Inteligencia Artificial

Informe final: Car Sequencing Problem

Marcela Vidal Ramírez

1 de julio de 2014

Evaluación

Resumen (5 %):	_____
Introducción (5 %):	_____
Definición del Problema (10 %):	_____
Estado del Arte (35 %):	_____
Modelo Matemático (20 %):	_____
Conclusiones (20 %):	_____
Bibliografía (5 %):	_____
Nota Final (100 %):	_____

Resumen

En el presente documento se muestra al lector una indagación en la literatura científica sobre una variante del modelamiento de secuenciación en líneas de montaje de modelos mixtos, llamada Car Sequencing Problem, utilizada en sistemas reales de fabricación de vehículos, y se dan a conocer algunos enfoques existentes para su resolución, desarrollando específicamente una heurística Greedy más una de búsqueda local. Su importancia radica en que la solución de este problema se encarga de buscar una secuencia de fabricación de automóviles que optimice el proceso productivo, de tal forma que se respeten ciertas reglas referente al número de veces que cada producto puede ser asignado en tramos de este ordenamiento, y así, optimizar el flujo de materiales y el ensamblado de las unidades sin sobrecargar las áreas de trabajo.

1. Introducción

Considerando los avances de la industria automotriz, es posible afirmar que ésta se mantiene a la vanguardia de la tecnología. Su dinamismo ha cobrado mayor importancia en el último tiempo dado a ciertos factores, como por ejemplo, su esfuerzo continuo para reducir costos, y el aumento en la flexibilidad que requieren actualmente las plantas de fabricación debido a la creciente tendencia de la personalización en masa.

Inicialmente, las líneas de montaje fueron diseñadas para llevar a cabo una producción eficiente donde los costos que se consideraban eran para productos estándar. Sin embargo, hoy en día hay una gran variedad de opciones dentro de las cuales lo clientes pueden seleccionar para

personalizar las unidades a confeccionar, lo que ha ocasionado que los fabricantes requieran manejar una mayor cantidad de materias primas. Así mismo, con la finalidad de disponer de una amplia gama diversificada de productos y aprovechando los beneficios que un flujo eficiente de producción puede proporcionar, se apoyan en las llamadas líneas de montaje de modelos mixtos (MMAL).

Una variante de este tipo de metodología es el Car sequencing problem (Secuenciación de coches) definido inicialmente en 1986 por Parello, Kabat y Wos [1]. Se clasifica como un problema de satisfacción de restricciones en el cual, se debe definir el orden de entrada de un conjunto de automóviles de distintas clases a una línea de montaje, con tal de buscar una secuencia de fabricación que optimice el proceso productivo y así, cumplir con los requerimientos de los pedidos que recibe la planta.

Esta publicación tiene como objetivo dar a conocer las investigaciones que se han realizado en torno al Car Sequencing Problem, cuáles han sido los métodos para enfrentar el problema y los resultados que éstos han entregado. Además, se planteará un modelamiento, para posteriormente, analizar y comparar los resultados obtenidos y establecer conclusiones.

Con el fin de lograr esto, se presentará inicialmente una descripción más detallada del problema junto a los métodos y representación utilizados para resolverlo, que se encuentran en la literatura. La sección siguiente presenta el modelo matemático con sus respectivos parámetros, restricciones y variables. Finalmente se ofrecerá al lector las conclusiones de la investigación y referencias utilizadas para el desarrollo de esta publicación.

2. Definición del Problema

Considerando que los costes de los tiempo de producción dependen directamente de la forma en que se gestionan los recursos que una empresa requiere, el área encargada de su manejo, generalmente conocida como Dirección de Operaciones, tendrá como objetivo fundamental el producir un bien específico a tiempo y coste mínimo. Para ello, pueden apoyarse en las denominadas líneas de montaje de productos mixtos (MML) que permiten manufacturar productos, ya sea si tienen pocas o muchas variantes, evitando inventarios y tiempos de preparación importantes. Es por esto mismo que las líneas de montaje se han convertido en un componente fundamental de los sistemas de manufactura repetitiva moderna, específicamente para los sistemas de producción Just in Time (JIT).

En la filosofía de producción JIT, el objetivo principal es reducir al mínimo los niveles de stocks en el sistema, a través del manejo y variación de las tasas de producción y consumo en la cadena de fabricación y de suministro. Por tanto, existirán diversas medidas de eficiencia que dependerán de la política de la empresa, y bajo esta perspectiva, es posible identificar tres categorías de problemas de secuencias [2].

- **Programación en niveles:** Secuencias que implican tasas de producción y consumo de materiales, de manera de que permanezcan lo más regulares en el transcurso del tiempo.
- **Secuenciación de modelos mixtos:** Secuencias que buscan completar el máximo trabajo requerido por el programa de producción.
- **Secuenciación de vehículos: (Car Sequencing Problem)** Secuencias que presentan límites sobre la frecuencia con que pueden aparecer en ellas algunas opciones especiales.

En este documento se trata el Car sequencing problem, cuya finalidad es encontrar la disposición óptima de automóviles a lo largo de una línea de producción, los cuales se encuentran posicionados sobre cintas transportadoras que se mueven a través de diferentes áreas o estaciones de trabajo. Aún cuando los automóviles son similares entre sí, cada uno requiere componentes particulares que serán ensamblados en las distintas estaciones a lo largo de la línea de producción.

Por otro lado, una de las restricciones principales del problema está asociada con la escasez de recursos y la respectiva capacidad de cada zona de producción y por tanto, éstas deben ser consideradas en la secuenciación de tal forma de suavizar la carga de trabajo para asegurar una salida adecuada.

En base a lo anterior, se debe tener en consideración que los automóviles que requieren cierta opción o requerimiento, no deben ser agrupados juntos, de lo contrario, las estaciones no serían capaces de proceder ante la situación. Por ejemplo, si una estación en particular sólo puede hacer frente como máximo, a la mitad de los automóviles que pasan a lo largo de la línea, la secuencia debe ser construida de manera que a lo sumo una unidad de cada dos, requiera esa opción. Es por lo mismo, que en su versión original, el Car Sequencing Problem es un problema de viabilidad, puesto que el objetivo es encontrar la secuencia de vehículos (si ésta existe) que satisfaga todas las limitaciones de carga máxima.

La siguiente figura, basada en la formulación del problema según Dincbas, Simonis y Van Hentenryck [3], es un ejemplo visual de como se entiende el problema:

		Class											
Option	*Constraint*	1	2	3	4	5	6	7	8	9	10	11	12
1	1/2		•	•				•	•	•			•
2	2/3	•		•	•	•		•				•	•
3	1/3		•					•		•	•	•	
4	2/5				•		•		•				•
5	1/5		•			•							
	no. of cars	3	1	2	4	3	3	2	1	1	2	2	1

Figura 1: Problema de ejemplo formulado por Dincbas, Simonis y Van Hentenryck [3] donde se requiere ensamblar 25 automóviles en una línea de producción, la cual consta de 5 opciones. En la figura, el símbolo • indica que la clase de automóvil en cuestión, requiere la opción.

