Aleatorización Sesgada

Alvarez Estarlich, Mauro
Requero Martín, David
Serrano Gómez, Enrique

December 6, 2020

1 Resumen y Comentario

1.1 On the use of Monte Carlo simulation, cache and splitting techniques to improve the Clarke and Wright Savings heuristics (Juan, A. et. al. (2011))

El transporte por carretera es el principal modo de transporte de bienes y se necesitan modelos y métodos eficientes para ayudar en los procesos de toma de decisiones de este campo.

El artículo habla sobre el Capacitated Vehicle Routing Problem (CVRP) del cual el objetivo principal es encontrar la solución (viable) de mínimo coste. Además, se introduce el algoritmo probabilístico SR-GCWS-CS, con el cual quieren probar como se pueden usar los métodos basados en simulación para mejorar heurísticos existentes.

El algoritmo SR-GCWS-CS

Combina la simulación de Monte Carlo (MCS) con el heurístico de ahorros Clarke and Wright (CWS). Además, utiliza técnicas de "Divide y vencerás" y memoria cache de soluciones obtenidas. De esta manera, puede proveer, de manera eficiente, un conjunto de soluciones pseudo-óptimas para CVRP.

Combinación de MCS con CWS

Este método asigna una probabilidad sesgada, con ruido, a cada arista en la lista de ahorros de CWS. Aristas con mayores ahorros tendrán más probabilidad de ser escogidas antes para combinarse.

Para cada ruta generada, el mejor orden de viaje conocido entre los nodos se guarda en una especie de memoria cache. Esto se usa para mejor la calidad de generación de futuras rutas.

El conjunto original de nodos se divide en subconjunto disjuntos y se resuelven de manera independiente para después generar la solución global.

Este método híbrido ofrece ventajas sobre otros metaheurísticos:

a) Simple.

b) Metodología flexible y robusta.

c) Buen rendimiento generando soluciones.

d) No requiere ajustes de alta precisión.

e) Se puede ejecutar en paralelo.

Valoración

Es un artículo bien estructurado ya que responden a la mayoría de las preguntas que alguien se podría hacer al leerlo: Qué problema hay, en que consiste, como se ha solucionado y que métodos se han utilizado anteriormente, como lo van a solucionar ellos y por qué lo hacen así, etc.

Aparte de la estructura, las explicaciones están bien hechas, dando contexto a cada nuevo concepto/término para utilizarlo más adelante. Además, da referencias para que el lector profundice si lo desea y es sencillo de entender. Hubiera sido interesante que cogieran una instancia pequeña como ejemplo y que mostraran como cambian las rutas en cada optimización añadida (CWS +MCS, CWS + MCS + Hash map, CWS + MCS + Hash map + Split).

1.2 The SR-GCWS hybrid algorithm for solving the capacitated vehicle routing problem. (Juan, A. et. al. (2010))

El artículo presenta un nuevo planteamiento para encontrar soluciones al "capacitated vehicle routing problem" (CVRP). Para ello introducen el concepto de Clarke and Wright Savings (CWS), que es usado para asignar una puntuación a cada arista en función del "ahorro" que supone seleccionarlo para moverse de un nodo a otro, siempre y cuando no se violen las reglas establecidas por el problema. También se añade la técnica de Monte Carlo simulation (MCS) ya que se ha visto que resulta muy útil a la hora de resolver problemas complejos mediante el uso de números aleatorios y distribuciones estadísticas.

Poniéndolo todo junto, para cada paso en el que se deba seleccionar una nueva arista, primero se seleccionan aquellas que tengan un mayor "ahorro" y a cada una de ellas se le asigna una probabilidad de ser seleccionada (mayor probabilidad a los que presenten un "ahorro" mayor). La probabilidad que se asignará vendrá determinada por una distribución (cuasi-) geométrica, también seleccionada de un conjunto de distribuciones posibles. En concreto, el punto clave es la generación de cientos de soluciones posibles, cada una de ellas generada a partir de una versión aleatorizada de la solución obtenida por CWS. Con todo ello, se consigue un algoritmo altamente explorativo que es capaz de mejorar los resultados obtenidos por los algoritmos más vanguardistas, dentro de un periodo de tiempo aceptable.

