_a^k \qquad (1)$$

Donde: x_a^k es una variable binaria que representa si la arista es atendido por el camión k, y_a^k es otra variable que representa si la arista es recorrida pero no atendida por el camión p y c_a es la variable que define el costo de pasar por la arista a.

Esta función objetivo está sujeta a las restricciones planteadas en las ecuaciones 2 y 3.

$$\sum_{k \in C} x_a^k = 1 \ \ \forall a \in A \quad (2)$$

Esta restricción es equivalente a decir que todas las aristas deben ser atendidas por un camión al menos una vez.

$$\sum_{a \in A} d_a * x_a^k \leq Q \ \ \forall k \in K \quad (3)$$

En esta restricción se presenta el parámetro d_a y Q estos hacen referencia a la demanda de cada arista y la capacidad del vehículo, la restricción indica que en el momento de satisfacer una demanda se debe respetar que esta no exceda la capacidad del vehículo.

3.2 Algoritmo

En este apartado se explicará el funcionamiento del algoritmo realizado para tratar de resolver el problema de recolección de basuras. Este se encuentra realizado en el lenguaje de programación Python y está basado en los principios de los algoritmos golosos anteriormente mencionados, los pasos de este se exponen a continuación:

1. Se carga la instancia que se desea evaluar y mediante una lectura de datos extraemos los nodos, aristas y dos matrices, una de distancias de llegar de un nodo i a uno j si estos están unidos por una arista y una matriz de demanda para cada i,j también unidos por una arista.
2. A partir de la matriz de distancia utilizar el algoritmo Floyd-Warshall para encontrar la distancia mínima entre cualesquiera pares de nodos en la red.
3. Para todas aristas si esta tiene asociada una demanda, crear la arista inversa es decir que para todo (i,j), crear (j,i) y esta nueva arista tendrá asociado el mismo costo de ser recorrida que la original.
4. Iniciando en el depósito, evaluar las aristas que comienzan en este y agregar a la ruta la primera encontrada y eliminarla del espacio así como a su inversa.
5. De la arista encontrada en el paso anterior, tomar el nodo final y repetir el proceso de búsqueda definido en el paso 4, agregarla a la ruta y eliminarla así como a su inversa, .
6. Repetir el paso 5 mientras el camión tenga capacidad.
7. Cuando la demanda de una arista encontrada en el paso 5 sumada a la capacidad restante del camión supere la capacidad máxima, acabar el recorrido del camión.

7.1. Si la ruta encontrada empieza en el depósito y termina en este cerrar la ruta, guardarla y calcular su costo, de lo contrario revisar paso 7.2.
7.2. Si la ruta encontrada empieza en el depósito pero no termina en este, cerrar la ruta y sumarle a la F.O el costo mínimo de llegar del último nodo visitado al depósito, este costo se obtiene de la matriz de distancias mínimas halladas con el algoritmo Floyd-Warshall, de no ser el caso pasar al paso 7.3
7.3. Si la ruta encontrada no empieza en el depósito pero termina en este, cerrar la ruta y sumarle a la F.O el costo mínimo de llegar del depósito al primer nodo visitado en la ruta presente. Si este no es el caso pasar al paso 7.4
7.4. Si la ruta encontrada no empieza en el depósito y no termina en este, cerrar la ruta y sumarle a la F.O el costo mínimo de llegar del depósito al primer nodo visitado en la ruta presente y sumarle a la F.O el costo mínimo de llegar del último nodo visitado al depósito

8. Volver al paso 3 hasta que ya no quede ninguna arista en el espacio que deba ser atendida.
9. Calcular los costos de cada ruta sumando los costos de recorrer las aristas que las componen y sumarlos para hallar el costo total de satisfacer la demanda del sistema.
10. Fin.

4. Resultados.

Se evaluó el funcionamiento del algoritmo utilizando 5 de las instancias definidas anteriormente y se obtuvieron los siguientes costos para suplir la demanda de las aristas definidas en cada una de estas, estos se pueden observar en la tabla 1.