Los principales elementos son:

- **Opciones en la línea de ensamblaje:** representan los requerimientos de la unidad que se está produciendo, como por ejemplo: radio, protecciones laterales, aire acondicionado, etc.
- **Restricciones de dispersión:** representadas como $Ho : No$ Esto señala que entre los N puestos posteriores de la secuencia, solo es posible como máximo Ho ocurrencias de cierta opción o. De esta forma, si se encuentra una secuencia que no viola estas restricciones, la sobrecarga de trabajo se puede evitar, o disminuir, ya que la sobrecarga de trabajo es menor cuando menos reglas son violadas.

- **Clases:** los automóviles son agrupados en clases de tal forma que todas las unidades (vehículos) que la conforman requerirán las mismas opciones.

Los parámetros del problema son la cantidad de automóviles que se espera salgan de la línea de producción, y la cantidad de clases. De esta forma, si la demanda de ensamblaje es D automóviles y se cuenta con P clases, se requieren x_i variables con $i = \{1, ..., D\}$ correspondientes a las casillas de la secuencia, a las que se les asignará una clase.

A esto se suma la variable binaria C_{ji} para identificar si la unidad i tendrá la opción j. Donde $i = \{1, ..., D\}$ y $j = \{1, ..., P\}$.

Respecto a las restricciones se puede afirmar que:

- La cantidad total de casillas que tiene un mismo número de clase, es a lo más, igual a la cantidad de unidades que tiene dicha clase.
- Si la variable x_i, toma cierto valor m (que representa a una clase), entonces la variable C_{ji} tomará el valor 1 siempre que la clase m requiera la opción j,o será 0 en caso contrario. Así es posible relacionar las dos variables que modelan el problema.

3. Estado del Arte

El Car sequencing problem es un problema de decisión que se introdujo por primera vez por Parello, Kabat y Wos [1] en 1986 y consiste en el ordenamiento de un conjunto de automóviles a lo largo de una línea de montaje con el fin de instalar ciertos componentes, buscando satisfacer las limitaciones de capacidad de las distintas estaciones por las que deben pasar. En una primera versión, las restricciones fueron impuestas únicamente por la línea de montaje, sin embargo, con el tiempo, se han presentado desafíos más complejos, pues se añaden otras limitaciones como el color de los vehículos.

El problema ha sido estudiado por un gran número de investigadores, que han desarrollado una diversa gama de enfoques, tantos procedimientos exactos, como procedimientos heurísticos para obtener una solución.

A continuación se resumen algunas implementaciones y criterios que se han considerado en la literatura para resolver el problema.

3.1. Procedimientos Exactos

Se basan principalmente en la programación de satisfacción de restricciones y la programación lineal. Inicialmente se debe determinar un modelamiento adecuado y para ello, las restricciones deben ser definidas de tal forma que solo un automóvil es asignado a una posición de la secuencia, y por otro lado, todos las unidades deben ser utilizadas en la solución [6].

En el caso en que el problema se modele bajo una metodología de optimización combinatoria, las restricciones blandas pueden ser violadas, pero es posible solucionar aquellos agregando una función objetivo que añade un factor de coste. De esta forma, la solución contendrá una secuencia de automóviles, donde todas las restricciones fuertes son satisfechas y la función objetivo es minimizada.

La programación de satisfacción de restricciones ha demostrado ser eficaz para resolver aquellos problemas que son más sencillos o con demanda pequeña de vehículos, pues sus resultados no logran competir con las otras técnicas cuando los casos son más difíciles o más grandes.

3.2. Procedimientos Heurísticos

Se caracterizan por sacrificar en cierta medida la calidad de la solución, a fin de acelerar el proceso de búsqueda de la misma.

3.2.1. Enfoques basados en Búsqueda Local

La idea es ir mejorando la solución actual iterativamente, teniendo en consideración que la solución inicial puede ser generada por una heurística greedy, o aleatoriamente. Además, los vecindarios se generan en base a ligeros cambios en la solución actual, por ejemplo, en [7] se intercambian dos automóviles de la línea de producción, y la selección de los vecindarios se hace al azar.

Es importante mencionar que hay gran diversidad en esta técnica debido al movimiento que se escoja para la creación del vecindario, no obstante, siempre se termina por confirmar que para escapar del óptimo local, lo mejor es asignar un peso a las restricciones no satisfechas, y aumentar dicho valor según sea el caso. Es además, en base a esta misma diversidad, que la técnica es útil tanto para problemas de alta o baja demanda de vehículos.

3.2.2. Heurística Greedy

Se parte de una solución vacía, es decir, una secuencia vacía y luego se agrega el siguiente automóvil a la secuencia existente, paso a paso. Para determinar la unidad que se agrega a la secuencia una función debe seleccionar un automóvil de manera que se violen la menor cantidad de restricciones posibles [8].

Este tipo de heurística resulta útil cuando se combina con cierto grado de aleatoriedad en las asignaciones, y se trabaja con reinicios, pues así se pueden resolver rápidamente las diferentes instancias.

3.2.3. Algoritmos Genéticos

Para este tipo de representación probabilístico, se trabaja con un conjunto de soluciones candidatas que serán consideradas por el algoritmo. Un subconjunto de soluciones de éste último, son combinadas para crear soluciones *hijas*[9]. Además, con el propósito de disminuir la aleatoriedad de la búsqueda, las mejores soluciones se seleccionan por tener probabilidades más altas.

Esta técnica es más eficiente para los problemas sencillos y de baja demanda de automóviles, especialmente si las probabilidades para posibles los cambios de las generaciones son usadas con discreción, ya que si estos valores aumentan, los resultados no terminan siendo tan exitosos como se espera.

3.2.4. Algoritmos de optimización con Colonias de Hormigas

También es un algoritmo probabilístico inspirado en la naturaleza de las hormigas. Inicialmente hay que transformar el problema para poder trabajarlo como un grafo, con el fin de encontrar un camino de coste mínimo. Para lograrlo, las hormigas *artificiales* ponen rastros de feromonas en el camino que utilizan [10], sabiendo que el camino que éstas escojan se basa en una decisión probabilística, que dependerá de la concentración de feromona aportado por la colonia de hormigas a medida que realizan sus recorridos.

Esta heurística es ligeramente superior que la búsqueda local cuando se compara la eficiencia de ambas en instancias con una pequeña cantidad de automóviles, no obstante, las diferencias se

hacen más evidentes a medida que la dificultad del problema aumenta, por lo que resulta difícil de comprar.