Valoración

El artículo plantea un algoritmo con claras ventajas frente a los algoritmos clásicos de optimización: es sencillo tanto en cuanto a la complejidad de programación como a las técnicas y conceptos que lo fundamentan, además no requiere de "tunning" de parámetros para obtener mejores resultados, por lo que se acorta el tiempo de experimentación para llegar a una solución subóptima pero aceptable cuando se compara con los mejores resultados obtenidos hasta la fecha. En cuanto a la redacción y la forma de presentar los diferentes conceptos, utilizando un lenguaje fácilmente comprensible, evitando un gran número de siglas y acrónimos, hace que la lectura no sea pesada y que el lector sea capaz de comprender los principales conceptos e ideas que se quieren transmitir. Queda por explorar el potencial de adaptación de esta solución para otros problemas de otros ámbitos.

1.3 Biased Randomization of Heuristics using Skewed Probability Distributions: a survey and some applications (Grasas, A. et. al. (2017))

La aleatorización sesgada (BRP) se platea como una solución al problema de obtener algoritmos más exploratorios que permitan obtener soluciones a lo largo de todo el dominio del problema, fuera del intervalo situado en torno a un óptimo local. Se puede implementar empleando funciones empíricas o a través del uso de uso de distribuciones de probabilidad sesgadas. El primer caso se basa en una función sesgada con un criterio que asigna un peso a cada candidato, a partir de los cuales se genera una distribución de probabilidad empírica. Sin embargo, el segundo caso presenta dos ventajas fundamentales: las distribuciones de probabilidad teóricas suelen ser función de un único parámetro fácilmente definible y se disponen de fórmulas analíticas para la obtención sus valores, que son más eficientes a nivel computacional al evitar el uso de bucles. La variación de dicho parámetro a partir de sus dos valores extremos permite obtener una selección más aleatoria o más sesgada. Empleando paralelización se pueden obtener numerosas soluciones candidato, lo cual resulta beneficioso para aplicaciones en las que le tiempo es un factor crítico. Se han implementado de forma experimental en cinco problemas de optimización con heurísticas constructivas (VRP, ARP, PFSP, UFLP, y 2DSPP). Se obtienen mejores resultados a mayor tiempo de ejecución y mayor número de agentes en paralelo. Además, se observa una mejora en los resultados frente a la heurística original, y en comparación con la aleatorización uniforme, donde se obtiene un resultado bastante deficiente en la mayoría de casos.

Valoración

Al tratar de evitar que los resultados obtenidos estén atrapados el entorno de un óptimo local, se corre el riesgo de que el proceso de selección aleatoria empleado descarte soluciones candidato plenamente válidas. El empleo de distribuciones estadísticas sesgadas se plantea como solución computacionalmente eficiente al combinar en una única función una fuente de aleatoriedad y de sesgo. Principalmente se han probado las distribuciones triangular y geométrica, aunque la gran variedad de distribuciones no simétricas permitirá implementar la opción más adecuada para el caso en cuestión, ya que las distintas distribuciones ofrecen distintos grados de asimetría y de dispersión de los valores. Es destable que este procedimiento se puede aplicar a heurísticas de problemas de muy diversa índole, en general con resultados óptimos. Por último, el empleo de técnicas de paralelismo con tal de aumentar su rendimiento permitirá aprovechar las últimas tecnologías de computación distribuida.

Diseño y desarrollo de un algoritmo de busqueda randomizada y heuristica CWS aplicada al VRP

1) **Introducción**

El problema del rutaje de vehiculos (VRP de sus siglas en inglés) se centra en la busqueda del mejor conjunto de rutas por las que guiar a una flota de vehiculos de mercancias cuyo deber es abastecer a los clientes. Pero el problema puede ser planteado con diferentes grados de complejidad en funcion de las restricciones y detalles que se quieran considerar con tal de asemejar el sistema cada vez mas al mundo real. Las principales restricciones que se aplican al VRP se basan en:

- El uso de vehiculos con una capacidad finita en cuanto al transporte de mercancias.
- Que los vehiculos no pueden visitar dos veces un mismo cliente en su ruta de reparto.
- La presencia de un almacen desde del que todos los vehiculos salen y al que deben volver tras el reparto.

El VRP tiene un gran interes tanto por los beneficios económicos que puede reportar, asi como por los beneficios ecológicos que se pueden deducir del mismo.

