Inteligencia Artificial

Estado del Arte: Car Sequencing Problem

Andrés Wulff Reyes

29 de Noviembre de 2021

Evaluación

Mejoras 1ra Entrega (10 %):	_____
Código Fuente (25 %):	_____
Representación (25 %):	_____
Descripción del algoritmo (35 %):	_____
Bibliografía (5 %):	_____
Nota Final (100):	_____

Resumen

En este documento se da una vista introductoria al ***Car Sequencing Problem***, problema que busca determinar una secuencia de automóviles para una linea de ensamblaje con tal de cumplir con la demanda de diversos tipos de vehículos, e intentando minimizar la penalización asociada a sobrepasar la capacidad máxima de las estaciones que instalan las diversas opciones que estos vehículos pueden requerir. Se suelen utilizar diversos métodos para solucionar este problema, tanto exactos como heurísticos, aunque dada la efectividad de estos últimos para encontrar soluciones en cortos periodos de tiempo, se suele favorecer su aplicación. Además se presentan una representación y algoritmo a utilizar para posterior solución de diversas instancias del problema con *Hill Climbing*.

1. Introducción

El objetivo de este documento es presentar el contexto del ***Car Sequencing Problem***, problema cuyo origen proviene de la necesidad por establecer una secuencia óptima de autos en una linea de ensamblaje, la cual cuenta con múltiples estaciones que se encargan de añadir piezas especificas a los autos que pasan por cada una de estas, y las cuales cuentan con una capacidad limitada de autos en los que pueden trabajar a la vez. Definir correctamente esta secuencia implica que el curso de los automóviles a lo largo de la linea de ensamblaje sea mas fluida, y por lo tanto que el trabajo sea mas eficiente, lo cual resulta en un ahorro de tiempo, que en consecuencia implica un ahorro de recursos y gastos en general.
Dado el motivo previamente mencionado es que se ha de presentar en este documento la definición del problema en mas detalle, seguido del estado de arte, identificando y describiendo las múltiples formas en que se ha propuesto solucionar este problema en la contemporaneidad, definiendo posteriormente un modelo matemático base que considerar al momento de diseñar una solución propia, y finalizando con una representación y definición del algoritmo a utilizar para la resolución de diversas instancias del problema, basado en el método *Hill Climbing*.

2. Definición del Problema

2.1. Descripción

El *Car Sequencing Problem* es originalmente definido por Parrello et al. en 1986[4]. El contexto del problema es una linea de ensamblaje de automóviles, por la cual deben pasar un numero definido de autos que requieren cada uno un conjunto definido de opciones, las cuales son instaladas en estaciones especializadas en cada una de las piezas a instalar, y que cuentan con una capacidad máxima de automóviles en los que pueden trabajar a la vez, restricción la cual se representa por razones p/q asociadas a cada estación, siendo p el numero máximo de autos que requieran de la opción correspondiente a la estación en cada subsecuencia q de autos. Es posible superar este máximo definido, aunque hacerlo implica una penalización presente en la función objetivo.
Los parámetros de este problema, como es definido por Solnon et al.[6], se presentan en la tupla (V, O, p, q, r), dentro de la cual V refiere al conjunto total de automóviles, los cuales a su vez se ven categorizados en V_k donde k es el numero de distintos tipos de autos que requieren un conjunto definido de opciones; O representa el conjunto de opciones disponibles; p y q representan, para cada opción, la cantidad máxima óptima p_i de autos que requieran esta opción dentro de cada subsecuencia de largo q_i; y r representa, con la denominación r_{ij}, si es que un determinado auto v_j requiere de una opción o_i.
Las variables a trabajar son x_{ij}, las cuales representan de forma binaria si un respectivo auto v_j se encuentra en la posición i de la secuencia, y diversas variables propias de los métodos de resolución relacionadas al sistema de penalización implementado por no respetar las restricciones de capacidad. Al buscar una solución para este problema, lo ideal es encontrar una secuencia de automóviles tal que se cumplan todas las restricciones de capacidad definidas para cada opción. Ahora bien, en caso de no encontrar una solución que cumpla con esto, el objetivo es encontrar una secuencia de automóviles tal que se rompan la menor cantidad de restricciones, para lo cual se consideran múltiples diversos factores según ciertas variantes del problema o formas de abordar su solución.