3.3. Variantes del problema

Algunas variantes de este problema consideran una representación inversa. Es decir, el modelamiento de las variables se realiza en base a cada automóvil, y no a las casillas de la línea transportadora de producción, de tal forma que los valores que tome cada variable será el número de una de las casillas de la secuencia. Aunque existe una desventaja, y es que los automóviles que tiene las mismas opciones se pueden cambiar, sin violar las restricciones. Además, las posibles asignaciones serían D^D en vez de P^D como en el caso general en que las unidades se agrupan en P clases.

Además, uno de los casos más recientes donde se varía el problema es en un reto pronunciado por franceses y apoyado por el fabricante de automóviles Renault [5], donde se agrega a las restricciones, el cambio de color para las unidades que pasan por la línea de producción.

4. Modelo Matemático

El objetivo del Car sequencing problem es encontrar una solución que satisfaga la mayor cantidad de restricciones o reglas que el pedido en la planta de producción automotriz recibe, para ello, la representación del problema que se considera en esta publicación es el modelo básico de programación de satisfacción de restricciones que fue desarrollado por Dincbas, Simonis y Van Hentenryck [3].

4.1. Parámetros

- C**:** cantidad de opciones.
- P**:** cantidad de clases.
- D demanda total de vehículos a ensamblar.
- D_i**:** demanda parcial, indica cuántas unidades hay de cada clase. De esta forma $\Sigma d_i = D$ con $i = \{1, ..., D\}$.
- C_{ji}**:** si toma valor 1, indica que la opción j debe estar en los automóviles de clase i. Toma valor 0 en caso contrario. Por lo tanto, $C_{ji} \in \{0, 1\}$.
- p_j/q_j**:** son las reglas que debe cumplir la planta de fabricación que especifica cuántas veces puede aparecer la opción p_j como máximo, en cualquier secuencia de q_j vehículos.

4.2. Variables

- x_{it}: variable binaria que toma valor 1 si un vehículo de la clase $i = \{1, ..., P\}$ se encuentra en la posición $t = \{1, ..., D\}$ de la secuencia. Por lo tanto, $x_{ji} \in \{0, 1\}$.
- $l_j(t) = \text{máx}\{1, t + 1 - q_j\}$ es la instancia de inicio del segmento $[l_j(t), t]$ de la secuencia cuya longitud es menor o igual a q_j, siendo $t = \{1, ..., D\}$.

4.3. Restricciones

$$\sum_{j=1}^{p} x_{it} = 1 \qquad \forall t = \{1, ..., D\} \tag{1}$$

$$\sum_{t=1}^{D} x_{it} = d_i \qquad \forall i = \{1, ..., P\} \tag{2}$$

$$\sum_{i=1}^{P} \sum_{k=l_j(t)}^{t} C_{ji} \cdot x_{ik} \leq P_j \qquad \forall j = \{1, ..., C\}; \forall t = \{1, ..., D\} \tag{3}$$

$$x_{ji} \in \{0, 1\} \qquad \forall i = \{1, ..., P\}; \forall t = \{1, ..., D\} \tag{4}$$

La restricción (1) se encarga de regular que en todos los instantes de secuenciación, se asigna una única unidad de producto.

La expresión (2) asegura que todos los vehículos de cada clase, requeridos originalmente, aparecen en la secuencia.

La restricción (3) determina que en todos los segmentos de longitud q_j de la secuencia, el número máximo de veces que puede aparecer la opción j es igual a p_j. Esto, con el propósito de evitar la sobrecarga de las áreas de trabajo en la línea de producción.

Finalmente, (4) corresponde a la naturaleza de las variables.

Cabe destacar que la expresión (3) es la única que puede ser considerada como restricción blanda, pues si bien se espera que se cumplan en la medida de lo posible, el no hacerlo, puede ser tratado fácilmente si el CSP es extendido a CSP Valuado (VSCP) [4]. De esta manera, las restricciones no serán relaciones sino funciones de costo que expresarán el grado de satisfacción para encontrar la mejor solución.

4.4. Función Objetivo

Si el Car sequencing problem es considerado desde un enfoque de decisión, la solución consiste en la búsqueda de una secuencia que satisfaga todas las restricciones de capacidad de las estaciones de ensamblaje, mientras que si se mira desde un punto de vista de un problema de optimización, y por tanto, se extiende el problema a un CSP valuado (VCSP), esto implica encontrar una secuencia de coste mínima, donde la función de coste evalúa las violaciones de las restricciones blandas. En otras palabras, la restricción (3) sería regulada por la Función objetivo de la siguiente forma:

$$min \sum_{i=1}^{P} \sum_{k=l_j(t)}^{t} C_{ji} \cdot x_{ik} > p_j \qquad \forall j = \{1, ..., C\}; \forall t = \{1, ..., D\} \tag{5}$$

5. Representación

Para representar el Car Sequencing Problem en el lenguaje de programación C se usaron básicamente variables enteras, además de arreglos y matrices. Estos últimos dos elementos usan asignación dinámica de memoria en tiempo de ejecución, pues se tuvo en consideración la gran diversidad de datos en las instancias que se probarían una vez desarrollado el algoritmo *Hill climbing Mejor-Mejora* (HC-MM), conocido en inglés como *Hill-Climbing best improvement.*

Considerando la simbología del modelo matemático, de C = cantidad de opciones, P = cantidad de clases, D = demanda, a continuación se detallan los distintos elementos que se usan para la representación del problema.

5.1. Variables Enteras

- **NroAutos** guarda la cantidad de autos que deberán ser fabricados y secuenciados (D).
- **NroOpciones** corresponde a la cantidad de opciones existentes en la planta de fabricación (por ejemplo: radio, aire acondicionado, techo solar, etc.)(C).
- **NroClases** almacena la cantidad de clases que existen, y que se diferencian unas de otras por la cantidad y tipo de opciones que los automóviles de dichas clases deben tener (P).

5.2. Arreglos de una dimensión

- **Solucion** Se encarga de almacenar la secuencia de automóviles, por lo que cada celda contendrá un único número que representará la clase de vehículo que se encuentra en una sección específica de la cinta transportadora. De esta forma, es posible asegurar el cumplimiento de la Restricción (1) definida en el Modelo Matemático.

  El largo del arreglo se establece en base a la cantidad de vehículos que se deben fabricar, es decir, según la demanda D.

Auto 1 - clase C_i	Auto 2 - clase C_i	$\cdots$	Auto D - clase C_i

- **NroMaxAutosPorOpcion y TamBloqueAutosPorOpcion**

  Estos arreglos trabajan en conjunto, ya que almacenan la relación p_j/q_j que se definió en el modelo matemático, y representan las reglas que debe cumplir la planta de fabricación que especifica cuántas veces puede aparecer la opción p_j como máximo, en cualquier secuencia q_j de vehículos. Por tanto, p_j es representado por NroMaxAutosPorOpcion y q_j por TamBloqueAutosPorOpcion, siendo el tamaño de ambos arreglos, definidos por la cantidad de opciones (C, como se indica en el modelo).