2) **Revisión de la literatura**

Para el desarrollo de nuestro algoritmo nos hemos basado en la revision de tres articulos: Juan, A. et. al. (2010) (1), Juan, A. et. al. (2011) (2) y Grasas, A. et. al. (2017) (3).
En el primer trabajo (1) demuestra como el uso de Clarke and Wright Savings (CWS) y simulaciones de Monte Carlo (MCS) favorecen la busqueda de soluciones con un menor "coste" y además se introduce el uso de funciones (cuasi-) geométricas para la asignación de probabilidades a los candidatos a mejor ruta. Con este enfoque se consigue que el mejor canidato pueda o no ser elegido, ya que se le asigna una probabilidad (mas elevada en funcion de la calidad de la solución) para que sea o no seleccionado.
En el segundo articulo (2) del mismo autor, a parte de la técnicas descritas anteriormente se añade el concepto de "splitting", basado en el principio de "divide y vencerás", por el cual se subdivide el conjunto de nodos en subgrupos para ser procesados de manera independiente y después unir las diferentes subsoluciones obtenidas en una solución única. Grácias a este planteamiento se consigue obtener soluciones donde las rutas se cruzan con menor frecuencia, es simple y permite ser ejecutado en paralelo.
En el articulo más reciente (3), se plantea el uso de la aleatorización sesgada como medida para obtener un algoritmo más explorativo, permitiendo obtener soluciones a lo largo de todo el dominio del problema. Para ello se plantean dos maneras de obtener distribuciones de probabilidad sesgadas, la primera basada en la obtención empirica y la segunda (y preferida) la basada en distribuciones teóricas generadas a partir de funciones matemáticas. Esta segunda opción es más rápida a nivel computacional y configurable a partir de los parametros que definen la función de distribución.

3) **Algoritmo desarrollado**

A partir del algoritmo *Clarke and Wright Savings* (CWS) hemos añadido 3 métodos que pueden ser independientes y combinados unos con otros, para mejorar la solución inicial que nos proporciona el algoritmo básico.

- Inicio sesgado.
- Método de aprendizaje utilizando una memoria de rutas.
- Método "Divide y vencerás".

El algoritmo básico tiene 3 fases:

1 Construir las aristas entre los nodos, calculando un coste asociado al movimiento entre nodos, este coste se usa como concepto de ahorro. Se genera una lista ordenada de las aristas que suponen mayor ahorro a menos ahorro.

2 Construir una solución inicial con rutas desde el nodo inicial al nodo destino, solución que no es eficiente, pero es el punto de partida para ejecutar el siguiente paso.

3 Se itera la lista de aristas creada y se comprueba si es factible fusionar las rutas que hay desde el nodo inicial a cada nodo en los extremos de la arista. Al fusionar las rutas se consigue ahorrar costes y se empieza intentando fusionar las rutas mas costosas (que producen más ahorro).

En este algoritmo básico aplicamos **inicio sesgado** a la lista de ahorro. Hay dos maneras de hacerlo:

1 El orden de la lista de ahorro es completamente aleatorio. Cada arista tiene la misma probabilidad de ser analizada y fusionada la ruta de sus nodos primero. A este caso se le puede limitar la aleatoriedad limitando cuantas aristas tiene en cuenta. Por ejemplo, la elección de aristas es completamente aleatoria pero solo entre las 10 aristas que suponen mayor ahorro.

2 El orden de la lista de ahorro es aleatorio con una probabilidad sesgada. Esto significa que las aristas con mayor ahorro tendrán mas probabilidad de ser elegidas primero. Podemos modificar cuanto sesgo queremos utilizar con un valor Beta. Utilizando una Beta pequeña el sesgo para elegir aristas de mayor ahorro es menor y con una Beta grande el sesgo es mayor y la elección de aristas se asemeja al algoritmo básico.

Nosotros hemos utilizado el segundo método basándonos en la literatura investigada. El código que nos modificaba la lista de ahorro según el valor Beta es el siguiente:

```
def generateBiasedSavingsList(self, beta):
    copySavings = self.savingsList.copy()
    biasedSavings = []
    for i in range( len(copySavings) ):
        index = int( math.log(random.random()) / math.log(1 - beta) )
        index = index % len(copySavings)
        biasedSavings.append( copySavings[index] )
        copySavings.pop( index )
    return biasedSavings
```

Como existe un componente aleatorio, la ejecución de este método ha de ser realizada varias veces para evitar que encontremos una anomalía estadística y que los resultados no sean fiables. Cuantas más veces iteramos las posibles anomalías desaparecen y tenemos un resultado más fiable que no depende de la "suerte" que tengamos con

las probabilidades.

El **método de aprendizaje** mediante la cache de rutas se basa en la idea de que en cada iteración se generan rutas distintas con diferente coste (debido al inicio sesgado que utilizamos).