2.2. Problemas Relacionados

El problema en cuestión, como es indicado por Parrello et al., es una versión especifica del **Job-shop Scheduling Problem**, el cual consta de asignar un orden de procesamiento óptimo para un conjunto de tareas con tiempos de computo diferentes, las cuales se deben asignar a maquinas con diverso poder de procesamiento, y con restricciones propias de precedencia, lo que refiere a que cada tarea requiere que otra tarea previa especifica haya sido ya realizada antes de comenzar. El objetivo de este problema es minimizar el tiempo total que toma realizar todas las tareas considerando los factores previamente mencionados, lo cual tiene una clara similitud al *Car Sequencing Problem*, cuyo objetivo aunque representado de distinta forma, también implica en términos prácticos un ahorro en tiempo por medio de la definición de una secuencia de elementos.

2.3. Variantes del Problema

Una de las variantes mencionadas por Solnon et al. en su vista general del estado del arte del *Car Sequencing Problem* es una originada por las necesidades de RENAULT, denominada en el mismo documento como **ROADEF'2005 Car Sequencing Problem**, donde se considera también una linea de pintura, en la cual se busca minimizar la cantidad de pintura a utilizar por medio de agrupar los autos que requieren de un mismo color, pero también considerando una restricción dura referente a la cantidad total de pintura que se puede utilizar, la cual agrega una restricción inviolable al problema.

3. Estado del Arte

El *Car Sequencing Problem*, como ha sido previamente mencionado, es originalmente definido en 1986 por Parrello et al., donde identifican la situación dada en la industria de la producción de automóviles, donde se requiere de armar autos heterogéneos que requieren de piezas distintas. A continuación se han de presentar las múltiples formas de representar el problema, solucionarlo, y los resultados de estas formas de abordar el problema con respecto a las instancias presentes en CSPLib[1] y a las definidas por Smith[5]. Estas han sido extraídas del paper ya previamente mencionado por Solnon et al., y de uno realizado por Oliveira dos Reis[2].

3.1. Representación y Métodos de Solución: Métodos Exactos

3.1.1. Programación de Restricciones

Este método según mencionan Solnon et al.[6], consiste en aprovechar los lenguajes de resolución de problemas de restricciones solamente definiendo las variables y restricciones a considerar. Para esto se definen como variables X_i que representa el tipo de auto a asignar en la posición i, O_{ij} que representa si es que una opción j se debe instalar en el auto asignado a i, y tres restricciones las cuales determinan, correspondientemente, que opciones deben instalarse en una posición según el auto asignado; que se cumplan las limitantes de capacidad; y por ultimo, que se cumpla con la demanda necesaria.
Este método resulta ser bastante fácil de implementar y útil para resolver instancias pequeñas del problema, pero dado que usa algoritmos genéricos de resolución de problemas de restricciones, en instancias grandes no es capaz de competir con otros métodos de resolución.

3.1.2. Programación Entera

Este método según mencionan Solnon et al.[6], consiste en utilizar variables binarias que representan si un tipo de auto ha sido asignado a una posición especifica. De esta forma se cuenta con un numero de variables equivalente a la demanda total de autos multiplicada por la cantidad de clases. Por medio de restricciones lineales es que se asegura que solo un tipo de auto sea asignado a cada posición, y que se cumplan las demandas definidas.
Este método ha logrado encontrar soluciones para todas las instancias del problema en CSPLib y a cuatro de las instancias planteadas por Smith, sin encontrar soluciones óptimas para otras cinco que este presenta.

3.1.3. Método Ad-hoc

Este método según mencionan Solnon et al.[6], utiliza una matriz ixj en la cual si una casilla cuenta con un valor 0 entonces en la posición i de la secuencia se instalara la opción j; en caso de ser 1 entonces es posible instalar la opción en la posición determinada; o en caso de ser -1, entonces no se puede instalar la opción en tal posición. Dada esta matriz se aplica el método *Branch and Bound*, donde a cada nodo del árbol corresponde una matriz que representa el estado de secuenciación de tal nodo.

3.2. Representación y Métodos de Solución: Métodos Heurísticos

3.2.1. Algoritmo Voraz

En este método se comienza con una secuencia de autos vacía, y dadas diversas posibles heurísticas voraces, se elige un tipo de auto para asignar al final de la secuencia. La heurística de

[1] CSPLib:
`https://www.csplib.org/Problems/prob001/`

mayor rendimiento identificada por Gottlieb et al. el 2003[3], consta de asignar el automóvil que maximice la suma de las tasas de utilización de opciones requeridas, las cuales son actualizadas después de asignar un vehículo.
Este método logra solucionar todas las instancias presentes CSPLib a gran velocidad.