NroMaxAutosPorOpcion

Max. numero de autos para Opción 0	$\cdots$	Max. numero de autos para Opción C

TamBloqueAutosPorOpcion

Tamaño del bloque de Opción 0	$\cdots$	Tamaño del bloque de Opción C

- **NroDeAutosEnCadaClase** El largo de este arreglo viene dado por la cantidad de clases P, y como el nombre indica, contiene la cantidad de autos que se deben fabricar de cada tipo de clase, y esto permite asegurar el cumplimiento de la restricción (2) definida en el Modelo Matemático.

Num. de autos de Clase 0	Num. de autos de Clase 1	$\cdots$	Num. de autos de Clase P

5.3. Arreglos de dos dimensiones

- **Matriz** Las filas de Matriz representan las clases P del problema, mientras que las columnas indican las opciones C, por tanto, su tamaño viene dado por $C \times P$. Este arreglo guarda 0 o 1, para indicar si la clase P_i tiene o no a la opción C_j, respectivamente.

  Para comprender mejor la estructura, a continuación se muestra un ejemplo de un problema con 3 opciones, y de 4 clases, siendo la clase 0 (primera fila de la matriz), aquella que solo posee la opción 0, y por lo mismo, su fila solo tiene un número uno. La clase 1, que posee las opciones 0 y 1, la clase 2, que tiene las opciones 0, 1, 2 y por último la clase 3, que solo tiene a la opción 2.

	Opciones		
Clases	0	1	2
0	1	0	0
1	1	1	0
2	1	1	1
3	0	0	1

Tabla 1: Ejemplo gráfico de Matriz que representa un problema de 4 clases de vehículos y 3 opciones.

- **MatrizEvaluacion**

  Este arreglo es creado para poder cuantificar la cantidad de restricciones que son violadas por una posible solución. Por lo tanto, es usada para evaluar cuantas restricciones de la expresión (3) del modelo, no son satisfechas.

  Esta matriz es creada en base a la secuencia de autos que se define a partir del arreglo *Solucion* y de *Matriz* previamente definidos, por tanto, su tamaño está dado por la cantidad de automóviles D y la cantidad de opciones C.

  Utilizando el mismo ejemplo anterior, y considerando una demanda de 8 vehículos, se podría tener una solución como la que sigue:

	Opciones		
Auto	0	1	2
0	1	0	0
1	1	1	0
2	1	1	1
3	0	0	1
4	0	0	1
5	1	1	1
6	1	1	0
7	1	1	0

Tabla 2: Ejemplo gráfico de MatrizEvaluacion que representa una posible solucion de un problema de 4 clases de vehículos,3 opciones, y 8 vehículos de demanda, donde específicamente se requieren 3 autos de Clase 1, 2 autos de Clase 2, 2 autos de Clase 3 y solo un auto de Clase 0.

En donde la primera fila indica que el primer auto a fabricar es de clase 0, ya que solo la opción 0 tiene un 1 en la celda. La segunda fila dice que el segundo auto a crear es de clase 1 ya que hay dos números 1 en la fila, en las primeras posiciones. La fila siguiente indica que el tercer auto a fabricar es de la clase 2, ya que las opciones 0 1 y 2, tienen números 1 en la matriz.

Considerando además que las restricciones p_j/q_j para cada opción son:

Opción 0	1/2
Opción 1	3/4
Opción 2	1/3

Tabla 3: Restricciones p_j/q_j para el ejemplo de un problema de 4 clases de vehículos,3 opciones, y 8 vehículos de demanda.

Es posible observar que en la primera opción (primera columna) cuyos bloques de revisión se componen de 2 casillas, se violan 2 restricciones, ya que tanto el primer rectángulo sombreado, como el último, tienen cada uno, más 1 de los requeridos.

Por otro lado, la opción 1 (segunda columna), trabaja con bloques de 4 celdas, y como máximo, estos bloques deben tener solo 3 automóviles. De esta forma, si se miran los rectángulos de esa columna, es posible ver que no se están violando las restricciones.

	Opciones		
Auto	0	1	2
0	1	0	0
1	1	1	0
2	1	1	1
3	0	0	1
4	0	0	1
5	1	1	1
6	1	1	0
7	1	1	0

Tabla 4: Ejemplo gráfico de MatrizEvaluacion con algunos bloques de revisión de restricciones sombreados, que representa una posible solucion de un problema de 4 clases de vehículos,3 opciones, y 8 vehículos de demanda, donde específicamente se requieren 3 autos de Clase 1, 2 autos de Clase 2, 2 autos de Clase 3 y solo un auto de Clase 0.

6. Descripción del algoritmo

La técnica implementada para la resolución del problema es *Hill-Climbing Mejor Mejora* (HC-MM), un algoritmo iterativo que inicia con una solución que puede ser obtenida de diversas formas, y que luego intenta encontrar una mejor, variando un único elemento de la solución, para crear un vecindario de posibles soluciones. Si dentro del vecindario hay una solución de mejor calidad, lo que viene determinado por la cantidad de restricciones que se violan con dicha solución, se realiza otro cambio a esta nueva solución, repitiendo el proceso hasta que ya no sea posible encontrar una mejora, lo que se conoce como óptimo local.

Si bien es cierto que los algoritmos de búsqueda local no garantizan encontrar la solución óptima, el Car Sequencing problem se trata de un CSP, y por tanto es posible saber que el óptimo se alcanza cuando la cantidad de restricciones violadas es cero.

La estructura básica del algoritmo HC-MM que se desarrolló se presenta en **Algoritmo 1**, donde se señala que la solución inicial se obtiene a partir de un algoritmo Greedy, y utiliza el nombre de las variables que fueron definidas en el apartado anterior.

Algoritmo 1 Hill-Climbing Mejor Mejora

Entrada: Solucion (vacía), Matriz, NroAutos, NroOpciones, NroClases.
Salida: Solucion (completa).

```
1: Local ← Falso
2: Sc ← Selecciona punto de partida a partir del algoritmo Greedy
3: Mientras (Local = Falso ) Hacer
4:    Sn ← Seleccionar mejor solución del vecindario
5:    Si Calidad Sn > Sc Entonces
6:       Sc ← Sn
7:    Si no
8:       Local ← Cierto
9:    Fin Si
10: Fin Mientras
```

En la línea 1 se inicializa una variable que permite identificar el cuándo se ha alcanzado el óptimo local, lo que indica que se debe detener el algoritmo.