Si mantenemos un mapa con las mejores rutas encontradas para determinado conjunto de nodos, podemos mejorar el coste de la solución final encontrada comparando las rutas de la solución con las mejores rutas encontradas.

Por ejemplo, quizás nuestra solución final ha encontrado una ruta que implica a los nodos 0,1,2,3,4 en el orden: 0-2-3-4-1. Como hemos guardado la mejor ruta encontrada para cada conjunto de nodos, comprobamos si hay alguna manera de recorrer esos nodos con menor coste, y encontramos que la ruta 0-3-2-1-4 es más "barata" de hacer. Así que sustituiríamos la ruta final de la solución con otra mejor.

El código con el que hacemos esta comprobación es el siguiente:

```
def improveSolutionWithBestRoutesFound(self):
    for route in self.sol.routes:
        routeKey = self.getRouteHash( route )

        if routeKey in self.bestRoutes.keys():
            if self.bestRoutes[routeKey].cost < route.cost:
                self.sol.cost -= route.cost
                self.sol.cost += self.bestRoutes[routeKey].cost
                route = self.bestRoutes[routeKey]
```

Y la actualización del mapa con rutas mejores así:

```
#populate bestRoutes
if( self.enabledRouteCacheUsage ):
    #create hash to search for the route
    routeKey = self.getRouteHash( iRoute )

    if routeKey in self.bestRoutes.keys():
        if self.bestRoutes[routeKey].cost > iRoute.cost:
            self.bestRoutes[routeKey] = copy.deepcopy( iRoute )
    else:
        self.bestRoutes[routeKey] =  copy.deepcopy( iRoute )
```

El último método que hemos aplicado es el de **"Divide y vencerás"**. Este método se basa en la división del problema inicial en subproblemas más pequeños de manera que se pueden calcular soluciones independientes que pueden mejorar las rutas encontradas. Estas soluciones una vez encontradas son juntadas y nos otorgan una solución final al problema.

En nuestro caso, la división del problema se hace mediante la localización de los nodos en el espacio. Cada nodo tiene unas coordenadas X e Y de manera que dividimos los nodos por su posición en el mapa.

Hemos utilizado cuatro maneras diferentes para dividir los nodos:

1 Nodos por encima y por debajo de la coordenada Y(0). Genera 2 subgrupos de nodos.
2 Nodos a la derecha o izquierda de la coordenada X(0). Genera 2 subgrupos de nodos.
3 Nodos en cada uno de los 4 cuadrantes del eje de coordenadas. Genera 4 subgrupos de nodos.
4 Nodos en cada uno de los 4 cuadrantes divididos por la diagonal. Genera 8 subgrupos de nodos.

Este método puede funcionar mejor o peor dependiendo de las características de la red. Por ejemplo, si todos los nodos están por debajo de Y(0) y utilizamos el primer enfoque, solo generaremos un subgrupo de nodos.
Además, hay que tener en cuenta que no siempre la división del problema nos otorgara mejores soluciones.

El código para dividir el mapa de nodos y encontrar soluciones independientes es el siguiente:

```
splittingTypes = ['TopBottom', 'LeftRight', "Cross", "Star"]
if splittingType not in splittingTypes:
    raise ValueError("Invalid sim type. Expected one of: %s" % splittingTypes)
if splittingType == "TopBottom":
    splt_Nodes = self.splitTopBottomNodes()
if splittingType == "LeftRight":
    splt_Nodes = self.splitLeftRightNodes()
if splittingType == "Cross":
    splt_Nodes = self.splitCrossNodes()
if splittingType == "Star": #8 cuadrants
    splt_Nodes = self.splitStarNodes()

self.sol = Solution()
for splt_node in splt_Nodes:
    self.bestRoutes = {}
    self.constructEdges(splt_node)
    self.constructDummySolution(splt_node)
    biasedList = self.generateBiasedSavingsList(beta)
    self.edgeSelectionRoutingMerging(biasedList)

    if( self.enabledRouteCacheUsage ):
        self.improveSolutionWithBestRoutesFound()
```

Por último, estos 3 métodos pueden combinarse entre ellos para conseguir una mejor solución explotando las posibilidades que nos ofrece cada uno. Hemos hecho diversas pruebas combinando los 3 métodos que aparecen en el siguiente apartado.

4) **Experimento computacional y análisis de resultados**

5) **Conclusiones y trabajo futuro**