3.2.2. Búsqueda Local

Este método según mencionan Solnon et al.[6], consta de, a partir de una solución inicial definida usualmente aleatoria, realizar cambios a esta secuencia con tal de explorar el vecindario de soluciones que lo rodean, buscando así una solución local óptima. Distintos tipos de movimientos se pueden realizar según se desee abordar el problema, generando distintos tipos de vecindarios.
Este método a probado ser bastante efectivo en encontrar soluciones óptimas para las instancias de CSPLib.

3.2.3. Algoritmos Genéticos

Este método consta de aplicar operaciones de cruce y mutación en una población de secuencias de automóviles, para generar nuevas generaciones de secuencias, las cuales, en caso de no cumplir con las restricciones, son reparadas con algoritmos voraces, y optimizadas con *Hill-Climbing*.
Este método no ha resultado ser muy efectivo, dado que solo logra resolver instancias fáciles del problema.

3.2.4. ACO: Optimización de Colonia de Hormigas

Este método consta de utilizar un sistema de "hormigas artificiales", las cuales dejan un rastro de feromonas en las soluciones que consideran óptimas, dejando así ver a otras hormigas cuales son las subsecuencias mas prometedoras. Este método se ve mejorado por Gottlieb et al. aplicando heurísticas voraces.
Resultados experimentales indican que este método resulta ser mas efectivo que el método de búsqueda local en instancias pequeñas del problema, y que son prácticamente equivalentes en instancias de mayor envergadura.

3.2.5. MEAs: Algoritmos Evolutivos Multiobjetivo

Un método derivado de los algoritmos genéticos, que en la implementación presentada por Oliviera dos Reis[2], consta de dos etapas, siendo estas la mutación y la selección. El autor indica que prefiere hacer énfasis en la etapa de mutación, ignorando la etapa de recombinación en la que se suelen enfocar los algoritmos genéticos implementados para este problema, con tal de reducir el tiempo de computo necesario. Para esto se reduce la aleatoriedad del proceso de mutación, utilizando estrategias basadas en el vecindario.
Los resultados de este método han sido comparados con otros métodos exactos aplicados al mismo problema, habiendo una diferencia de varios ordenes de magnitud en el tiempo de computo, siendo el método heurístico aplicado mucho mas efectivo en su ejecución.

3.2.6. GRASP

Este método, presentando por Bautista et al.[1], aplica métodos voraces en la etapa de exploración para encontrar soluciones iniciales factibles, para posteriormente explotar esta solución buscando una solución óptima en el vecindario de esta.
Los resultados obtenidos indican que en la gran mayoría de los casos se logro cumplir con las restricciones duras de las instancias del problema, aunque no en todas, y que los resultados obtenidos son aceptables con respecto a las restricciones suaves.

3.3. Resultados

Los resultados recopilados por Solnon et al. en el desafió ROADEF'2005 indican que los algoritmos de **búsqueda local** han resultado ser bastante efectivos en resolver el *Car Sequencing Problem*, y en general los métodos heurísticos demuestran una tendencia a resolver de forma mas eficiente y rápida este problema en comparación a los métodos exactos presentados.

4. Modelo Matemático

Para el modelo que se ha de trabajar se considerara la estructura presentada en el video introductorio del trabajo asignado[2]. Este considera la siguiente definición:

4.1. Parámetros

Los parámetros generales son los siguientes:

- D: Demanda total de vehículos. Este parámetro representa el largo total de la secuencia a definir.
- $O = o_1, ..., o_m$: Conjunto de opciones (partes) que pueden requerir los autos.
- $K = k_1, ..., k_n$: Conjunto de clases de automóviles. Cada clase refiere a una configuración especifica de opciones que cada automóvil perteneciente a esta clase requiere instalar. En las definiciones de las instancias a trabajar con el algoritmo se consideran las clases como comenzando desde un índice equivalente a 0, pero para su definición teórica consideraremos sumar 1 a estos índices para simplificar ciertos aspectos, por lo que el índice inicial de las clases sera teóricamente 1.
- d_k: Demanda total para cada clase $k \in 1, ..., n$.
- $c_o : O \to \mathbb{N}; l_o : O \to \mathbb{N}$: Estos valores representan, para una opción o_i, la cantidad máxima óptima c_o de autos que requieran esta opción dentro de cada subsecuencia de largo l_o.
- $r_{ij} : KxO \to 0, 1$: El valor r_{ij} representa de forma binaria si es que el auto de tipo k_j requiere de la instalación de la opción $o_i \in O$. Si r_{ij} es 1, entonces el auto requiere de esta opción. En caso contrario ($r_{ij} = 0$), el auto no requiere la opción.

4.2. Variables

Las variables de decisión a considerar son las siguientes:

- X_i : Variable que refiere al tipo de automóvil que se asigna a la posición i. El dominio de esta variable es el conjunto K.