La línea siguiente señala que la solución, que inicialmente está vacía, será creada en base a un algoritmo Greedy. De esta forma, se irán agregando elementos a la solución, donde cada uno de éstos, se elige de un conjunto de posibles candidatos, y para ello, se debe contar con una función de evaluación de costos para determinar que tanto afecta la adición de ese determinado componente en una cierta posición.

El pseudo-código del algoritmo Greeedy, se presenta en **Algoritmo 2**.

Algoritmo 2 Greedy

Entrada: Solucion (vacía), Matriz, NroAutos, NroOpciones, NroClases, NroDeAutosEnCadaClase.
Salida: Solucion (completa).

```
1: ViolacionesSolucionActual ← 0
2: Para i < NroAutos Hacer
3:    j ← 0
4:    Mientras queden autos de la clase j por crear Hacer
5:       Agregar auto de la clase i a Solucion
6:       ViolacionesSolucionNueva ← Evaluar restricciones violadas de Solucion
7:       Si (ViolacionesSolucionNueva - ViolacionesSolucionActual) ≤ 1 Entonces
8:          Restar cantidad de autos que quedan por crear de la clase j
9:       Si no
10:         Probar asignar una clase distinta
11:         Si se han probado todas las clases Entonces
12:            Asignar la clase j por defecto
13:         Fin Si
14:      Fin Si
15:      Cambiar la clase j que se asignará en la siguiente iteración
16:   Fin Mientras
17: Fin Para
```

Como se mencionó con anterioridad, tras agregar un elemento a la solución, ésta debe ser evaluada para determinar si adicionar ese componente en un determinado momento es conveniente o no.

La Evaluación de la solución consiste básicamente en recorrer MatrizEvaluacion por columnas, de esta forma es posible identificar los bloques con los que cada opción trabaja, y así, contabilizar el número de 1 que hay en éstos, para identificar si el bloque está violando una restricción. Cabe mencionar que la mejor calidad viene dada por aquella solución que tiene menor cantidad de restricciones No satisfechas. El pseudo-código se muestra en **Algoritmo 3**

Algoritmo 3 EvaluarSolucion

Entrada: Solucion, Matriz, NroAutos, NroOpciones, NroMaxAutosPorOpcion, TamBloqueAutosPorOpcion.
Salida: Cantidad de restricciones violadas por Solucion.

```
1: Crear MatrizEvaluacion
2: CantCeldas ← 0
3: ContadorUnos ← 0
4: ContadorViolaciones ← 0
5: Para i < NroOpciones Hacer
6:   Para j < NroAutos Hacer
7:     Si CantCeldas < TamBloqueAutosPorOpcion[i] Entonces
8:       Si MatrizEvaluacion[i][j] = 1 Entonces
9:         Aumentar ContadorUnos
10:      Fin Si
11:      Aumentar CantCeldas
12:    Si no
13:
14:      Si ContadorUnos > NroMaxAutosPorOpcion[i] Entonces
15:        Aumentar ContadorViolaciones
16:      Fin Si
17:      CantCeldas ← 0
18:      ContadorUnos ← 0
19:    Fin Si
20:  Fin Para
21: Fin Para
```

Otro factor importante a considerar en HC-MM es el movimiento que permite la creación del vecindario. Debido a la representación del problema, una buena técnica es *Swap*.

Para esta implementación, inicialmente se debe obtener un número aleatorio entre 0 y el Número de autos a secuenciar. De esta forma, cuando se vaya recorriendo el arreglo de la solución dentro de un ciclo, el valor de la casilla, cuyo índice viene dada por la iteración del ciclo, será intercambiado por el valor de la casilla que tiene un índice igual al número aleatorio.

Considerando que cada vecino creado debe ser evaluado para obtener su calidad, y por tanto, se debe conocer la secuencia de autos que entrega ese valor, la memoria utilizada para almacenar toda esa información sería excesiva.

Por lo mismo, para evitar este inconveniente, a lo largo de la creación del vecindario se va guardando en un arreglo auxiliar, el vecino que hasta el momento ha sido el mejor, y éste es actualizado cuando un nuevo vecino creado tenga mejor calidad. De esta manera, solo se conoce

la mejor solución que hay hasta el momento en el vecindario que se está creando, y que se va actualizando hasta que se termine de crear todos los vecinos respectivos.

El pseudo-código está disponible en **Algoritmo 4: MejorSolucionDelVecindario**

Algoritmo 4 MejorSolucionDelVecindario

Entrada: Solucion, NroAutos.
Salida: SolucionMejorVecino.

```
1: RestricionesVioladasSolucion ← Evaluar Solucion
2: SolucionMejorVecino ← Solucion
3: Para i<NroAutos Hacer
4:    Obtener número aleatorio
5:    A Solucion, intercambiar el auto de la casilla i, por el que se encuentra en la casilla cuyo
      índice viene dado por el número aleatorio
6:    RestricionesVioladasNuevaSolucion ← Evaluar NuevaSolucion
7:    Si RestricionesVioladasNuevaSolucion < RestricionesVioladasSolucion Entonces
8:       Actualizar SolucionMejorVecino
9:    Fin Si
10: Fin Para
```

7. Experimentos

La tabla siguiente señala las características del computador que fue utilizado para realizar las pruebas del algoritmo.

Procesador	AMD Athlon(tm) X2 Dual-Core Ql-65 2.10GHz
RAM	2.75 G
Opción Sistema Operativo	Fedora 20 64 bits

Tabla 5: Tabla resumen de las características del computador utilizado para la prueba del algoritmo.

En primer lugar, se realiza la lectura del archivo que contiene la información relevante del problema, y se crean las variables respectivas para su almacenamiento. Si bien las instancias fueron escogidas al azar, se usaron todos los posibles valores de cantidad de automóviles, siendo éstos, 200, 300 y 400. Además, considerando que las instancias tenían el mismo número de opciones, se buscó variar tanto la cantidad de clases, con el fin de comparar los resultados, por categoría de autos y clases.

Es importante notar además, que como el algoritmo Greedy trabaja con un valor aleatorio para la creación del vecindario, la ejecución de la misma instancia fue repetida tres veces, de modo de poder analizar que tanto variaban los resultados en base a dicha aleatoriedad. Por lo mismo, de las 33 pruebas que se realizaron, solo se utilizaron 11 instancias, cuya información se resume en la siguiente tabla.

	Valores		
Instancia	Automóviles	Opciones	Clases
1	200	5	25
2	200	5	24
3	200	5	20
4	200	5	19
5	300	5	25
6	300	5	24
7	300	5	23
8	300	5	19
9	400	5	25
10	400	5	24
11	400	5	21

Tabla 6: Características de las 11 instancias probadas en el algoritmo.

8. Resultados

Con la finalidad de analizar que tanto varían los resultados, en base a la cantidad de autos y de clases, se aprovecharon las repeticiones que se realizan a una misma instancia para poder calcular los promedios de los resultados de las otras variables que se van modificando a lo largo de la experimentación.