Tabla 1 Costos siete instancias del CARP

Instance	ID	Vertices	Required edges	BKS	Greedy solution (GS)	CPU Time GS	GAP
gbd 1	1	12	22	329	394	0.02297115	19.76%
gbd 2	2	12	26	365	383	0.00766706	4.93%
gbd 3	3	12	22	324	341	0.00739979	5.25%
gbd 4	4	11	19	337	358	0.0053369	6.23%
gbd 5	5	13	26	445	493	0.0088701	10.79%
gbd 6	6	12	22	344	382	0.0064949	11.05%
gbd 7	7	12	22	332	393	0.0067420	18.37%

Como se puede observar los costos encontrados por el algoritmo para las diferentes rutas en comparación de los óptimos tienen GAP´S bastante variados, siendo para unas instancias una buena metodología y para otras no tanto, esto se basa en un componente pseudoaleatorio presente en las rutas ya que, dependiendo de las aristas presentes en cada instancia ya que, si las aristas de cada nodo al depósito y a su vez del depósito al nodo, son en total una gran parte de las aristas a recorrer el algoritmo no se ve en la necesidad de sumar los costos mínimos de llegar al depósito o salir de este, consiguiendo soluciones más cercanas al óptimo y en dado caso de que este no sea el caso este se verá en la obligación de sumar muchos costos que probablemente no sean necesarios pero que, debido al criterio de eliminación de rutas este debe hacer esto para asegurar que se sale y se llega al depósito.

5. Discusión

Haciendo un análisis de los GAP vemos como los costos de las rutas encontradas por el algoritmo son mucho mayores que las óptimas en algunos casos y bastantes óptimos en otros por lo tanto, se puede

decir que los resultados de la aplicación de este algoritmo goloso, depende mucho de la configuración de la instancia, ya que dependiendo de las aristas este será más o menos óptimo. Esto se puede observar en la figura 1 ya que, en las rutas 1 y 2 el algoritmo encuentra una ruta que comienza en el depósito y termina en este, sin embargo en el momento que se empieza a configurar la ruta 3 al haber eliminado tantos caminos para este punto la única manera de asegurar que se está llegando al depósito es sumando el costo mínimo de llegar del nodo 7 al depósito ya que, en este momento, no existe ninguna manera de lograr esto.

Figura 1 Ejemplo ruta

```
Ruta 1 : (1,2),(2,3),(3,4),(4,1)
Ruta 2 : (1,7),(7,8),(8,10),(10,1)
Ruta 3 : (1,12),(12,5),(5,6),(6,7),('7...',1)
Ruta 4 : ('1...',12),(12,6),(12,7),(8,11),(11,5),('5...',1)
Ruta 5 : ('1...',2),(2,4),(2,9),(9,10),(11,9),('9...',1)
Ruta 6 : ('1...',5),(5,3),('3...',1)
Costo total: 394
```

En cuanto a tiempo de cómputo, el tiempo de ejecución ronda los 0.005 segundos, por lo cual se puede decir que en este ámbito este es bastante eficiente además de que este tiempo no se ve afectado por el tamaño del problema.

6. Conclusiones.

Habiendo evaluado el desempeño del algoritmo, vemos como la dependencia de como está configurado el problema puede ser una gran desventaja ya que, para problemas con condiciones muy similar se podría estar dando o no una buena solución por lo tanto, se recomienda optar por otras metodologías que contemplen mes sesgos.

En general por la condicion presente en los algoritmos greedy, de quedarse con soluciones localmente óptimas en cada paso del proceso de optimización sin considerar el impacto global de estas elecciones, a veces pueden producir soluciones de baja calidad. En el caso de problemas de ruteo sobre arcos capacitados, un algoritmo goloso podría producir rutas que no son las más eficientes en términos de la cantidad total de basura recolectada o la distancia recorrida, muestra de esto es el algoritmo goloso utilizado en este documento por lo tanto, a partir de esta experiencia y de lo expuesto en la literatura se concluye que se pueden obtener mejores resultados aplicando algoritmos o heurísticas contemplen el impacto global de las soluciones óptimas locales en la global estos pueden ser: Clarke and Wright, Sweep algorithm, Simulated Annealing, entre otros.

7. Referencias

[1] Golden, B.L. and Wong, R.T. (1981), Capacitated arc routing problems. Networks, 11: 305-315. https://doi.org/10.1002/net.3230110308.

[2] Benavent, Enrique & Campos, B. & Corberán, Ángel & Vidal, Enrique. (2011). The capacitated arc routing problem. A heuristic algorithm. QÜESTIIÓ (Quaderns d'Estadística i Investigació Operativa); Vol.: 14 Núm.: 1 -3. 14.

[3] Cerrone, Carmine & Cerulli, Raffaele & Golden, Bruce. (2017). Carousel Greedy: A Generalized Greedy Algorithm with Applications in Optimization. Computers & Operations Research. 85. 10.1016/j.cor.2017.03.016.