4.3. Espacio de Búsqueda

Dado que lo que se busca es definir una secuencia de automóviles, el espacio de búsqueda corresponde a todas las posibles secuencias diferentes entre si que se pueden formar. Para esto, considerando las demandas totales para cada clase, y la demanda total de la secuencia, el espacio de búsqueda se representa por la siguiente expresión extraída de Solnon et al., ajustada al modelo presente:

$$\frac{D!}{d_1! \cdot ... \cdot d_n!}$$

[2] *Car Sequencing Problem - Inteligencia Artificial* por Diego Norambuena
`https://www.youtube.com/watch?v=d8TmvcI09m4&ab_channel=DiegoNorambuena`

4.4. Restricciones

Las restricciones a considerar se categorizan en restricciones duras y suaves. Estas son las siguientes:

- Restricción dura: Se debe cumplir exactamente con la demanda total de vehículos. Se puede expresar de la siguiente forma:

$$\prod_{i=1}^{D} X_i! = 0$$

  Considerando que las clases de vehículos se vean representadas por números mayores a 0, debido a lo previamente mencionado de establecer el índice inicial en 1 al trabajar teóricamente, y que un valor equivalente a 0 indica que no se ha asignado una clase de vehículo en cierta posición, entonces siempre que la expresión previa sea distinta a 0, se cumple con la demanda total.

- Restricción dura: Se debe cumplir exactamente con la demanda parcial asociada a cada una de las clases de automóviles. Para comprobar esto se define una función $g(x, k)$ de la siguiente forma:

$$g(x,k) = \begin{cases} 1; x = k \\ 0; x \neq k \end{cases}$$

  Y una función $j(k)$ tal que:

$$j(k) = \begin{cases} 1; \sum_{x=1}^{D} = d_k \\ 0; \sum_{x=1}^{D} \neq d_k \end{cases}$$

  Con lo cual se puede utilizar la siguiente expresión:

$$\sum_{k=1}^{n} j(k) = n$$

  Esta expresión refiere a que para cada clase, si es que la sumatoria de las funciones $g(x, k)$ para todos los vehículos asignados es equivalente a la demanda de esta clase, la función $j(k)$ para cada clase equivaldrá a 1, y la suma de estas deberá de ser equivalente al numero de clases. En caso de no ser equivalente al numero de clases, no se cumple con la restricción.

- Restricción dura: En cada posición dentro de la secuencia determinada solo deber haber un automóvil. Esta restricción se encuentra incluida en la representación de la solución, dado que en cada elemento dentro del arreglo referente a la secuencia asignada solo se puede asignar un valor perteneciente al dominio de clases de vehículos disponibles.

- Restricción suave: Cada subsecuencia de vehículos debe cumplir con las limitantes de capacidad de cada estación dentro de lo posible. Esta restricción se aborda principalmente en la función objetivo, a ver en la sección posterior.

4.5. Función Objetivo

Consideramos que para cada subsecuencia de largo l_i, dada una opción o_i, en caso de exceder la capacidad definida c_i, se suma un punto a la función objetivo por cada automóvil de exceso. De esta forma lo que buscamos es minimizar la suma total de vehículos que exceden la capacidad de cada subsecuencia para cada opción, representando esto con la siguiente expresión:

$$minimize \sum_{o=1}^{m} \sum_{j=1}^{D-l_o+1} \frac{((\sum_{i=j}^{j+l_o} r_{oX_i}) - c_o) + |((\sum_{i=j}^{j+l_o} r_{oX_i}) - c_o)|}{2}$$

Donde r_{oX_i} representa si es que el tipo de automóvil X_i asignado requiere de la opción o. Para explicar la expresión presentada, esta sigue la siguiente lista de pasos:

- Se itera para cada opción disponible.
- Para cada opción se itera nuevamente en cada subsecuencia posible según los parámetros de capacidad de la opción.
- Para cada subsecuencia se sigue el siguiente proceso:
  - Se suman la cantidad de vehículos en la subsecuencia que requieran la opción determinada.
  - Al resultado se resta el valor correspondiente a la capacidad máxima de la opción (c_o).
  - Para evitar valores negativos en el caso de contar con menos vehículos que requieran la opción que la capacidad máxima, se suma a esto su valor absoluto.
  - El resultado se divide en 2 dado que en caso de sobrepasar la capacidad máxima, sumar el valor absoluto implica duplicar el numero de puntos sumados.