La siguiente tabla muestra los resultados obtenidos, donde se agrupan las 3 repeticiones para cada instancia. Cabe destacar además, que el tiempo de ejecución del algoritmo Greedy, se incluye en el tiempo de Ejecución del HC-MM.

Automóviles	Opciones	Clases	Violaciones Greedy	Tiempo Greedy (miliseg.)	Violaciones HC-MM	Tiempo HC-MM (miliseg.)	Iteraciones
200	5	25	85	115,18	39	481,88	31
				114,54	44	444,86	28
				113,13	42	455,38	29
200	5	24	92	117,71	38	532,25	35
				115,7	50	466,91	30
				116,83	43	563,53	38
200	5	20	81	89,06	31	502,85	35
				91,35	28	527,32	37
				89,44	33	452,04	31
200	5	19	84	92,21	39	469,46	32
				93,04	35	515,47	36
				93,04	43	401,18	26
300	5	25	119	262,46	44	1.632,10	52
				263,11	54	1.434,61	45
				267,23	59	1.309,97	40
300	5	24	137	256,7	55	1.809,38	59
				255,42	58	1.570,89	50
				256,67	52	1.702,82	55
300	5	23	126	257,37	57	1.383,48	43
				253	41	1.746,57	57
				256,21	41	1.772,71	58
300	5	19	129	205,04	43	1.676,74	55
				205,31	44	1.663,28	55
				203,17	41	1.780,90	60
400	5	25	158	516,16	59	3.602,72	67
				515,79	50	3.786,10	71
				512,72	60	3.326,90	61
400	5	24	174	467,89	60	3.816,23	72
				470,99	48	4.223,05	81
				469,58	52	4.368,23	83
400	5	21	181	378,66	70	3.781,49	73
				374,47	63	4.072,88	79
				380,44	63	3.984,19	78

Tabla 7: Resultados obtenidos en la experimentacion, donde las 11 instancias fueron ejecutadas 3 veces, para ver los efectos de la aleatoriedad del HC-MM en la creación del vecindario.

8.1. Análisis de iteraciones

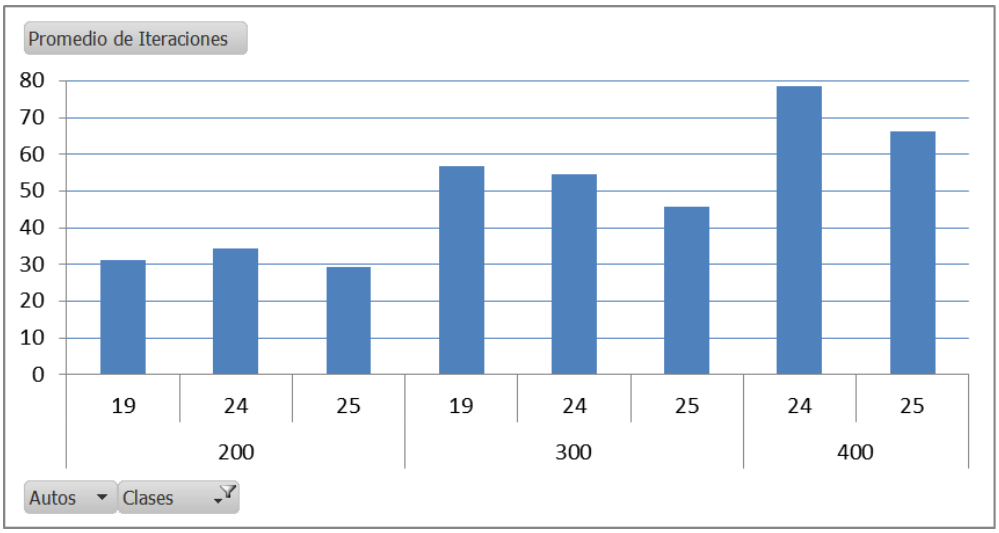

Figura 2: Gráfico comparativo del número promedio de iteraciones de las 3 repeticiones de algunas instancias, agrupados según número de autos (200, 300 y 400) y número de clases (19, 24, 25).

Es fácil observar que a medida que aumenta la cantidad de autos, también lo hará la cantidad de iteraciones que realiza el HC-MM, no obstante, para una misma cantidad de vehículos, a medida que aumentan las clases, disminuyen las iteraciones.

8.2. Análisis de cantidad de violaciones de las restricciones blandas

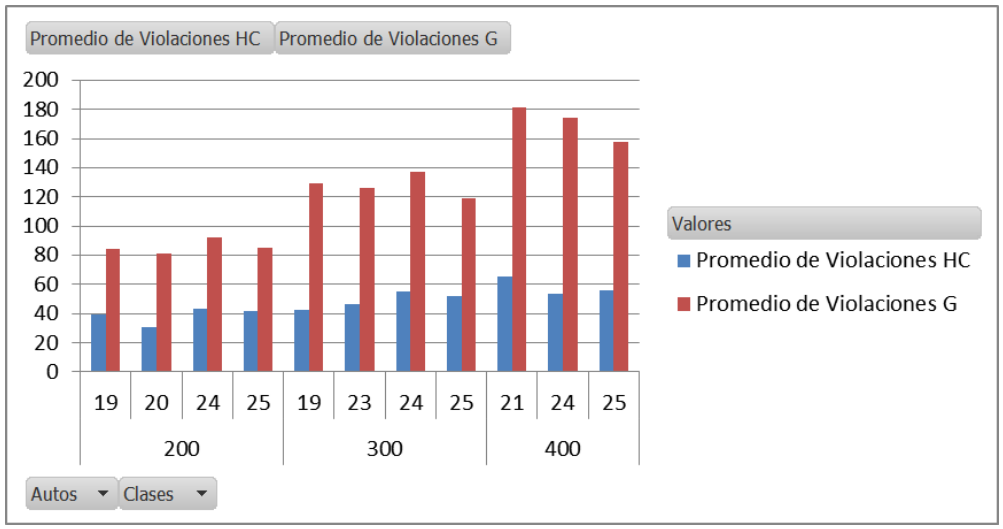

Figura 3: Gráfico comparativo que promedia el número de restricciones violadas de las 3 repeticiones de las instancias, agrupados según número de autos (200, 300 y 400) y número de clases (19, 24, 25).

Nuevamente a mediada que aumentan los vehículos a secuenciar, lo harán la cantidad de violaciones de las restricciones, sin embargo, se puede ver que el algoritmo Greedy (barras rojas) tiene un crecimiento mayor, en comparación al que tiene el HC-MM. Por otro lado, con respecto a las clases, no es posible identificar algún patrón.

Además, considerando la diferencia entre la cantidad de restricciones violadas entre la Solución inicial obtenida por Greedy, y la final, que entrega HC-MM, es posible afirmar que la combinación de ambos algoritmos es más conveniente si la cantidad de autos es mayor, ya que la reducción de las restricciones violadas es mayor para esos casos.

8.3. Análisis de tiempos de ejecución

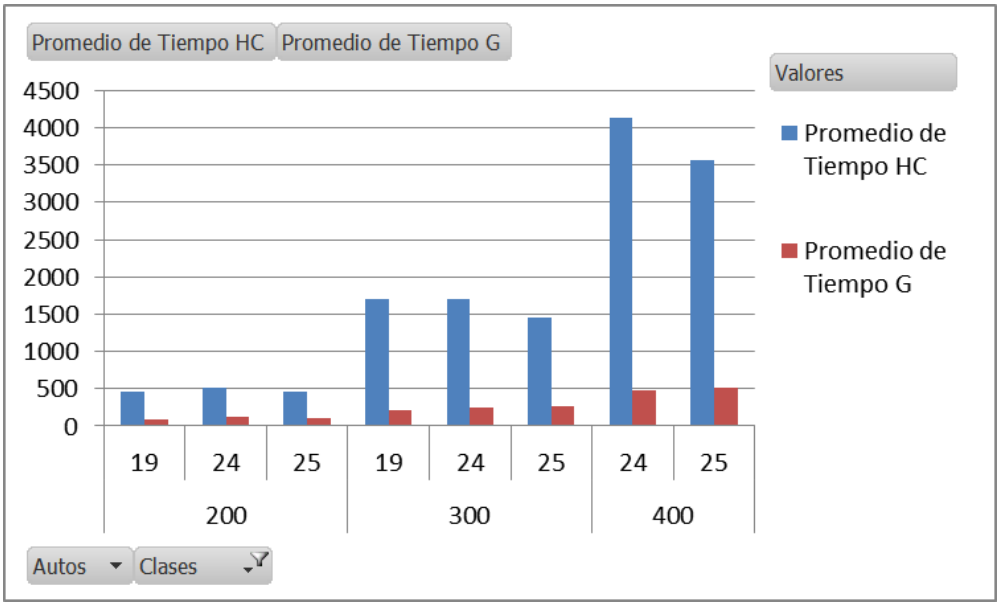

Figura 4: Gráfico comparativo que promedia el tiempo de ejecución de HC-MM y Greedy de las 3 repeticiones de algunas instancias, agrupados según número de autos (200, 300 y 400) y número de clases (19, 24, 25).

En este caso, el aumento tiempo de ejecución del Algoritmo Greedy, es más constante que para HC-MM, y aumenta en base a la cantidad de autos, pero disminuye a mediad que aumentan las clases. Esto viene directamente relacionado con las iteraciones realizadas, ya que como se indica en el gráfico uno, a mediad que aumentan las clases, las cantidad de iteraciones disminuye, por lo que coincide con el tiempo de ejecución.

8.4. Análisis de la influencia de la aleatoriedad en las repeticiones de las instancias

Para evitar la sobrecarga de información visual en el siguiente análisis, se opta por mostrar solo 2 gráficos para una misma cantidad de autos. Cada uno de estos gráficos contiene las 3 repeticiones que se realizaron a una misma instancia, para que sea posible comparar que tan grande es el efecto de la aleatoriedad del movimiento Swap que se realiza al crear el vecindario.

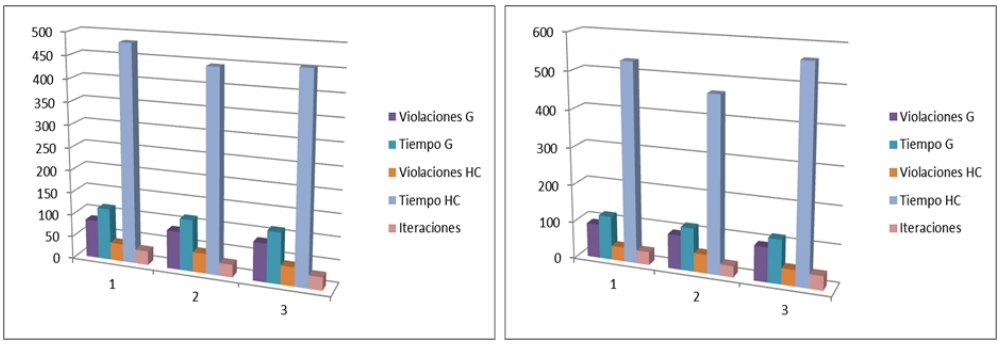

Figura 5: Gráficos comparativos que señalan los resultados obtenidos al repetir las instancias, ambos, con una demanda de 200 autos y 5 opciones. La figura de la izquierda es un problema de 25 clases. A la derecha, una instancia de 24 clases.

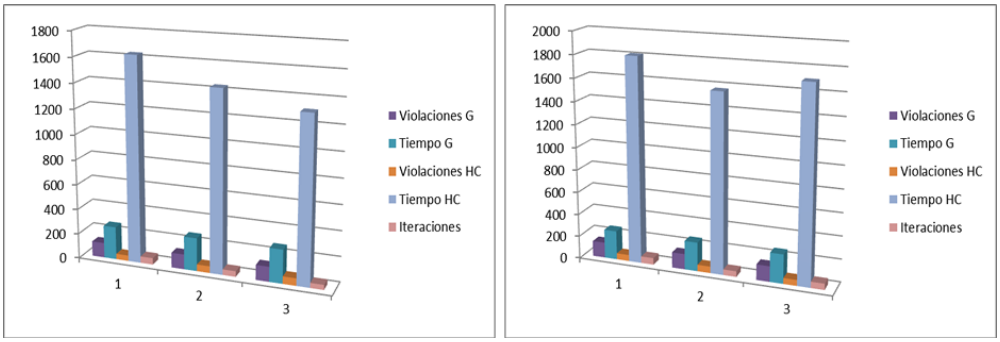

Figura 6: Gráficos comparativos que señalan los resultados obtenidos al repetir las instancias, ambos, con una demanda de 300 autos y 5 opciones. La figura de la izquierda es un problema de 25 clases. A la derecha, una instancia de 24 clases.