5. Representación

La representación de la solución a utilizar es de un arreglo unidimensional, de largo equivalente al parámetro D (demanda total de automóviles). Dentro de este arreglo, cada uno de sus elementos contendrá un numero entero referente a la clase del vehículo asignado. La razón de utilizar esta representación es para facilitar la verificación de las restricciones de demanda por clase, y para identificar con mayor facilidad el vecindario de la solución.
Aparte se han de mantener en memoria los siguientes parámetros, dentro de un objeto denominado *Parameters*:

- Numero de autos (D): Se ha de almacenar como un **numero entero**.
- Numero de opciones ($|O|$): Se ha de almacenar como un **numero entero**.
- Numero de clases ($|K|$): Se ha de almacenar como un **numero entero**.
- Parámetros c_o y l_o: Dado que cada opción cuenta con un c_o y l_o asignados, se almacenaran **dos arreglos**, uno para cada parámetro. Utilizando los métodos del objeto de clase *Parameters* se pueden identificar estos parámetros, utilizando como argumento el numero de la opción correspondiente.
- Opciones: Se almacena una matriz bidimensional en la forma de un vector de vectores de números binarios. Esta representación permite almacenar un arreglo cuyos elementos corresponden a cada una de las clases, representadas por arreglos correspondientes en los cuales sus elementos refieren, de forma binaria, a si es que los vehículos de la clase requieren la opción del índice respectivo.
- Demanda por clase: Se almacena un arreglo unidimensional de números enteros, donde cada elementos corresponde a la cantidad de vehículos que se requieren para la clase correspondiente al índice del arreglo.

6. Descripción del algoritmo

El método asignado para la solución de este problema ha sido **Hill Climbing** con **alguna mejora**. Dado esto, y la representación seleccionada para el problema, el algoritmo comenzara por inicializar el problema con una solución aleatoria en forma de un arreglo de largo D, en el cual cada elemento tendrá ya una clase de vehículo asignado, y obligatoriamente cumpliendo con la restricción de demanda por clase, osea, no asignando mas o menos vehículos de una clase que los que esta específicamente requiere según los parámetros presentados. Para esto se genera una variable de tipo *vector<int>* en el programa, a la cual se agregan, de forma iterativa por clase, el índice de la clase un numero de veces equivalente a la demanda especifica de la clase. Hecho esto con todas las clases, se desordena el vector resultante utilizando funciones de aleatorización, con lo cual obtenemos un arreglo que satisface las restricciones de demanda total y parcial.
Para explorar el espacio de búsqueda se utilizara un movimiento *swap*, cambiando la posición de los vehículos ya asignados. De esta manera se seguirá cumpliendo siempre con las restricciones de demanda total y por clases. Dado el paso inicial y el movimiento mencionado, se calcula el valor de la función objetivo con esta solución, y el algoritmo aplicara la siguiente lista de pasos de forma iterativa:

- Se intercambia la posición de dos valores dentro del arreglo.
- Se calcula el valor de la función objetivo con la secuencia obtenida.
- En caso de que este valor no sea menor al actual, se restaura la secuencia de asignación y se vuelve al primer paso, probando con un valor distinto.
- Dado el paso previamente mencionado, si no quedan mas operaciones de *swap* posibles, se considera que se ha llegado a un óptimo local, y se detiene el algoritmo.
- En caso contrario, que el valor si sea menor, se realiza una nueva iteración a partir de la nueva secuencia definida.

Adjunto a este documento se encuentra un programa escrito en el lenguaje *C++*, cuyas instrucciones de ejecución se encuentran en un archivo README.txt, y cuya principal función es leer los parámetros de una instancia del problema definida en un archivo .txt según el formato presentado en CSPLib[3], y mostrando posteriormente en la terminal de ejecución la solución inicial aleatoria definida a partir de estos parámetros.

Referencias

[1] Joaquín Bautista, Jordi Pereira, and Belarmino Adenso-Díaz. A grasp approach for the extended car sequencing problem, Nov 2007.

[2] Ricardo José de Oliveira dos Reis. Solving the car sequencing problem from a multiobjective perspective.

[3] Jens Gottlieb, Markus Puchta, and Christine Solnon. A study of greedy, local search, and ant colony optimization approaches for car sequencing problems, 2003.

[4] BruceD. Parrello, WaldoC. Kabat, and L. Wos. Job-shop scheduling using automated reasoning: A case study of the car-sequencing problem, 1986.

[5] Barbara M. Smith. Succeed-first or fail-first: A case study in variable and value ordering, 1996.

[3] CSPLib: `https://www.csplib.org/Problems/prob001/`

[6] Christine Solnon, Van Dat Cung, Alain Nguyen, and Christian Artigues. The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the roadef'2005 challenge problem, Dec 2008.