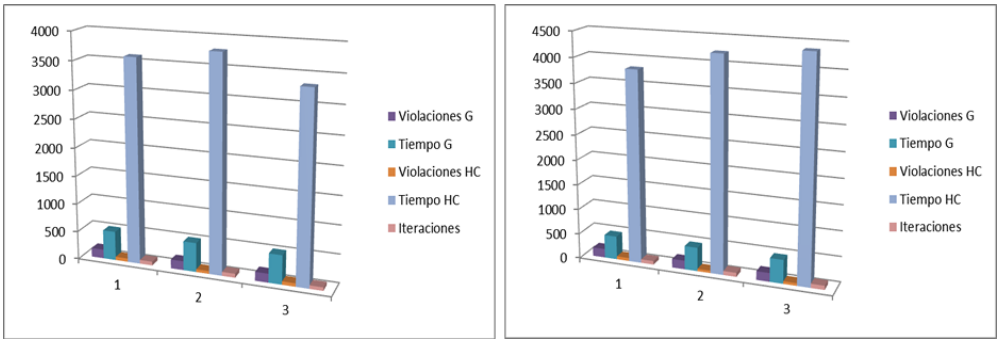

Figura 7: Gráficos comparativos que señalan los resultados obtenidos al repetir las instancias, ambos, con una demanda de 400 autos y 5 opciones. La figura de la izquierda es un problema de 25 clases. A la derecha, una instancia de 24 clases.

Como es posible percibir, un elemento que varía es el tiempo del HC-MM (barra más grande), sin embargo, esos valores no presentan grandes diferencias entre una repetición y otra. Así, es posible confirmar que el movimiento escogido, y por tanto, el vecindario que se crea, no es tan sensible a la aleatoriedad, y por lo mismo, puede ser utilizado de forma segura, pues independiente de las repeticiones que se realicen de una instancia, el comportamiento del algoritmo y sus resultados son similares entre si.

9. Conclusiones

Además, se ha llevado a cabo una análisis de la literatura científica sobre los diversos enfoques que se han desarrollado para hacer frente al Car Sequencing Problem, donde se pudo apreciar que la mayoría de los trabajos consideran que la secuencia tiene un principio y final, y que se conoce la cantidad existente de unidades o vehículos a secuenciar. Por lo mismo, un buen trabajo a futuro sería desarrollar una metodología que permita resolver el problema con una mejor aproximación a la situación real, de tal forma que los productos se consideren para su secuenciación en base a su disponibilidad y no a un número fijado desde el comienzo.

Tras la revisión de las diversas técnicas que se han desarrollado para la resolución de este problema, fue posible identificar bajo qué parámetros eran recomendables de usar. Por ejemplo, las técnicas de Procedimientos exactos y de Colonias de Hormigas son más eficientes cuando se trabajan con una demanda pequeñas de vehículos. En cambio, para los casos de Búsqueda Local, Greedy y Algoritmos Genéticos, no importa la dificultad del problema, pues los resultados tenderán a ser exitosos. Aunque es importante destacar que las dos últimas heurísticas mencionadas trabajan mejor bajo ciertas características. Por ejemplo, para el algoritmo Greedy es conveniente obtener valores al azar para la asignación, ya que de modo contrario, y tal como se evidencia en la experimentación, las restricciones no satisfechas aumentan considerablemente, si la demanda también lo hace. En cambio, los Algoritmos Genéticos trabajan mejor generalmente si las probabilidades de que las generaciones cambien, no es tan alto, pues de esta forma se mantiene a lo largo del tiempo aquellos individuos de buena calidad.

Por otro lado, se desarrolló una implementación del Algoritmo de búsqueda local Hill-Climbing Mejor Mejora, junto a un algoritmo Greedy para la obtención de la solución inicial. Ésta es mejorada a partir de reparaciones iterativas, hasta que se alcanza el óptimo local. No obstante, para el caso de problemas CSP, como lo es el Car Sequencing Problem, es posible identificar que el óptimo global se logra cuando la cantidad de restricciones no satisfechas del problema es nula. Sin embargo, a pesar de las 33 pruebas que se realizaron, el algoritmo convergía a un óptimo local antes de alcanzar el óptimo global esperado.

Del análisis de la experimentación, es importante rescatar algunos hechos destacables:

- A mayor cantidad de autos, HC-MM realiza más iteraciones antes de alcanzar un óptimo local.
- Para una cantidad de autos constantes, si la cantidad de clases aumenta, las iteraciones disminuyen.
- Si aumenta la cantidad de autos, las restricciones violadas de la solución final aumentan de manera suave. No así las restricciones violadas por la solución inicial entregada por el Algoritmo Greedy, que aumentan en mayor medida entre las distintas demandas.
- Si la demanda de vehículos es mayor, el tiempo de ejecución de HC-MM aumenta en mayor medida que el del Algoritmo Greedy.

- La aleatoriedad utilizada en el Algoritmo HC-MM para la creación del vecindario, no afecta de manera significativa el comportamiento del mismo, ya que los resultados comparados mostraron ser muy similares entre las 3 repeticiones de cada instancia, por tanto, es posible confirmar que el movimiento Swap escogido, es seguro de utilizar.

Finalmente, fue posible comprobar el gran interés que existe sobre el problema debido a la cantidad de investigaciones que se han realizado desde la década del ochenta.

10. Bibliografía

[1] *Parello, B.D., W.C. Kabat and L. Wos (1986) "Job-shop scheduling using automated reasoning"Journal of Automated Reasoning. 1-42.*

[2] *Aigbedo, H. (2004). Analysis of parts requirements variance for a JIT supply chain. International Journal of Production Research, Vol. 42.*

[3] *M. Dincbas, H. Simonis, and P. Van Hentenryck. Solving the car-sequencing problem in constraint logic programming. In Proceedings ECAI-88, pages 290-295, 1988.*

[4] *T.Schiex, H. Fargier. Valued constraint satisfaction problema: hard and easy problems. IJCA-95. Montreal, Canada, Agosto 1995.*

[5] *A. Nguyen. Challenge ROADEF'2005: Car sequencing problem. Disponible en http://challenge.roadef.org/2005/en/. Última vez visitada el 12 de Mayo, 2014.*

[6] *M. Prandtstetter and G. R. Raidl. An integer linear programming approach and a hybrid variable neighborhood search for the car sequencing problem. European Journal of Operational Research, 2008.*

[7] *M. Puchta and J. Gottlieb. Solving car sequencing problems by local optimization. In Proceedings of the Applications of Evolutionary Computing on EvoWorkshops 2002, p: 132-142, Londres, Reino Unido, 2002. Primavera.*

[8] *J. Gottlieb, M. Puchta, and C. Solnon. A study of greedy, local search, and ant colony optimization approaches for car sequencing problems. In G. R. Raidl et al., editors, Applications of Evolutionary Computing, volume 2611, p: 246-257, 2003.*

[9] *J. Cheng, Y. Lu, G. Puskorius, S. Bergeon, and J. Xiao. Vehicle sequencing based on evolutionary computation. In Proceedings of 1999 Congress on Evolutionary Computation, Washington D. C., volumen 2, p: 1207-1214, 1999.*

[10] *C. Solnon. Solving permutation constraint satisfaction problems with artificial ants. In W. Horn, editor, Proceedings of the 14th European Conference on Artificial Intelligence, p: 118-122, Amsterdam, 2000.*